Transwarp StellarDB Version 5.0.1 使用手册

星环信息科技（上海）股份有限公司

2023-08-17

目录

1. StellarDB 5.0.1 变更说明 . . . 2
1.1. 设计变更 . . . 2
1.2. 新增功能 . . . 2
1.3. 功能优化 . . . 2
1.4. 问题修复 . . . 2
2. StellarDB 5.0.0 版本变更说明 . . . 3
2.1. 设计变更 . . . 3
2.2. 语法变更 . . . 7
2.3. 新增内容 . . . 7
2.4. 优化内容 . . . 8
2.5. Deprecated（功能废除清单） . . . 8
2.6. 已知问题 . . . 9
3. 简介 . . . 10
3.1. 产品背景 . . . 10
3.2. 产品架构 . . . 11
3.3. 产品组成 . . . 11
3.3.1. 存储引擎 . . . 11
3.3.2. 计算引擎 . . . 11
3.3.3. 可视化引擎 . . . 12
4. 快速上手 . . . 13
4.1. StellarDB初探 . . . 13
4.1.1. 使用KG Explorer构建图 . . . 13
4.1.2. 使用beeline构建图 . . . 18
4.2. StellarDB进阶 . . . 21
4.2.1. 使用KG Explorer交互 . . . 21
4.2.2. 使用beeline交互 . . . 23
4.2.3. 使用DBAService查看任务信息 . . . 26
5. Transwarp Extended openCypher . . . 29
5.1. TEoC 基础 . . . 29
5.1.1. 图数据简介 . . . 29
5.1.2. 存储模式 . . . 29
5.1.3. 数据模式 . . . 30
5.2. TEoC 前置参数 . . . 32
5.3. 创建图 . . . 33
5.3.1. 使用TEoC创建图 . . . 33
5.3.2. 使用JSON创建图 . . . 34
5.3.3. 使用已有图的schema创建新图 . . . 36
5.3.4. 查看已创建的图 . . . 36
5.3.5. 查看图schema信息 . . . 37
5.3.6. 选择/切换图 . . . 37
5.4. 删除图 . . . 38
5.5. 批量数据导入 . . . 39

5.5.1. 批量数据导数约束（新增） . . . 39
5.5.2. 准备数据 . . . 39
5.5.3. 数据导入 . . . 39
5.5.3.1. 使用bulk upsert/create/delete导入数据 . . . 39
5.5.3.2. 使用load语句进行批量数据导入（deprecated） . . . 41
5.5.3.3. 数据导入方法对比 . . . 41
5.5.4. 数据类型对应关系 . . . 41
5.5.5. 从分区表导入数据的注意事项 . . . 42
5.5.6. 数据校验 . . . 42
5.5.6.1. 错误数据输出 . . . 43
5.5.6.2. HDFS错误数据目录结构 . . . 43
5.5.7. 应用实例 . . . 43
5.5.8. 常见错误 . . . 45
5.5.8.1. 客户端执行异常 . . . 45
5.5.8.2. 服务端日志异常 . . . 46
5.6. 数据类型 . . . 48
5.6.1. 类型及表达式 . . . 48
5.6.2. 地理坐标类型 . . . 48
5.6.2.1. 类型简介 . . . 48
5.6.2.2. 使用方法 . . . 48
5.6.2.3. 查询方法 . . . 49
5.6.2.4. 空间索引 . . . 50
5.6.2.5. 可比性和可排序性 . . . 50
5.6.3. 时间序列类型 . . . 51
5.6.3.1. 时序类型特征 . . . 51
5.6.3.2. 使用方法 . . . 51
5.6.3.3. 更新和删除方法 . . . 52
5.6.3.4. 查询方法 . . . 52
5.7. 数据操作语句 . . . 54
5.7.1. 数据创建 . . . 54
5.7.1.1. 创建点 . . . 54
5.7.1.2. 创建边 . . . 54
5.7.1.3. 使用SQL表中的列创建点和边 . . . 55
5.7.2. 更新属性值 . . . 56
5.7.3. 删除点或边 . . . 56
5.7.4. 批量清除数据 . . . 57
5.7.4.1. 清除label数据（新） . . . 57
5.7.4.2. 清除全图数据 . . . 57
5.7.4.3. 清理悬挂边（新） . . . 58
5.8. 数据查询语句 . . . 59
5.8.1. 查找点 . . . 59
5.8.1.1. 按照属性和类型查找 . . . 59
5.8.1.2. 按照复杂条件查找 . . . 60
5.8.1.3. 按照UID查找 . . . 60
5.8.1.4. 查找精确点数量（较慢） . . . 61

