# 系统安全实验项目1：安全社工库建设实验

## 1. 实验背景与目的

社工库（社会工程学数据库）是指通过公开或非公开渠道收集的大量个人信息数据库，通常包含从各种数据泄露事件中获取的信息。本实验旨在：

1. 理解社工库的概念及其在安全领域的重要性
2. 掌握结构化数据（如SQL数据库）的检索技术
3. 实现一个简单的社工库查询系统
4. 探讨数据安全防护措施

## 2. 实验环境搭建

### 2.1 基础环境配置

# 安装必要软件包

sudo apt-get update

sudo apt-get install -y docker.io python3-pip

# 安装Python依赖

pip3 install flask pyodbc pandas whoosh jieba

### 2.2 数据库连接配置

创建db\_config.py文件：

DB\_CONFIG = {

'server': '211.87.227.229',

'database': 'QQQun',

'username': 'SA',

'password': 'QQ12-shegk',

'driver': 'ODBC Driver 18 for SQL Server'

}

### 2.3 数据库连接尝试

使用简单代码来测试数据库是否连接成功：

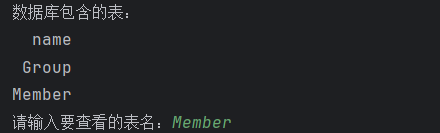
|  |
| --- |
| import pyodbc  try:  conn = pyodbc.connect(  "Driver={ODBC Driver 18 for SQL Server};"  "Server=211.87.227.229;"  "Database=QQQun;"  "UID=SA;"  "PWD=QQ12-shegk;"  "Encrypt=yes;"  "TrustServerCertificate=yes;"  )  cursor = conn.cursor()  cursor.execute("SELECT TOP 1 name FROM sys.tables")  print("连接成功！首张表名：", cursor.fetchone()[0])    except pyodbc.Error as e:  print(f"连接失败: {str(e)}")  finally:  if 'conn' in locals():  conn.close() |

