穿透式检索

本文档描述了对下一版本的数据检索系统中穿透式检索的功能特性的设计

什么是穿透式检索

当前的检索

- 通过关键字搜索相关结果，并根据关联度进行排名，之后根据得到的结果中进行整理，获得最终结果
- 最终结果并不能保证是需要的结果

穿透式检索

- 直接检索需要的数据，不需要人脑再次整理，可以参考Google OneBox的概念
- 穿透式检索因为其结果的精确，可以为大数据和人工智能提供准确的素材，进而帮助大数据和人工智能获得更加准确的结果

为什么区块链需要穿透式检索

- 区块链数据是严谨的业务数据，对业务数据的分析有利于业务的增强。
- 当前简单地关键词搜索无法提供所需的业务信息
- 在具体的业务中，快速方便地检索出需要的数据，为业务分析提供支持
- 为人工智能等系统提供基础数据

实例解析

样例数据

样例数据采用了电影数据。电影本身有以下基础信息：

```
id：标识号
title：电影名称
popularity：在 Movie Database 上的相对页面查看次数
cast：演员列表
runtime：电影时长
production_companies：制作公司列表
release_date：首次上映日期
crew：剧务人员列表
```

提交数据元信息，创建索引

由于系统本身无法确定哪些数据需要被索引，而全部索引不仅浪费资源，还会影响重要数据，因此需要用户根据具体的业务需要对需要索引的数据进行指定，即指定索引的元信息。

元信息定义如下：

```
type Crew{
    id(isIndex: Boolean = true, isPrimaryKey: Boolean = true):                  Int
    name(termIndex: Boolean = true):               String
    gender(isIndex: Boolean = true):               Int
    credit_id(termIndex: Boolean = true):          String
    job(termIndex: Boolean = true):                String
    department(termIndex: Boolean = true):         String
}

type Cast{
    id(isIndex: Boolean = true, isPrimaryKey: Boolean = true):                   Int
    cast_id(isIndex: Boolean = true):               Int
    character(termIndex: Boolean = true):           String
    credit_id(termIndex: Boolean = true):           String
    gender(isIndex: Boolean = true):                Int
    name(termIndex: Boolean = true):                String
    order(isIndex: Boolean = true):                 Int
}

type Company{
    id(isIndex: Boolean = true, isPrimaryKey: Boolean = true):                  Int
    name(termIndex: Boolean = true):               String
}

type Movie{
    id(isIndex: Boolean = true, isPrimaryKey: Boolean = true):                   Int
    popularity(isIndex: Boolean = true):            Float
    release_date(isIndex: Boolean = true):          DateTime
    runtime(isIndex: Boolean = true):               Int
    title(termIndex: Boolean = true):               String
    companies:                                      [Int]
    crew:                                           [Int]
    casts:                                           [Int]
}

type EdgeMovieCompany{
    Movie(companies: [Int]): EdgeFrom
    Company(id: Int): EdgeTo
}

type EdgeMovieCrew{
    Movie(crew: [Int]): EdgeFrom
    Crew(id: Int): EdgeTo
}

type EdgeMovieCast{
    Movie(casts: [Int]): EdgeFrom
    Cast(id: Int): EdgeTo
}
```

系统接收到这些元信息后，会对区块链中现有的和之后新产生的该类数据的这些属性创建索引。

其中，

```
isIndex: Boolean = true

termIndex: Boolean = true
```

表示为该字段创建索引（具体索引方式此处暂时不做详述）

数据检索

包含关键字 Avatar 的电影记录

模糊查询的典型实例，指获取所有包含某个关键字列表的查询

可以使用类似如下格式的查询语句：

```
{
    movies(func: anyofterms(movie-title, "Avatar")){
      uid
     expand(_all_)
    }
}
```

查询结果如下：

```
{
    "data": {
        "movies": [
            {
                "uid": "0xb3f908",
                "movie-id": 19995,
                "movie-title": "Avatar",
                "movie-popularity": 150.437577,
                "movie-runtime": 162
            }
        ]
    }
}
```

查询由 James Cameron 导演的电影列表

列表查询的典型实例，指对符合逻辑条件的记录进行过滤的查询。

可以使用类似如下格式的查询语句：

```
{
    movies(func: allofterms(crew-name, "James Cameron")){
        crew-movie{
            uid
            expand(_all_)
        }
    }
}
```