5.8.1.5. 查找近似点数量（较快） . . . 61
5.8.2. 查询边 . . . 61
5.8.2.1. 按照属性和类型查找 . . . 61
5.8.2.2. （新）查找精确边数量（较慢） . . . 61
5.8.2.3. 查找近似边数量（较快） . . . 61
5.8.2.4. 单层路径查找 . . . 62
5.8.2.5. 多层路径查找 . . . 62
5.8.2.6. 多层路径查找（中间点边过滤） . . . 63
5.8.2.7. （新）多层路径查找（无上界） . . . 63
5.9. 高级查询 . . . 65
5.9.1. 多match查询 . . . 65
5.9.2. OPTIONAL MATCH . . . 65
5.9.3. K跳邻居（新） . . . 65
5.9.4. UNION和UNION ALL . . . 65
5.9.5. 返回所有非匿名内容 . . . 66
5.9.6. 返回null . . . 66
5.9.7. 结果列命名 . . . 66
5.9.8. 跳过前N条结果 . . . 66
5.9.9. 结果排序 . . . 67
5.9.10. 模糊查询 . . . 67
5.9.11. DISTINCT去重 . . . 67
5.9.12. 聚合函数 . . . 67
5.9.13. CASE WHEN语句 . . . 68
5.9.14. WITH的用法 . . . 68
5.9.15. 唯一性语义 . . . 69
5.9.16. 简单跨图查询 . . . 69
5.9.17. （新）查询最短路径 . . . 69
5.9.18. 查询结果导出到Quark SQL表 . . . 71
5.9.19. 查询结果导出到StellarDB其他图 . . . 72
5.9.20. Session级WHILE循环查询 . . . 73
5.9.21. 查询级别参数配置 . . . 74
5.10. 更新Schema . . . 75
5.10.1. 更新点属性（新） . . . 75
5.10.2. 更新点label . . . 75
5.10.3. 更新边属性（新） . . . 76
5.10.4. 更新边label . . . 77
5.10.5. 更新comment注释信息 . . . 77
5.11. 表达式 . . . 79
5.11.1. 类型表达式 . . . 79
5.11.2. 指代型表达式 . . . 79
5.11.3. 混合型表达式 . . . 79
5.11.4. 命名规则 . . . 80
5.11.5. 保留关键字 . . . 80
5.11.6. 符号 . . . 80
5.11.7. 转义字符 . . . 81

5.12. Explain语句结构 . . . 83
5.13. 变量声明 . . . 84
5.13.1. 声明简介 . . . 84
5.13.2. 变量声明 . . . 84
5.13.3. 变量赋值 . . . 84
5.13.4. 变量取值 . . . 85
5.13.5. 删除变量 . . . 85
5.13.6. 显示所有变量 . . . 86
5.14. 重命名图 . . . 87
5.15. 图拷贝 . . . 88
5.15.1. 使用COPY进行图拷贝 . . . 88
5.15.2. 使用BULK COPY进行图拷贝（新） . . . 88
5.15.3. 图拷贝语法扩展 . . . 88
5.16. 函数 . . . 89
5.16.1. 基础函数 . . . 89
5.16.2. 聚合函数 . . . 92
5.16.3. 数学函数 . . . 92
5.16.4. 字符串操作函数 . . . 93
5.16.5. 集合操作函数 . . . 94
5.16.6. crux-cypher内部函数 . . . 94
5.16.7. Reduce函数 . . . 94
5.16.8. 空间函数 . . . 95
5.16.8.1. point()函数 . . . 95
5.16.8.2. pt_distance()函数 . . . 95
5.16.8.3. withinBox()函数 . . . 96
5.16.8.4. withinCircle()函数 . . . 97
5.17. 断言函数 . . . 98
5.17.1. all()函数 . . . 98
5.17.2. any()函数 . . . 98
5.17.3. single()函数 . . . 98
5.17.4. none()函数 . . . 98
5.18. 索引（新） . . . 100
5.18.1. 新增索引 . . . 100
5.18.2. 删除索引 . . . 100
5.18.3. 查看索引 . . . 101
5.18.4. 索引重建 . . . 101
5.18.4.1. 提交索引重建任务 . . . 101
5.18.4.2. 查看索引重建任务状态 . . . 102
5.18.4.3. StellarDB相关配置项 . . . 103
5.19. 全文索引 . . . 105
5.19.1. 查看分词器 . . . 105
5.19.2. 创建全文索引 . . . 105
5.19.3. 指定图名查看全文索引 . . . 107
5.19.4. 查询全文索引 . . . 107
5.19.5. 删除全文索引 . . . 108