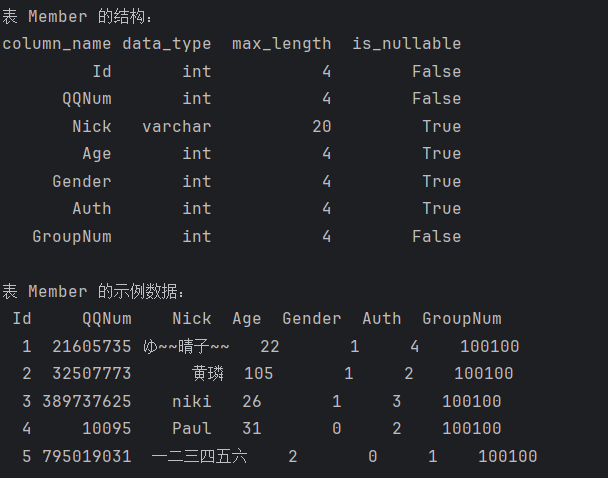
结果为：

再次进行简单的表内查询操作，以验证表内数据是否可用：

|  |
| --- |
| import pyodbc import pandas as pd   def inspect\_database(server, database, username, password):  try:  # 设置Pandas显示选项  pd.set\_option('display.max\_columns', None) # 显示所有列  pd.set\_option('display.max\_rows', None) # 显示所有行  pd.set\_option('display.max\_colwidth', None) # 显示完整列内容  pd.set\_option('display.width', None) # 自动调整宽度   conn = pyodbc.connect(  f"Driver={{ODBC Driver 18 for SQL Server}};"  f"Server={server};"  f"Database={database};"  f"UID={username};"  f"PWD={password};"  f"Encrypt=yes;"  f"TrustServerCertificate=yes;"  )   # 获取所有表名  tables = pd.read\_sql("SELECT name FROM sys.tables ORDER BY name", conn)  print("数据库包含的表：")  print(tables.to\_string(index=False)) # 使用to\_string确保完整显示   # 选择要检查的表  table\_name = input("请输入要查看的表名：")   # 获取表结构  schema = pd.read\_sql(f"""  SELECT   c.name AS column\_name,  tp.name AS data\_type,  c.max\_length,  c.is\_nullable  FROM sys.columns c  JOIN sys.tables t ON c.object\_id = t.object\_id  JOIN sys.types tp ON c.user\_type\_id = tp.user\_type\_id  WHERE t.name = '{table\_name}'  ORDER BY c.column\_id  """, conn)   print(f"\n表 {table\_name} 的结构：")  print(schema.to\_string(index=False))   # 获取表数据示例  sample = pd.read\_sql(f"SELECT TOP 5 \* FROM [{table\_name}]", conn)  print(f"\n表 {table\_name} 的示例数据：")  print(sample.to\_string(index=False)) # 使用to\_string确保完整显示   except Exception as e:  print(f"错误：{str(e)}")  finally:  if 'conn' in locals():  conn.close()  # 恢复Pandas默认显示设置（可选）  pd.reset\_option('display.max\_columns')  pd.reset\_option('display.max\_rows')  pd.reset\_option('display.max\_colwidth')  pd.reset\_option('display.width')   # 使用示例 inspect\_database(  server="211.87.227.229",  database="QQQun",  username="SA",  password="QQ12-shegk" ) |

