🚀 AI Judge - Hybrid Database Setup

Architecture Overview

```
┌─────────────────┐
│   Frontend      │  Next.js (React)
│   Port: 3000    │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│   Backend API   │  FastAPI (Python)
│   Port: 8000    │
└────────┬────────┘
         │
    ┌────┴────┬──────────┬──────────┐
    ▼         ▼          ▼          ▼
┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐
│PostgreSQL│ │ Neo4j   │ │ Redis   │ │  Groq   │
│Port:5432│ │7474/7687│ │ 6379    │ │  API    │
└─────────┘ └─────────┘ └─────────┘ └─────────┘
 Documents    Graphs      Cache      AI Judge
```

📦 What Each Database Does

PostgreSQL (Primary Storage)

- **Purpose**: Store case documents, user data, arguments
- **Data**: Text content, file metadata, verdicts
- **Why**: ACID compliance, reliable persistence, easy queries

Neo4j (Relationship Graph)

- **Purpose**: Track relationships between cases, arguments, precedents
- **Data**: Case similarities, argument patterns, winning strategies
- **Why**: Fast relationship traversal, pattern discovery, analytics

Redis (Cache Layer)

- **Purpose**: Cache verdicts, API responses, session data
- **Data**: Temporary cached results
- **Why**: Reduce AI API calls, faster response times, cost savings

---

🛠️ Setup Options

Option 1: Docker Compose (Recommended) 🐳

Fastest way to get everything running:

```
# 1. Copy environment file
cp backend/.env.example backend/.env

# 2. Add your Groq API key to backend/.env
# GROQ_API_KEY=your_actual_key_here

# 3. Start all services
docker-compose up -d

# 4. Wait for services to be healthy (30-60 seconds)
docker-compose ps

# 5. Access the services:
# - Frontend: http://localhost:3000
# - Backend API: http://localhost:8000
# - Neo4j Browser: http://localhost:7474 (neo4j/password123)
# - PostgreSQL: localhost:5432 (postgres/postgres123)
# - Redis: localhost:6379
```

Stop services:

```
docker-compose down
```

View logs:

```
docker-compose logs -f backend
```

Option 2: Local Development (Manual) 💻

If you want to run services individually:

1. Start Databases (Docker)

```
# PostgreSQL
docker run -d --name ai-judge-postgres \
  -e POSTGRES_DB=ai_judge \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=postgres123 \
  -p 5432:5432 \
  postgres:15-alpine

# Neo4j
```

```
docker run -d --name ai-judge-neo4j \
  -e NEO4J_AUTH=neo4j/password123 \
  -p 7474:7474 -p 7687:7687 \
  neo4j:5.14-community

# Redis
docker run -d --name ai-judge-redis \
  -p 6379:6379 \
  redis:7-alpine
```

2. Setup Backend

```
cd backend

# Create virtual environment
python -m venv venv
.\venv\Scripts\Activate.ps1  # Windows
# source venv/bin/activate  # Mac/Linux

# Install dependencies
pip install -r requirements.txt

# Create .env file
cp .env.example .env
# Edit .env and add your GROQ_API_KEY

# Update .env with local database URLs:
# DATABASE_URL=postgresql://postgres:postgres123@localhost:5432/ai_judge
# NEO4J_URI=bolt://localhost:7687
# NEO4J_USER=neo4j
# NEO4J_PASSWORD=password123
# REDIS_URL=redis://localhost:6379

# Run backend (use hybrid version)
python app_hybrid.py
```

3. Setup Frontend

```
cd frontend

# Install dependencies
npm install

# Create .env.local
echo "NEXT_PUBLIC_API_URL=http://localhost:8000" > .env.local
```

```
# Run development server
npm run dev
```

📊 Database Connections

PostgreSQL

```
# Connect via psql
psql -h localhost -U postgres -d ai_judge
# Password: postgres123

# View tables
\dt

# Query cases
SELECT * FROM cases;
```

Neo4j

```
# Open browser
http://localhost:7474

# Login: neo4j / password123

# Sample queries:
MATCH (c:Case) RETURN c LIMIT 10;
MATCH (c:Case)-[r:SIMILAR_TO]-(similar:Case) RETURN c, r, similar;
MATCH (c:Case)-[:HAS_VERDICT]->(v:Verdict) RETURN v.winning_side, count(*) as
wins;
```

Redis

```
# Connect via redis-cli
docker exec -it ai-judge-redis redis-cli

# View all keys
KEYS *

# Get cached verdict
GET verdict:case123
```