5.20. 查看系统内置函数（新） . . . 109
5.21. StellarDB 5.0.0 动态图(Dynamic Graph)模型（新） . . . 110
5.21.1. 为什么引入动态图模型？ . . . 110
5.21.2. 动态图模型的动态变化 . . . 110
5.21.3. 创建动态图模型 . . . 110
5.21.4. 动态图模型的查询 . . . 110
5.21.5. 动态图的其他操作 . . . 110
6. StellarDB图算法 . . . 111
6.1. 图计算 . . . 111
6.1.1. 图计算简介 . . . 111
6.1.2. 图数据视图 . . . 111
6.1.3. 在子图中执行图算法 . . . 112
6.1.4. 基本度量(Network Measurements) . . . 112
6.1.4.1. 图直径(Diameter) . . . 113
6.1.4.2. 图半径(Radius) . . . 113
6.1.4.3. 度(Degree) . . . 113
6.1.4.4. K-Core . . . 113
6.1.4.5. 三角计数(Triangle Counting) . . . 114
6.1.4.6. 局部聚集系数(Local Clustering Coefficient) . . . 114
6.1.4.7. 平均聚集系数(Average Clustering Coefficient) . . . 114
6.1.5. 中心性算法(Centrality) . . . 115
6.1.5.1. PageRank . . . 115
6.1.5.2. Personalized PageRank . . . 115
6.1.5.3. ArticleRank . . . 116
6.1.5.4. Weighted PageRank . . . 116
6.1.5.5. 特征向量中心性(Eigenvector Centrality) . . . 117
6.1.5.6. 介度中心性(Betweenness Centrality) . . . 118
6.1.5.7. 接近中心性(Closeness Centrality) . . . 118
6.1.5.8. 调和中心性(Harmonic Centrality) . . . 119
6.1.5.9. 影响力最大化(Greedy Influence Maximization) . . . 119
6.1.5.10. CELF Influence Maximization . . . 120
6.1.5.11. TrustRank . . . 120
6.1.5.12. Hyperlink-Induced Topic Search(HITS) . . . 121
6.1.6. 路径搜寻(Pathfinding) . . . 122
6.1.6.1. 深度优先搜索(DFS) . . . 122
6.1.6.2. 广度优先搜索(BFS) . . . 122
6.1.6.3. 最短路径(Shortest Path) . . . 123
6.1.6.4. Dijkstra Shortest Path . . . 124
6.1.6.5. SPFA Shortest Path . . . 124
6.1.6.6. SPFA Single Source Shortest Path . . . 125
6.1.6.7. 随机游走(RandomWalk) . . . 125
6.1.7. 社区发现(Community Detection) . . . 126
6.1.7.1. 连通子图(Connected Components) . . . 126
6.1.7.2. 强连通子图(Strongly Connected Components) . . . 126
6.1.7.3. 标签传播(LPA) . . . 127

6.1.7.4. Speaker-Listener标签传播(SLPA) . . . 128
6.1.7.5. Fast Unfolding(Louvain) . . . 128
6.1.8. 相似度(Similarity) . . . 129
6.1.8.1. Jaccard相似度 . . . 129
6.1.8.2. Pearson相似度 . . . 129
6.1.8.3. Cosine相似度 . . . 130
6.1.8.4. Overlap相似度 . . . 131
6.1.8.5. 欧式距离相似度(Euclidean Distance) . . . 131
6.1.8.6. 节点相似度 . . . 132
6.1.9. 链路预测(Link Prediction) . . . 133
6.1.9.1. 共同邻居(Common Neighbors) . . . 133
6.1.9.2. 总邻居数(Total Neighbors) . . . 133
6.1.9.3. Adamic Adar . . . 134
6.1.9.4. Resource Allocation . . . 134
6.1.9.5. Preferential Attachment . . . 135
6.2. StellarDB3.0算法兼容性说明 . . . 135
6.2.1. PageRank . . . 136
6.2.2. GuaranteeCircle . . . 136
6.2.3. All Pair Shortest Path . . . 136
6.2.4. Triangle Counting . . . 137
6.2.5. Global Cluster Coefficient (deprecated) . . . 137
6.2.6. Eccentricity (deprecated) . . . 138
6.3. 图算法相关扩展： . . . 139
6.3.1. 图算法结果写回图 (3.0版本) . . . 139
6.3.2. 图算法结果写回图 (4.0版本及以上) . . . 139
6.3.3. 图算法结果导出至关系型表 . . . 139
7. StellarDB自定义函数、存储过程和图算法 . . . 140
7.1. 自定义函数 . . . 140
7.1.1. 自定义函数实现 . . . 140
7.1.1.1. 继承UDF基类 . . . 140
7.1.1.2. 继承GenericUDF基类 . . . 141
7.1.2. 自定义函数加载 . . . 142
7.2. 自定义存储过程 . . . 143
7.2.1. 自定义存储过程实现 . . . 143
7.2.2. 自定义存储过程加载 . . . 145
7.3. 自定义图算法 . . . 145
7.3.1. 自定义图算法实现 . . . 145
7.3.2. 自定义图算法加载 . . . 149
8. StellarDB运行与维护 . . . 150
8.1. 回收站功能 . . . 150
8.1.1. 禁用/启用回收站功能 . . . 150
8.1.2. 查看回收站内容 . . . 150
8.1.3. 从回收站彻底删除图 . . . 151
8.1.4. 从回收站恢复图 . . . 151
8.2. 图备份与图恢复 . . . 153