返回如下格式的查询结果：


```
{
    "data": {
        "movies": [
            {
                "crew-movie": [
                    {
                        "uid": "0xb3f77b",
                        "movie-runtime": 141,
                        "movie-id": 36955,
                        "movie-title": "True Lies",
                        "movie-popularity": 38.729418
                    },
                    {
                        "uid": "0xb3f783",
                        "movie-runtime": 194,
                        "movie-id": 597,
                        "movie-title": "Titanic",
                        "movie-popularity": 100.025899
                    },
                    {
                        "uid": "0xb3f825",
                        "movie-runtime": 137,
                        "movie-id": 280,
                        "movie-title": "Terminator 2: Judgment Day",
                        "movie-popularity": 101.74155
                    }
                ]
            }
        ]
    }
}
```


查询由 James Cameron 导演的，popularity 超过 100 的电影列表

条件复杂一些的查询，在一个条件查询结果的基础上，根据下一个条件进行进一步查询的查询

可以使用类似如下格式的查询语句：

```
{
    movies(func: allofterms(crew-name, "James Cameron")){
        crew-movie @filter(gt(movie-popularity, 100)){
            uid
            expand(_all_)
        }
    }
}
```

返回如下格式的查询结果：


```
{
    "data": {
        "movies": [
            {
                "crew-movie": [
                    {
                        "uid": "0xb3f783",
                        "movie-runtime": 194,
                        "movie-id": 597,
                        "movie-title": "Titanic",
                        "movie-popularity": 100.025899
                    },
                    {
                        "uid": "0xb3f825",
                        "movie-runtime": 137,
                        "movie-id": 280,
                        "movie-title": "Terminator 2: Judgment Day",
                        "movie-popularity": 101.74155
                    },
                    {
                        "uid": "0xb3f908",
                        "movie-runtime": 162,
                        "movie-id": 19995,
                        "movie-title": "Avatar",
                        "movie-popularity": 150.437577
                    }
                ]
            }
        ]
    }
}
```


与 James Cameron 合作过的演员还参与过哪些电影的拍摄

说明

这个实例体现了一种更复杂的查询，主要有两个方面的作用：
1. 根据确定的关系直接检索结果
2. 它可以在间接相关的数据之间建立直接相关的联系，相关的关系结合在一起形成一张直观的关系图，展示出普通条件下难以发现的内容