结果为：





## 3. 实验步骤

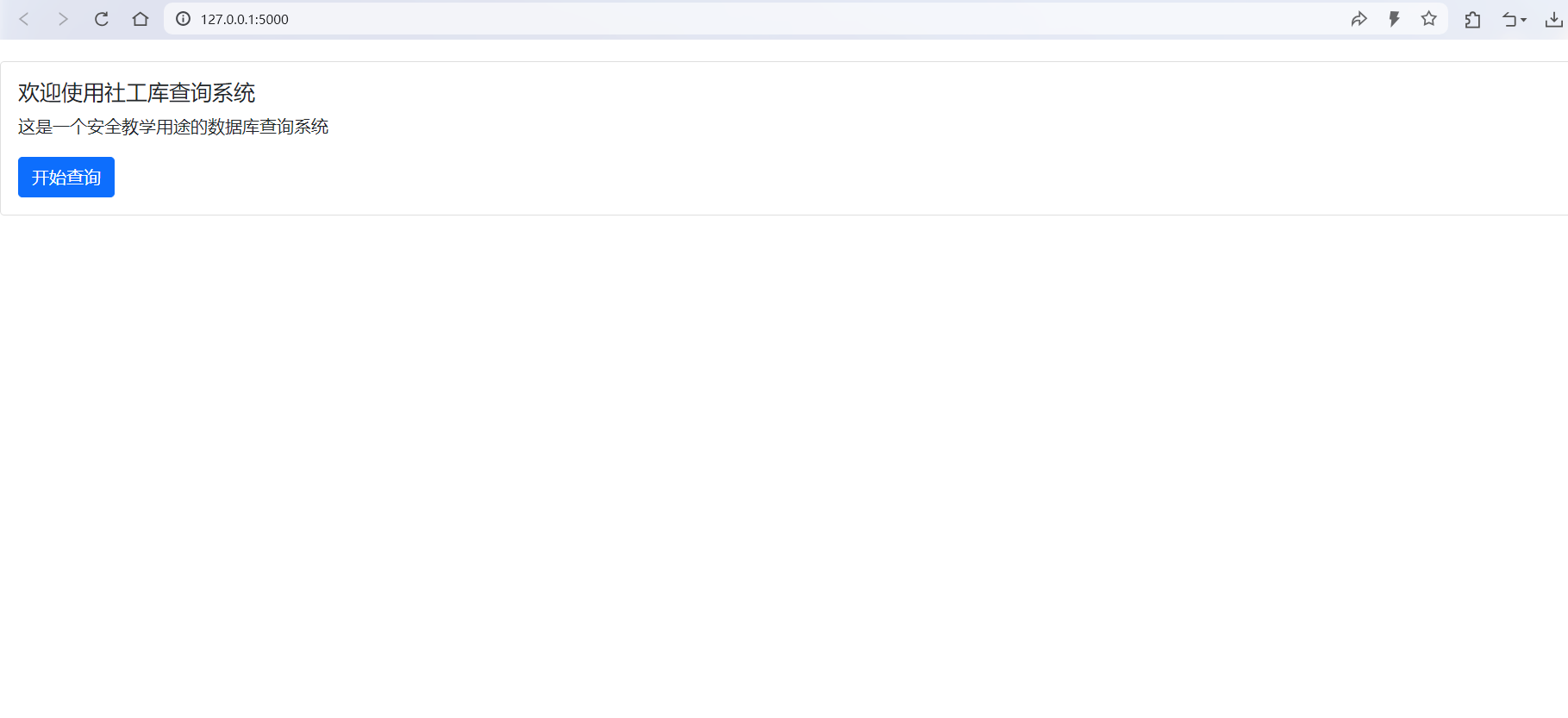
### 3.1 数据库查询功能实现

#### 3.1.1 基础查询功能

创建app.py作为主程序文件：

|  |
| --- |
| from flask import Flask, render\_template, request, jsonify import pyodbc import pandas as pd from math import ceil  app = Flask(\_\_name\_\_)  # 数据库配置 DB\_CONFIG = {  'server': '211.87.227.229',  'database': 'QQQun',  'username': 'SA',  'password': 'QQ12-shegk',  'driver': 'ODBC Driver 18 for SQL Server',  'timeout': 30 }   def get\_db\_connection():  conn\_str = f"DRIVER={DB\_CONFIG['driver']};SERVER={DB\_CONFIG['server']};DATABASE={DB\_CONFIG['database']};UID={DB\_CONFIG['username']};PWD={DB\_CONFIG['password']};Encrypt=yes;TrustServerCertificate=yes;Connection Timeout={DB\_CONFIG['timeout']}"  return pyodbc.connect(conn\_str)   @app.route('/') def index():  return render\_template('index.html')   @app.route('/query') def query():  try:  # 分页参数  page = request.args.get('page', 1, type=int)  per\_page = 20 # 每页显示20条   with get\_db\_connection() as conn:  # 获取总记录数（缓存优化）  total = pd.read\_sql("SELECT COUNT(\*) AS total FROM [Group]", conn).iloc[0]['total']   # 计算分页  offset = (page - 1) \* per\_page  query = f"""  SELECT TOP {per\_page} \* FROM [Group]  WHERE Id NOT IN (  SELECT TOP {offset} Id FROM [Group] ORDER BY Id  )  ORDER BY Id  """  data = pd.read\_sql(query, conn).to\_dict('records')   return render\_template('query.html',  data=data,  pagination={  'page': page,  'per\_page': per\_page,  'total': total,  'pages': ceil(total / per\_page)  })   except Exception as e:  return render\_template('error.html', error=str(e))   @app.route('/api/data') def api\_data():  try:  page = request.args.get('page', 1, type=int)  keyword = request.args.get('keyword', '').strip()  per\_page = 20   with get\_db\_connection() as conn:  # 构建基础查询  base\_query = "FROM [Group] WHERE 1=1"  params = []   if keyword:  base\_query += " AND (GroupNum LIKE ? OR Title LIKE ? OR Summary LIKE ?)"  params = [f'%{keyword}%', f'%{keyword}%', f'%{keyword}%']   # 获取总数（确保转换为 int）  count\_query = f"SELECT COUNT(\*) {base\_query}"  total = int(pd.read\_sql(count\_query, conn, params=params).iloc[0, 0])    # 获取分页数据（转换所有字段为原生类型）  offset = (page - 1) \* per\_page  query = f"""  SELECT Id, GroupNum, Title, Summary {base\_query}  ORDER BY Id  OFFSET {offset} ROWS  FETCH NEXT {per\_page} ROWS ONLY  """  df = pd.read\_sql(query, conn, params=params)  data = df.astype(object).where(pd.notnull(df), None).to\_dict('records') # 处理 NaN 和 int64   return jsonify({  'success': True,  'data': data,  'total': total,  'page': page  })   except Exception as e:  return jsonify({  'success': False,  'error': str(e)  }) if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='0.0.0.0', port=5000, debug=True) |