8.2.1. 进行图全量备份 . . . 153
8.2.2. 进行图增量备份 . . . 153
8.2.3. 进行图恢复 . . . 154
8.2.4. 查看图备份信息 . . . 154
8.2.5. 删除全部备份数据 . . . 155
8.2.6. 删除指定备份数据 . . . 155
8.2.7. 删除失败的恢复目录 . . . 155
8.3. Shard数据副本迁移 . . . 156
8.3.1. 执行副本迁移 . . . 156
8.4. 故障shard副本修复 . . . 157
8.4.1. shard副本数损失少于一半的情况 . . . 157
8.4.1.1. 创建新副本 . . . 157
8.4.1.2. 删除副本/删除副本元信息 . . . 158
8.4.2. shard副本数损失大于一半的情况 . . . 159
8.4.2.1. 从shard正常节点复制到目标节点 . . . 159
8.4.2.2. 查询复制任务状态 . . . 159
8.4.2.3. 手动删除复制任务状态缓存 . . . 159
8.5. 硬盘写满异常处理方式 . . . 160
8.5.1. 解除shard的IOE-Paused状态 . . . 160
8.6. 图存储配额限制 . . . 162
8.6.1. StellarDB native存储配额设置 . . . 162
8.6.2. Guardian存储配额设置 . . . 162
8.6.3. 查询指定图的存储占用量 . . . 162
9. StellarDB数据安全 . . . 164
9.1. 图数据静态加密 . . . 164
9.1.1. StellarDB数据文件编码 . . . 164
9.1.2. 对称加密功能 . . . 164
9.2. 数据脱敏 . . . 165
9.2.1. 规则表结构 . . . 165
9.2.2. 配置掩码流程 . . . 165
9.2.3. 参考案例 . . . 165
10. KG Explorer使用文档（新） . . . 170
10.1. KG Explorer简介 . . . 170
10.2. KG Explorer功能介绍 . . . 170
10.3. KG Explorer常用参数说明 . . . 171
10.4. 访问KG Explorer . . . 173
10.5. KG Explorer用户认证与安全 . . . 174
10.6. KG Explorer主要功能介绍 . . . 175
10.6.1. 可视化图谱创建 . . . 175
10.6.2. CSV数据导入 . . . 177
10.6.3. Hive表数据导入 . . . 180
10.6.4. 2D查询分析展示 . . . 185
10.6.4.1. 查询 . . . 185
10.6.4.2. 查询结果 . . . 185
10.6.4.3. 点边信息展示 . . . 187

10.6.4.4. 切换布局 . . . 188
10.6.4.5. 自定义点边大小、颜色 . . . 188
10.6.4.6. 节点扩展 . . . 189
10.6.4.7. schema预览 . . . 190
10.6.4.8. 样式配置 . . . 191
10.6.4.9. 结果筛选 . . . 191
10.6.4.10. 最短路径和全路径展示 . . . 192
10.6.4.11. 画布工具栏 . . . 195
10.6.4.12. 编辑点边 . . . 196
10.6.4.13. 执行图算法 . . . 196
10.6.4.14. 查看查询语句记录 . . . 198
10.6.4.15. 语句收藏 . . . 198
10.6.5. 动态图时间轴可视化 . . . 198
10.6.5.1. 配置时间轴 . . . 199
10.6.5.2. 案例上传和下载 . . . 200
10.6.6. 3D查询分析展示 . . . 200
10.6.7. 图备份与恢复 . . . 205
10.6.8. 副本迁移 . . . 207
10.6.9. 副本重分布 . . . 209
10.6.10. 索引重建. . . . 209
10.6.11. 监控与运维. . . . 211
10.6.11.1. 磁盘状态监控 . . . 211
10.6.11.2. 日志查看 . . . 211
10.6.11.3. 系统参数查看 . . . 213
10.6.11.4. 集群状态查看 . . . 214
10.6.11.5. 缓存状态查看 . . . 217
10.6.11.6. 图数据库参数配置 . . . 219
10.6.12. 页面权限设置. . . . 219
11. StellarDB认证与权限控制 . . . 223
11.1. StellarDB安全认证 . . . 223
11.1.1. 安全认证方式说明 . . . 223
11.1.2. KG Explorer的用户登录认证. . . . 223
11.1.2.1. 开启KG Explorer的用户登录功能的步骤 . . . 223
11.1.2.2. KG Explorer的用户登录以及登出 . . . 225
11.2. StellarDB权限控制. . . . 226
11.2.1. 开启图权限 . . . 226
11.2.2. 设置用户权限 . . . 227
11.2.3. 权限粒度与类别 . . . 228
11.2.4. 查看用户权限 . . . 229
11.2.5. 应用分析 . . . 230
11.2.6. 附录：常用TEoC关键字与Guardian权限的对应关系 . . . 231
12. 常见问题及报错解决方案 . . . 233
12.1. 常见使用问题 . . . 233
12.2. 常见安装问题 . . . 233
12.3. 常见使用报错及解决方案 . . . 234

