# Natural Language Query Database Schemas

## Overview

This document describes the database schemas for the Natural Language Query system, designed to support AI-powered natural language to SQL conversion, query processing, result management, and continuous improvement through user feedback.

## Schema Architecture

### 🏗️ Design Principles

* **NLP Workflow Support**: Complete pipeline from natural language to SQL to results
* **Performance Optimized**: Strategic indexes for fast query processing and analysis
* **User Feedback Integration**: Built-in feedback collection for ML model improvement
* **Template System**: Reusable query patterns for common use cases
* **Audit Trail**: Complete tracking of query processing and user interactions
* **Extensible Design**: JSON fields for flexible metadata and entity storage

## Tables

### 💬 nl\_queries

**Purpose**: Tracks user natural language queries through the complete processing pipeline from input to results, including performance metrics and user feedback.

**Columns**:

id SERIAL PRIMARY KEY  
query TEXT NOT NULL  
user\_id INTEGER NOT NULL REFERENCES users(id)  
status enum\_nl\_queries\_status DEFAULT 'pending'  
query\_type enum\_nl\_queries\_query\_type  
intent VARCHAR(255)  
entities JSONB DEFAULT '{}'  
sql\_query TEXT  
results JSONB DEFAULT '{}'  
result\_count INTEGER  
execution\_time DECIMAL(15,2)  
confidence DECIMAL(15,2)  
feedback enum\_nl\_queries\_feedback  
feedback\_comment TEXT  
error\_message TEXT  
metadata JSONB DEFAULT '{}'  
created\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL  
updated\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL

**Indexes**: - idx\_nl\_queries\_user\_id - User query lookups - idx\_nl\_queries\_status - Status-based filtering - idx\_nl\_queries\_query\_type - Query type analysis - idx\_nl\_queries\_intent - Intent-based searches - idx\_nl\_queries\_created\_at - Time-based queries - idx\_nl\_queries\_confidence - Confidence filtering - idx\_nl\_queries\_feedback - Feedback analysis - idx\_nl\_queries\_execution\_time - Performance analysis - idx\_nl\_queries\_user\_status - User + status composite - idx\_nl\_queries\_status\_created - Status + time composite - idx\_nl\_queries\_query\_type\_status - Type + status composite

**Enums**: - status: pending, processing, completed, failed, cancelled - query\_type: asset\_search, cost\_analysis, vulnerability\_report, compliance\_check, lifecycle\_planning, operational\_metrics, risk\_assessment, general\_query - feedback: helpful, not\_helpful, partially\_helpful, incorrect, needs\_improvement

### 📋 query\_templates

**Purpose**: Stores reusable query templates with parameters for common natural language query patterns, enabling consistent and optimized query generation.

**Columns**:

id SERIAL PRIMARY KEY  
name VARCHAR(100) NOT NULL  
description TEXT  
query\_text TEXT NOT NULL  
category VARCHAR(50)  
parameters JSONB DEFAULT '{}'  
created\_by INTEGER REFERENCES users(id)  
created\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL  
updated\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL

**Indexes**: - idx\_query\_templates\_name - Name-based lookups - idx\_query\_templates\_category - Category filtering - idx\_query\_templates\_created\_by - Creator lookups - idx\_query\_templates\_created\_at - Time-based queries - idx\_query\_templates\_category\_created - Category + time composite - idx\_query\_templates\_created\_by\_name - Creator + name composite

**Constraints**: - query\_templates\_name\_unique - Unique template names

## Natural Language Processing Workflow

### 🔄 Query Processing Pipeline

1. **Input**: User submits natural language query
2. **Classification**: Query type and intent identification
3. **Entity Extraction**: Extract structured entities from text
4. **SQL Generation**: Convert to SQL using templates or AI
5. **Execution**: Run generated SQL query
6. **Results**: Store and format results
7. **Feedback**: Collect user feedback for improvement

### 📊 Status Lifecycle

pending → processing → completed  
 → failed  
 → cancelled

### 🎯 Query Types and Use Cases

#### **Asset Search** (asset\_search)

* Find assets by type, location, or properties
* Asset inventory queries
* Hardware/software discovery

#### **Cost Analysis** (cost\_analysis)

* Financial reporting and analysis
* Budget tracking and forecasting
* Cost optimization queries

#### **Vulnerability Report** (vulnerability\_report)

* Security vulnerability analysis
* Risk assessment queries
* Compliance reporting

#### **Compliance Check** (compliance\_check)

* Regulatory compliance verification
* Policy adherence monitoring
* Audit preparation queries

#### **Lifecycle Planning** (lifecycle\_planning)

* Asset lifecycle management
* Replacement planning
* Warranty and EOL tracking

#### **Operational Metrics** (operational\_metrics)

* Performance monitoring
* Utilization analysis
* Operational efficiency queries

#### **Risk Assessment** (risk\_assessment)

* Risk analysis and scoring
* Threat assessment
* Security posture evaluation

#### **General Query** (general\_query)

* Miscellaneous queries
* Custom analysis requests
* Ad-hoc reporting

## Template System

### 📝 Template Structure

Templates use parameterized SQL with JSON schema for parameter validation:

{  
 "name": "asset\_search\_by\_type",  
 "description": "Search for assets by type with optional filters",  
 "query\_text": "SELECT \* FROM assets WHERE asset\_type = $1 AND ($2 IS NULL OR hostname ILIKE $2)",  
 "category": "asset\_search",  
 "parameters": {  
 "type": "object",  
 "properties": {  
 "asset\_type": {"type": "string"},  
 "hostname\_filter": {"type": "string", "optional": true}  
 }  
 }  
}