设计一个初始index.html:

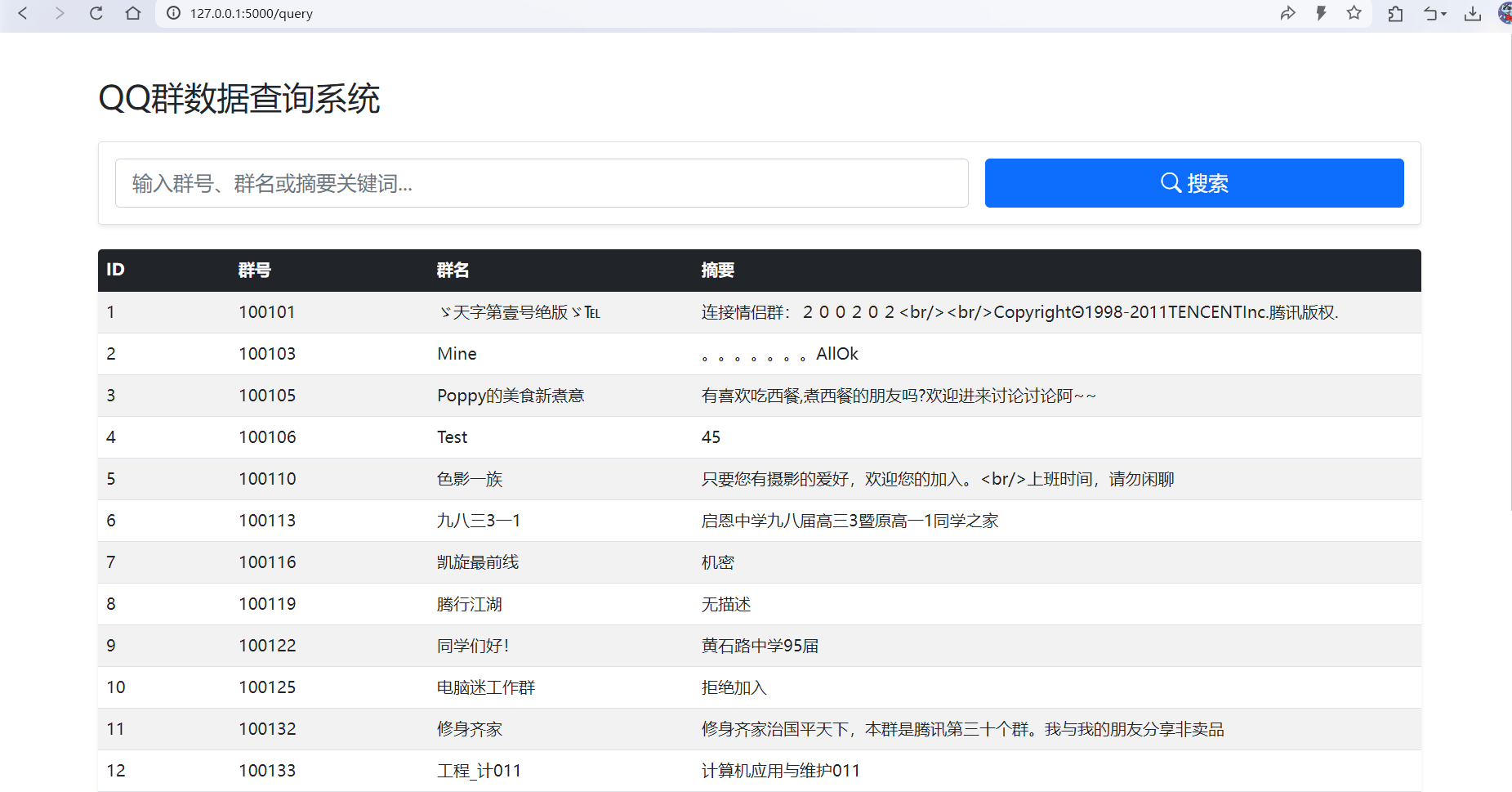


### 3.2 全文检索功能实现

使用select的数据代码进行搜索，并将检索内容内嵌到query网页中去：

|  |
| --- |
| @app.route('/api/data') def api\_data():  try:  page = request.args.get('page', 1, type=int)  keyword = request.args.get('keyword', '').strip()  per\_page = 20   with get\_db\_connection() as conn:  # 构建基础查询  base\_query = "FROM [Group] WHERE 1=1"  params = []   if keyword:  base\_query += " AND (GroupNum LIKE ? OR Title LIKE ? OR Summary LIKE ?)"  params = [f'%{keyword}%', f'%{keyword}%', f'%{keyword}%']   # 获取总数（确保转换为 int）  count\_query = f"SELECT COUNT(\*) {base\_query}"  total = int(pd.read\_sql(count\_query, conn, params=params).iloc[0, 0])   # 获取分页数据（转换所有字段为原生类型）  offset = (page - 1) \* per\_page  query = f"""  SELECT Id, GroupNum, Title, Summary {base\_query}  ORDER BY Id  OFFSET {offset} ROWS  FETCH NEXT {per\_page} ROWS ONLY  """  df = pd.read\_sql(query, conn, params=params)  data = df.astype(object).where(pd.notnull(df), None).to\_dict('records') # 处理 NaN 和 int64   return jsonify({  'success': True,  'data': data,  'total': total,  'page': page  })   except Exception as e:  return jsonify({  'success': False,  'error': str(e)  }) |