12.3.1. KG Explorer执行语句报错 . . . 235
12.3.2. Quark执行语句报错 . . . 236
13. 附录 - 常用系统参数汇总 . . . 239
13.1. 常用服务端口 . . . 239
13.2. KG Explorer常用系统参数 . . . 239
13.2.1. Quark LDAP认证方式相关参数 . . . 239
13.2.2. KG Explorer页面权限和用户校验相关参数 . . . 240
13.2.3. KG Explorer页面失效时间相关参数 . . . 240
13.2.4. JDBC相关参数 . . . 240
13.2.5. 其他参数 . . . 241
13.3. StellarDB常用系统参数 . . . 241
13.4. 数据操作、图算法调用过程常用参数 . . . 241
13.4.1. 创建图时常用参数 . . . 241
13.4.2. 切换查询语言、查询模式参数 . . . 242
13.4.3. bulkload数据校验常用参数 . . . 242
13.4.4. StellarDB重建索引时常用参数 . . . 243
13.4.5. 设置return语句结果中的列名显示参数 . . . 244
13.4.6. 其他参数 . . . 244
13.5. JDBC常用配置参数 . . . 244
13.5.1. JDBC功能性配置参数 . . . 244
13.5.2. JDBC优化参数 . . . 245
14. 客户服务 . . . 246

免责声明

本说明书依据现有信息制作，其内容如有更改，恕不另行通知。星环信息科技（上海）股份有限公司在编写该说明书的时候已尽最大努力保证期内容准确可靠，但星环信息科技（上海）股份有限公司不对本说明书中的遗漏、不准确或印刷错误导致的损失和损害承担责任。具体产品使用请以实际使用为准。

注释：Java® 是Oracle公司在美国和其他国家的商标或注册的商标。Intel® 和Xeon® 是英特尔公司在美国、中国和其他国家的商标或注册的商标。

手册版本信息

版本号：5.0.1

发布日期：2023-08-17

1. StellarDB 5.0.1 变更说明

随着图数据库技术快速迭代和应用场景的不断深化成熟，图数据库用户对图数据库性能和产品化功能有了更多的需求。StellarDB 5.0.0 版本在继承前置版本丰富功能的基础上，进行了大规模的优化和升级，为图数据库用户打造性能更优、数据分析交互能力更强、使用体验更友好的图数据库产品。请参阅《5.0版本变更说明》了解StellarDB图数据库功能和设计内容的变更详情。

此外，在5.0.1版本中，新增如下变更内容：

1.1. 设计变更

1. KG Explorer 中图算法库中图算法不再使用useJNI和baseRDD参数

1.2. 新增功能

1. KG Explorer 新增日志审计功能

1.3. 功能优化

1. StellarDB 优化PageRank、bfs、dfs、shortestpath、dijkstra_shortestpath、spfa_shortestpath、spfa_single_source_shortestpath、scc、hits算法，
2. StellarDB 优化count and degree查询策略
3. StellarDB 优化存储侧文件管理
4. StellarDB 优化批量导数记录文件的管理策略
5. StellarDB 优化存储端线程池的管理
6. KG Explorer 优化创建图时的图名检测规则

1.4. 问题修复

1. StellarDB
    a. 修复若干图算法边界条件异常的问题
    b. 修复配置文件异常导致的Guardian权限页面异常的问题
    c. 修复Guardian安全开启后mixed模式下查询array、decimal类型数据异常的问题
    d. 修复影响力最大化算法结果异常波动的问题
2. KG Explorer
    a. 修复splitSQLBySemicolon时jdbc连接串错误的问题
    b. 修复server.opts不生效、端口校验的问题
    c. 修复KG Explorer中使用图算法时，输入小数变为整数的问题
    d. 修复KG Explorer中使用图算法时，变量赋值语句没有加label导致算法异常的问题
    e. 修复动态图历史结果解析异常的问题

2. StellarDB 5.0.0 版本变更说明

2.1. 设计变更

1. 悬挂边
   a. StellarDB 5.0.0版本对实时写入和批量导入的边，均要求对应的起点和终点是图中存在的点。如果有一个端点不存在，该边数据将被认为是非法数据，不会写入图数据库；
   b. 批量导入时必须先导入点数据，再导入边数据；
   c. 批量删除和truncate label时应先删除边数据，再删除点数据；
   d. 直接删除点数据会产生悬挂边，会影响不同语句统计边的数量：

```
# 下列语句在查询时不会统计悬挂边
match (a)-[f]->(b) return count(f);

# 下列语句在查询时会统计包括悬挂边在内的所有边
match [f] return count(f);
```

   e. 找到悬挂边的方式：

```
match [f] where is_hanging_edge(id(f)) return f;
```

   f. 删除悬挂边的方式：

```
match [f] where is_hanging_edge(id(f)) bulk delete f;
```

   g. 删除点产生悬挂边后，如果再次插入相同uid和label的点，数据库将分配新的内部id，无法和已有的悬挂边关联

```
# 创建uid为1的Person节点指向uid为2的Person节点的有向边
match (a:Person),(b:Person) where uid(a)="1" and uid(b)="2" create (a)-[f:friends]->(b);

# 可以通过match语句查找到这条边
match (a:Person)-[f]->(b:Person) where uid(a)="1" and uid(b)="2" return f;

# 删除uid为2的节点，预期产生一条悬挂边
match (b:Person) where uid(b)="2" delete b;

# 创建Person节点，label为Person，__uid为“2”
create (b:Person {__uid:"2"});

# 无法通过前述match语句查找到这条边，结果为空
match (a:Person)-[f]->(b:Person) where uid(a)="1" and uid(b)="2" return f;
```