### 🔧 Template Categories

* **asset\_search**: Asset discovery and inventory
* **cost\_analysis**: Financial analysis and reporting
* **vulnerability\_report**: Security and vulnerability queries
* **compliance\_check**: Compliance and audit queries
* **lifecycle\_planning**: Asset lifecycle management
* **operational\_metrics**: Performance and utilization
* **risk\_assessment**: Risk and security analysis

## Performance Optimization

### 📈 Index Strategy

1. **User Queries**: Fast user-specific query retrieval
2. **Status Filtering**: Efficient processing pipeline management
3. **Type Analysis**: Quick query categorization and routing
4. **Time-based Queries**: Historical analysis and reporting
5. **Feedback Analysis**: ML model improvement insights
6. **Template Management**: Fast template lookup and categorization

### 🚀 Query Performance Features

* **Composite Indexes**: Support complex filtering scenarios
* **JSONB Optimization**: Efficient JSON operations for entities and results
* **Status Indexing**: Fast processing pipeline queries
* **Time-series Support**: Optimized for temporal analysis

## Data Storage and Analysis

### 📊 Entity Storage

The entities JSONB field stores extracted entities in structured format:

{  
 "asset\_types": ["server", "workstation"],  
 "date\_ranges": [{"start": "2024-01-01", "end": "2024-12-31"}],  
 "locations": ["datacenter-1", "office-ny"],  
 "cost\_ranges": [{"min": 1000, "max": 50000}],  
 "vendors": ["Dell", "HP", "Cisco"]  
}

### 📈 Results Storage

The results JSONB field stores query results and metadata:

{  
 "data": [...],  
 "summary": {  
 "total\_records": 150,  
 "processing\_time": 0.45,  
 "cache\_hit": false  
 },  
 "visualization": {  
 "chart\_type": "bar",  
 "x\_axis": "cost\_type",  
 "y\_axis": "total\_amount"  
 }  
}

### 🔍 Metadata Storage

The metadata JSONB field stores processing information:

{  
 "nlp\_model": "gpt-4",  
 "processing\_version": "1.2.0",  
 "template\_used": "cost\_analysis\_by\_period",  
 "parameters": {"start\_date": "2024-01-01", "end\_date": "2024-12-31"},  
 "cache\_key": "cost\_analysis\_2024\_hash123",  
 "user\_context": {"department": "IT", "role": "analyst"}  
}

## Analytics and Reporting

### 📊 Query Analytics

* **Usage Patterns**: Most common query types and intents
* **Performance Metrics**: Execution time analysis and optimization
* **Success Rates**: Query completion and failure analysis
* **User Satisfaction**: Feedback analysis and improvement tracking

### 🎯 Template Analytics

* **Template Usage**: Most popular templates and categories
* **Parameter Analysis**: Common parameter patterns and values
* **Template Performance**: Execution time and success rates
* **Template Evolution**: Version tracking and improvement history

## Security and Privacy

### 🔒 Data Protection

* **Query Sanitization**: Prevent SQL injection through parameterization
* **Result Filtering**: Apply user permissions to query results
* **Audit Logging**: Complete query and access tracking
* **Data Masking**: Sensitive data protection in results

### 👤 User Context

* **Permission Integration**: Respect user access controls
* **Department Filtering**: Scope queries to user’s department
* **Role-based Access**: Different query capabilities by role
* **Data Sovereignty**: Respect data access boundaries

## Integration Points

### 🔗 System Integration

* **Asset Management**: Query asset data and relationships
* **Cost Analysis**: Financial data analysis and reporting
* **Vulnerability Management**: Security data queries
* **User Management**: Authentication and authorization
* **Audit System**: Query logging and compliance

### 🤖 AI/ML Integration

* **NLP Models**: Intent classification and entity extraction
* **Query Generation**: Natural language to SQL conversion
* **Result Ranking**: Relevance scoring and optimization
* **Feedback Learning**: Continuous model improvement
* **Template Suggestion**: Intelligent template recommendations

## Best Practices

### 📋 Development Guidelines

1. **Parameterized Queries**: Always use parameterized SQL in templates
2. **Input Validation**: Validate all user inputs and parameters
3. **Error Handling**: Comprehensive error capture and logging
4. **Performance Monitoring**: Track execution times and optimize
5. **User Feedback**: Encourage and analyze user feedback

### 🎯 Operational Guidelines

1. **Template Management**: Regular template review and optimization
2. **Performance Monitoring**: Monitor query performance and bottlenecks
3. **Feedback Analysis**: Regular analysis of user feedback for improvements
4. **Security Audits**: Regular security reviews of query processing
5. **Data Retention**: Implement appropriate data retention policies

## Testing and Validation

### 🧪 Schema Testing

# Test schema structure and exports  
npm run test:nl-query-schemas  
  
# Generate migration SQL  
npm run generate:nl-query-migration

### 📊 Query Testing

* **Template Validation**: Test all query templates with sample data
* **Performance Testing**: Validate query performance under load
* **Security Testing**: Test for SQL injection and access control
* **Integration Testing**: Validate with connected systems

This schema design provides a comprehensive foundation for natural language query processing with built-in support for machine learning, user feedback, and continuous improvement.