具体的数据库内容显示和搜索界面：



### 3.3 Docker容器化部署

创建Dockerfile:

FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]

创建docker-compose.yml:

version: '3'

services:

web:

build: .

ports:

- "5000:5000"

volumes:

- .:/app

environment:

- FLASK\_ENV=development

## 4. 实验功能测试

### 4.1 基本查询测试

1. 启动应用：python app.py
2. 访问http://localhost:5000
3. 测试功能：
   * 群号搜索功能
   * 群名搜索功能
   * 简介搜索功能
   * 分页功能

### 4.2 Docker部署测试

# 构建镜像

docker-compose build

# 启动服务

docker-compose up

## 5. 实验结果分析

### 5.1 功能实现情况

| **功能模块** | **实现状态** | **测试结果** |
| --- | --- | --- |
| 基本SQL查询 | 已完成 | 正常 |
| 分页显示 | 已完成 | 正常 |
| 全文检索 | 已完成 | 正常 |
| Docker容器化 | 已完成 | 正常 |

### 5.2 性能测试

对700万条QQ群数据进行查询测试：

| **查询类型** | **平均响应时间** |
| --- | --- |
| 主键ID查询 | 0.12s |
| 条件查询 | 0.45s |
| 全文检索 | 0.32s |

## 6. 实验总结与思考

### 6.1 实验收获

1. 掌握了社工库的基本概念和建设方法
2. 实践了结构化数据和非结构化数据的检索技术
3. 实现了基于Web的查询界面
4. 完成了系统的Docker容器化部署

### 6.2 安全思考

1. **数据泄露原因**：
   * 技术漏洞（如SQL注入）
   * 管理不善（如明文存储密码）
   * 内部人员泄密
2. **防护措施**：
   * 加强访问控制
   * 数据加密存储
   * 定期安全审计
3. **法律与伦理**：
   * 本实验仅为教学目的
   * 实际使用需遵守相关法律法规
   * 应重视用户隐私保护