2. 系统内映射ID访问

   StellarDB 5.0.0通过使用hex_string_id()函数访问点/边系统内映射ID：

```
match (a) return hex_string_id(a) limit 1;
match [r] return hex_string_id(r) limit 1;
```

3. 点/边数据结构设计变更

系统内置属性	数据类型	所属对象	备注

__uid	string	点/边	不可使用其他类型的数据进行赋值
__tags	array	边	数据存储顺序为正序且不支持在schema中修改；没有值的情况下默认不作为查询结果返回
__srcuid	string	边	边的起始节点的uid
__dstuid	string	边	边的终止节点的uid

a. __uid，__srcuid，__dstuid三个系统内置属性不支持用户通过bulkload、set等语法更新值；

b. 系统内置属性在Restful API的/api/open/query接口不会被filter过滤条件过滤

4. 复制图操作的设计变更

a. StellarDB 5.0.0暂时不支持当前版本内图数据的复制；

b. bulk copy graph 语法当前仅支持StellarDB图数据库版本升级时历史图迁移过程中使用

5. 计算引擎执行模式变更

计算引擎默认使用mixed混合模式，混合模式中默认先执行local模式，超出预设执行时限或者数据量过大时自动转换为cluster模式

6. 图算法设计变更

项目	旧版本	新版本	备注
算法返回属性的参数设置变更	create query temporary graph view sample_1 as (v) [e] with graph_pagerank(@sample_1, "/tmp/view_1", '{factor: 0.85, rounds: 10, limit: 10}', "name") as unapply(vertex, rank, p1) return uid(vertex), rank, p1;	create query temporary graph view sample_1 as (v) [e] with graph_pagerank(@sample_1, "/tmp/view_1", '{factor: 0.85, rounds: 10}') as unapply(vertex, rank) return uid(vertex), rank, vertex.p1;	新版本返回点后，可以通过 . 符号访问点的属性，不再需要在调用算法的参数重进行配置
算法返回点的数据类型变更	点的内部id	点类型	变更后返回点的类型和match语句返回点的类型一致
部分算法返回整型数值的数据类型变更	int类型	long类型	图算法如图直径、半径算法仍然保留int类型作为返回值类型

7. Restful API对外接口设计变更

a. /api/open/query 接口的点和边的内部数据格式发生变更；

i. 该接口返回结果中，"vertices"字段的返回值发生如下变更：去除了“id”字段；去除了“userId”字段，点的userId信息放在“attr”数组中，用“__uid”表示，见示例：

```
# 当前版本
{
    "RowKeyHexString":"30 0 0",
    "label":"v",
    "attr":[
            {
                "fieldName":"__uid",
                "fieldValue":"0"
            },
            {
                "fieldName":"commid",
                "fieldValue":2
            },
            {
                "fieldName":"risk",
                "fieldValue":1
            }
    ]
}

# 旧版本
{
    "RowKeyHexString":"30 0 0",
    "label":"v",
    "id":"0_0",
    "attr":[
        {
            "fieldName":"commid",
            "fieldValue":2
        },
        {
            "fieldName":"risk",
            "fieldValue":1
        }
    ],
    "userId":"0"
}
```

ii. 该接口的"edges"字段的返回结果发生如下变更：去除“srcId”、“dstId”、“srcLabel”、“dstLabel”、“id”字段；边起点“srcUserId”、“dstUserId”字段，放在“attr”数组中，用“__srcuid”、“__dstuid”表示；“attr”数组中添加"__srcuid"、"__dstuid"、“__uid”字段，如下示例：

```
# 当前版本
{
    "RowKeyHexString":"31 30 31 0 0 31 30 32 0 0 1 0 5 0 0 0 5 0 0 0 0 0 0 0",
    "extraId":"",
    "label":"e",
    "__src":"r_2_0_1",
    "__dst":"r_0_0_1",
    "attr":[
            {
                "fieldName":"__uid",
                "fieldValue": ""
            },
            {
                "fieldName":"__srcuid",
                "fieldValue": "0"
            },
            {
                "fieldName":"__dstuid",
                "fieldValue": "1"
            },
            {
                "fieldName":"attr1",
                "fieldValue": "aa"
            }
    ]
}

# 旧版本
{
    "RowKeyHexString":"31 30 31 0 0 31 30 32 0 0 1 0 5 0 0 0 5 0 0 0 0 0 0 0",
    "srcId":"101_0",
    "srcUserId":"101",
    "dstId":"102_0",
    "srcLabel":"v",
    "extraId":"",
    "label":"e",
    "id":"101_0_102_0__1_5_5_0",
    "attr":[
            {
                "fieldName":"attr1",
                "fieldValue": "aa"
            }
        ],
    "dstUserId":"102",
    "dstLabel":"v"
}
```