🧪 Testing the System

1. Create a Case

```
curl -X POST http://localhost:8000/api/case/create \
  -H "Content-Type: application/json" \
  -d '{"case_id": "test-case-001"}'
```

2. Submit Arguments

```
# Side A
curl -X POST http://localhost:8000/api/case/test-case-001/argument \
  -H "Content-Type: application/json" \
  -d '{
    "case_id": "test-case-001",
    "side": "A",
    "text": "Plaintiff argues breach of contract..."
  }'

# Side B
curl -X POST http://localhost:8000/api/case/test-case-001/argument \
  -H "Content-Type: application/json" \
  -d '{
    "case_id": "test-case-001",
    "side": "B",
    "text": "Defendant argues force majeure..."
  }'
```

3. Generate Verdict

```
curl -X POST http://localhost:8000/api/case/test-case-001/verdict
```

4. Check Statistics

```
curl http://localhost:8000/api/statistics
```

🔍 API Endpoints

Endpoint	Method	Description
`/`	GET	API info
`/health`	GET	Health check (all databases)

Endpoint	Method	Description
`/api/case/create`	POST	Create new case
`/api/case/{id}/upload`	POST	Upload documents
`/api/case/{id}/argument`	POST	Submit argument
`/api/case/{id}/verdict`	POST	Generate verdict
`/api/case/{id}/followup`	POST	Submit follow-up (max 5)
`/api/case/{id}`	GET	Get case details
`/api/case/{id}/similar`	GET	Find similar cases (Neo4j)
`/api/statistics`	GET	System statistics

📈 Scaling Features

Caching Strategy

- Verdicts cached for 24 hours
- Case data cached for 1 hour
- Redis automatically evicts old entries
- Cache invalidation on updates

Database Indexing

- PostgreSQL: Indexed on case_id, created_at, status
- Neo4j: Indexed on case_id for fast lookups
- Optimized JOIN queries with proper foreign keys

Connection Pooling

- SQLAlchemy manages PostgreSQL connections
- Neo4j driver uses connection pooling
- Redis connection pooling built-in

🛠 Troubleshooting

"Cannot connect to PostgreSQL"

```
# Check if running
docker ps | grep postgres

# Restart
docker restart ai-judge-postgres
```

```
# Check logs
docker logs ai-judge-postgres
```

"Neo4j not available"

This is OK! Neo4j is optional. The system works without it, you just won't get:

- Case similarity matching
- Argument pattern analysis
- Relationship graphs

"Redis not available"

This is OK too! Redis is optional. Without it:

- No caching (slower responses)
- More API calls to Groq (higher cost)
- But everything still works

"Import errors in Python"

```
# Make sure you're in virtual environment
.\venv\Scripts\Activate.ps1

# Reinstall requirements
pip install -r requirements.txt --upgrade
```

🚀 Production Deployment

Database Options

Free Tier:

- PostgreSQL: Supabase (500MB free)
- Neo4j: Aura Free (200k nodes)
- Redis: Upstash (10k requests/day free)

Paid:

- PostgreSQL: AWS RDS, Google Cloud SQL
- Neo4j: Aura Professional
- Redis: Redis Enterprise Cloud

Environment Variables (Production)

```
# backend/.env
DATABASE_URL=postgresql://user:pass@host:5432/ai_judge
NEO4J_URI=neo4j+s://xxxxx.databases.neo4j.io
NEO4J_USER=neo4j
NEO4J_PASSWORD=<secure-password>
REDIS_URL=redis://default:password@host:12345
GROQ_API_KEY=<your-key>
```

📚 Additional Resources

- PostgreSQL Docs: https://www.postgresql.org/docs/
- Neo4j Docs: https://neo4j.com/docs/
- Redis Docs: https://redis.io/documentation
- FastAPI Docs: https://fastapi.tiangolo.com/
- SQLAlchemy Docs: https://docs.sqlalchemy.org/

🎯 Benefits of Hybrid Approach

Feature	Before (In-Memory)	After (Hybrid)
Persistence	❌ Lost on restart	✅ Permanent
Scalability	❌ Single server	✅ Horizontal scaling
Relationships	❌ Manual	✅ Graph queries
Performance	⚠️ Fast but limited	✅ Fast + cached
Analytics	❌ None	✅ Pattern discovery
Cost	✅ $0	⚠️ $0-50/month

You now have a production-ready, scalable AI Judge system! 🎉 ⚖️