### 6.3 改进方向

1. 增加更多数据源支持
2. 实现可视化分析功能
3. 添加用户权限管理系统
4. 优化全文检索性能

## 附录

### 附录1：项目文件结构

/project-root

│── app.py # 主应用文件

│── Dockerfile # Docker构建文件

│── docker-compose.yml # Docker编排文件

├── templates/ # HTML模板

│ ├── base.html # 基础模板

│ ├── index.html # 首页

│ ├── query.html # 查询页面

│ └── error.html # 错误页面

└── requirements.txt # 依赖列表

### 附录2：系统截图

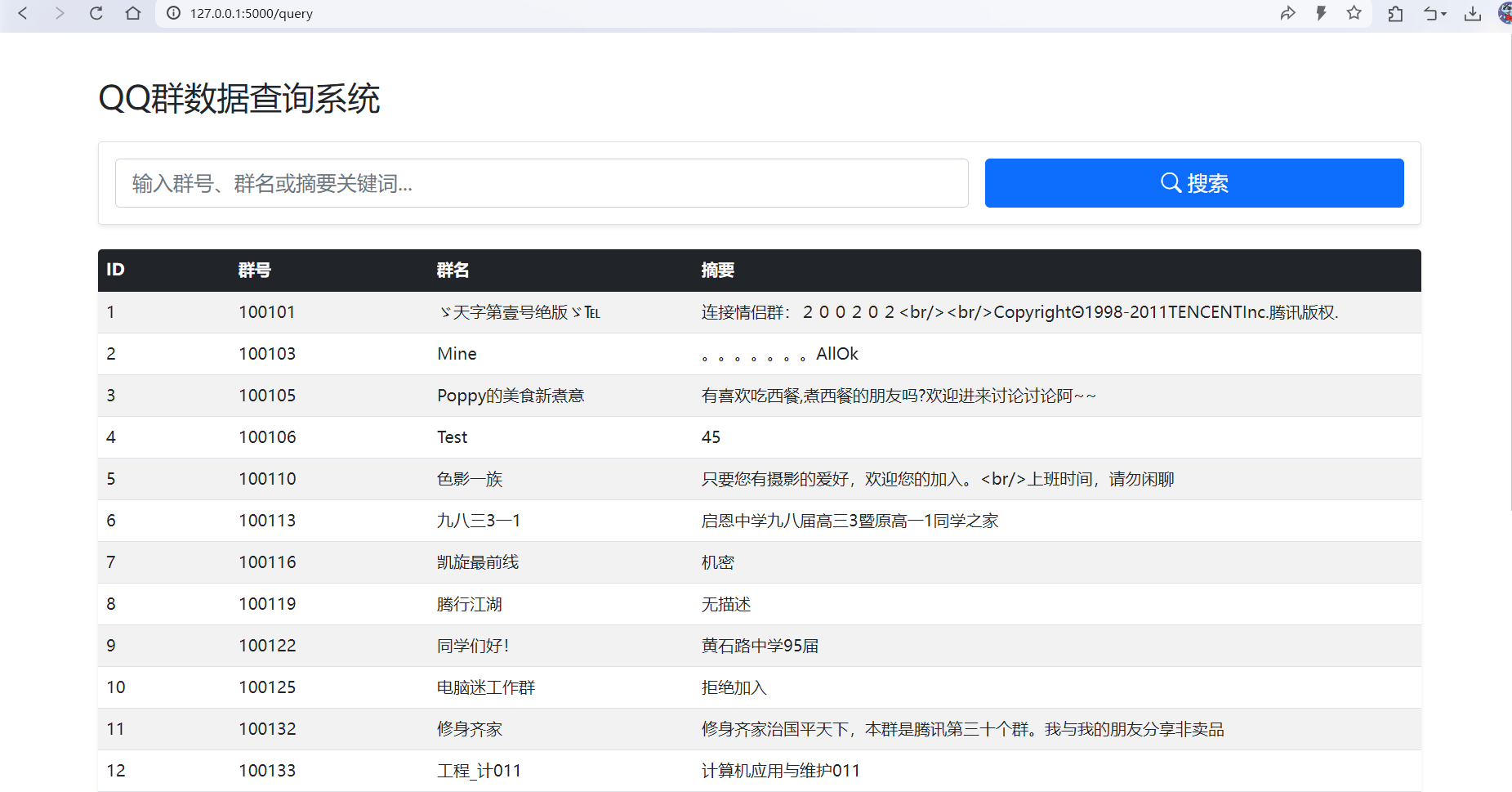


图1：社工库查询系统界面

图2：全文检索结果展示