b. /api/open/data-file/upload上传csv文件接口在返回值中新增uoloadTime字段，数据类型为long

```
# 当前版本
{
     "data": [    //data是个数组，每一项对应一个文件的上传结果
        {
            "fileId": "hive__1663748471349",      //会返回一个文件id， 之后导入数据接口需要传入该参数
            "fileName": "v1.csv",  //文件名
            "msg": "success",     //额外信息，失败时会返回异常信息
            "success": true,    //是否上传成功
            "uploadTime": 1691137981053    //上传时间，long型数据
        }
    ],
    "messageCode": 601,
    "message": "Success",
    "requestTime": 1006
}

# 旧版本
{
     "data": [    //data是个数组，每一项对应一个文件的上传结果
        {
            "fileId": "hive__1663748471349",
            "fileName": "v1.csv",
            "msg": "success",
            "success": true,
        }
    ],
    "messageCode": 601,
    "message": "Success",
    "requestTime": 1006
}
```

2.2. 语法变更

1. bulkload批量数据导入语法变更

使用场景	前置版本语法	5.0.0版本语法	备注
批量数据导入	load node into graph graph_name from source_db.v with file schema "hdfs:///home/load.json";	select str, name from source_db.v bulk upsert (:XX{__uid: uid, prop1: str, prop2: name});	5.0.0废除 load … into 语法，需要使用bulk upsert语法改写相关业务
批量数据导入	select name from table1 upsert (a:Person {n:name})	select name from table1 bulk upsert (a:Person {n:name})	5.0.0使用select * bulk upsert语法代替并废除select * upsert语法
数据脱敏的规则加载	LOAD DESENSITIZATION RULES FROM TABLE [db_name].[tbl_name] INTO GRAPH [graph_name];	无变化	load … from … into 语法仅在该场景中使用

2. 图备份的语法变更

```
# 旧版本语法
load graph graph1 into <hdfs_path>

# 当前版本语法
manipulate graph graph1 backup full <hdfs_path>
```

3. 修改schema的语法变更

使用场景	旧版本语法	5.0.0版本语法
修改schema中点/边的属性	alter_field bank (:EMPLOYEE {emp_addr int})	1. delete_field bank (:EMPLOYEE {emp_addr int}); 2. add_field bank (:EMPLOYEE {emp_addr int});
创建索引	manipulate graph bank create_index vertex (EMPLOYEE.emp_addr),(EMPLOYEE.emp_name),(EMPLOYER.emp_name)	CREATE INDEX IF NOT EXISTS FOR (person) ON [name, age]
删除索引	manipulate graph bank delete_index vertex (EMPLOYEE.emp_addr),(EMPLOYEE.emp_name),(EMPLOYER.emp_name)	DROP INDEX FOR (person) ON [name] IF EXISTS

2.3. 新增内容

1. 动态图模型

StellarDB 5.0.0新增动态图模型。动态图模型用于刻画图数据中的数据变化，包括点/边属性的变化和/

或途中新增、删除点或者边产生的图拓扑结构的变化。参见《动态图模型》小节

2. TEoC
   a. 新增show functions [like <字符串>] 语法查看所有函数或者根据提供的字符串匹配到的函数，详见《查看系统内置函数》小节；
   b. 新增 graph_bsp_khop k跳邻居语法，详见《k跳邻居》小节；
   c. 新增 graph_bsp_sssp 单源最短路径语法，详见《最短路径》小节；
   d. 支持变长路径无上界扩层，详见《多层路径查找》小节；
3. KG Explorer
   a. KG Explorer新增动态图时间轴可视化，通过查询、拖拽、点选的方式即可浏览图数据随时间的变化；
   b. KG Explorer支持动态图时间轴可视化数据样式变化规则的配置；
   c. KG Explorer新增支持在画布中显示选中节点的最短路径和全路径；
   d. KG Explorer支持对当前画布中数据案例进行下载和上传的功能；
   e. KG Explorer支持更丰富的样式配置，新增图例配置；
   f. KG Explorer支持副本均衡功能；
   g. KG Explorer新增画布操作撤销和回退功能；
   h. KG Explorer新增schema预览功能；
   i. KG Explorer新增语句收藏和记录功能；
   j. KG Explorer新增新手引导

   新版本KG Explorer改动内容较多，详细介绍见《KG Explorer》章节。

2.4. 优化内容

1. TEoC
   a. 优化OPTIONAL MATCH语法，详见《OPTIONAL MATCH》小节
2. KG Explorer
   a. 优化可视化数据导入流程；
   b. 优化页面权限管理功能

2.5. Deprecated（功能废除清单）

变更项目	前置版本	5.0.0版本变更	备注
系统内映射ID访问	startid()	废除	可根据业务具体情况选择其他函数替换，例如startuid()
系统内映射ID访问	endid()	废除	可根据业务具体情况选择其他函数替换，例如enduid()
系统内映射ID访问	id()	废除	可根据具体情况选择其他函数替换，例如uid()