思路分析

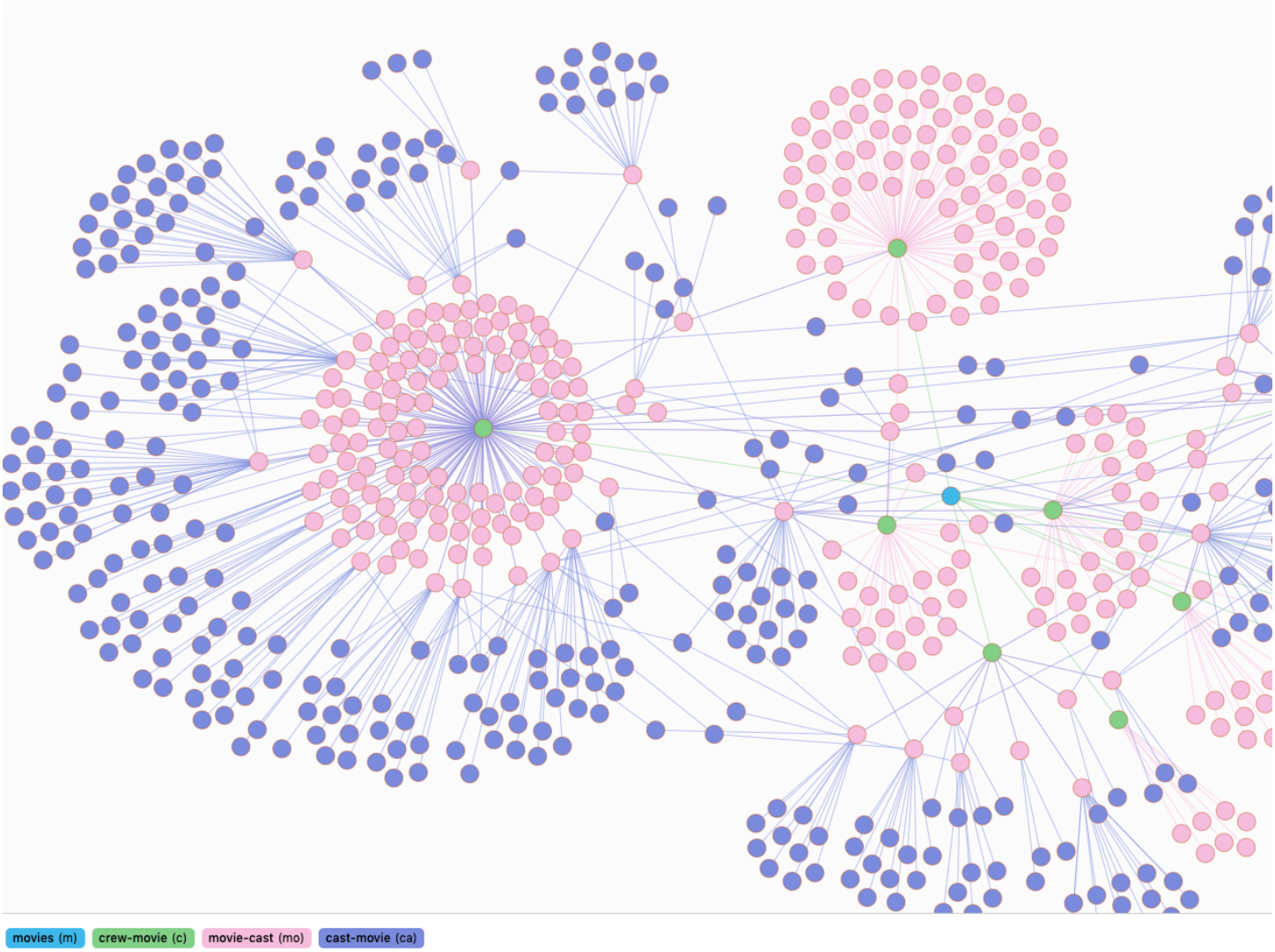
一般来说按照以下思路就可以解决这个问题：

1. Cameron 导演的电影有哪些
2. 电影涉及的演员有哪些
3. 这些演员参与的电影

按照思路，写出查询语句

```
{
    movies(func: allofterms(crew-name, "James Cameron")){
        crew-movie {
            movie-title
            movie-cast{
                cast-name
                cast-movie{
                    movie-title
                }
            }
        }
    }
}
```

查询结果如图：



结果解析

- 图中绿色的节点代表电影，与绿色电影节点连接的粉色节点代表的是演员，与粉色演员节点连接的蓝色节点代表演员出演的其它电影
- 粉色演员节点连接的蓝点越多，说明这个演员参演的电影越多，这意味着这个演员有名的概率较大
- 可以继续加入其它过滤条件得到更精确的信息

系统组件

功能模块解析

穿透式检索系统可以划分为以下模块：

- 业务逻辑管理模块
- 数据索引管理模块
- 查询管理模块

业务逻辑管理模块

主要包括以下功能：

- 数据中包含的业务逻辑需要由业务相关人员进行定义，包括业务逻辑的增加和删除。
- 业务逻辑通过特定的格式进行定义，定义中不仅包含了业务逻辑本身，同时包含了数据在区块链账本中的位置和范围
- 指定应用业务逻辑进行数据索引。

数据索引管理模块

主要包括以下功能：

- 根据业务逻辑定义进行数据的索引，为之后的查询提供支持
- 当业务逻辑定义应用时，能够根据变化进行相应地处理

查询管理模块

查询管理模块针对已经索引的数据进行查询。

查询的具体方式可以采用具体的查询语句，除此之外，对于一些常用的业务查询，可以进行一定封装以方便使用

第一阶段开发（仅供参考）

第一阶段的目标是在当前实例的基础上，在测试环境下，完成基本核心功能的开发

根据之前功能模块的分析，第一阶段需要开发以下工作和系统（或组件）

基础环境

搭建一个基础的区块链环境，环境中包含了可用的样例数据，为之后的系统（或组件）的开发提供测试和验证环境

业务逻辑管理系统

客户端

业务逻辑定义的编写界面的开发

可以通过服务端接口保存业务逻辑，并能从服务端拉取已经保存的定义列表

服务端

提供保存和拉取业务逻辑定义的接口

数据索引管理系统

根据业务逻辑的应用命令，进行数据索引的变更

查询管理系统（或组件）

客户端

- 编写查询语句，以及执行查询的界面
- 查询的执行通过将查询语句发送到服务端完成，根据服务端返回的查询进行相应处理
- 常用查询的管理（可以保存在本地）

服务端

提供查询请求的接口