变更项目	前置版本	5.0.0版本变更	备注
图算法	create query temporary graph view sample_1 as (v) [e] with graph_degree(@sample_1, "/tmp/view_1", '{direction: "out", limit: 10}') as unapply(vertex, degree) return node_rk_to_uid(vertex), degree;	create query temporary graph view sample_1 as (v) [e] with graph_degree(@sample_1, "/tmp/view_1", '{direction: "out", limit: 10}') as unapply(vertex, degree) return uid(vertex), degree;	
图算法	支持Global Cluster Coefficient算法	暂不不支持Global Cluster Coefficient算法	
图算法	支持Eccentricity算法	暂不不支持Eccentricity算法	

2.6. 已知问题

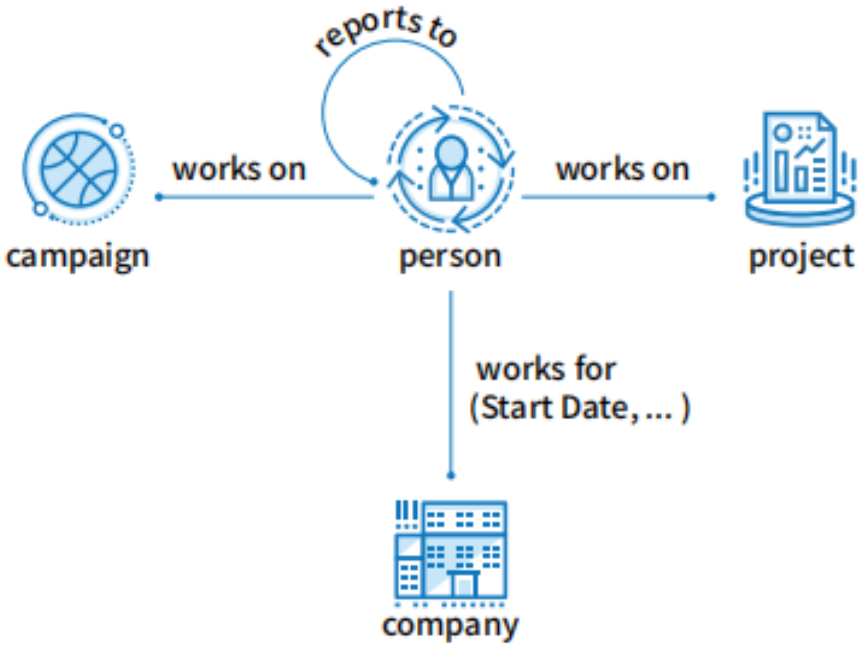
1. StellarDB 5.0.0版本内复制图暂不可用，后续迭代支持；
2. match (a) bulk delete a & truncate label会产生悬挂边；
3. __srcuid 和 __dstuid 属性的权限管理不够完善；
4. 当前版本不支持单条语句中对点属性和边属性同时进行值更新，例如 match (a)-[f]→(b) bulk set a.xx = 1 and f.xx = 2；
5. KG Explorer未完整支持Pearson、Cosine、Overlap、Euclidean Distance相似度算法的两种计算方式

3. 简介

Transwarp StellarDB是一款为企业级图应用而打造的分布式图数据库，不仅可以用于快速查找数据间的关联关系，而且还可以提供强大的算法分析能力。StellarDB克服了海量关联图数据存储的难题，通过自定义图存储格式和集群化存储，实现了传统数据库无法提供的低延时多层关系查询。在社交网络、公安、金融等领域都有巨大应用潜力。

3.1. 产品背景

图（Graph）是一种由**点**和**边**组成的半结构化数据，用于映射事物之间的关系，如人际关系、交易往来、交通道路等模型。属性图（Property Graph）是近年来兴起的一种图模型，可以在点、边上自由定义属性和类型，从而形成社交网络、交易网络等复杂图。



传统关系型数据库擅长处理拥有固定结构的表格型数据，数据之间的关联关系需要通过一些JOIN操作来得到。但是，在数据量增长或数据类型复杂时，关系型数据库会存在以下几个瓶颈：

1. 存在大量JOIN操作

   为了获得数据之间的连接信息，关系型数据库不得不通过JOIN的方法来取得“下一跳”节点。大量的JOIN操作不仅对计算资源造成极大浪费，也无法快速返回数据结果。

2. 固化的数据模型

   图数据在应用场景中可能频繁地修改数据模型，关系型数据库在应对这种场景时，对用户的模型设计能力要求极高。

关系型数据库由于数据模型限制而无法适配图场景，图数据库因此孕育而生。相较于关系型数据库，图数据库在以下方面具有优势：

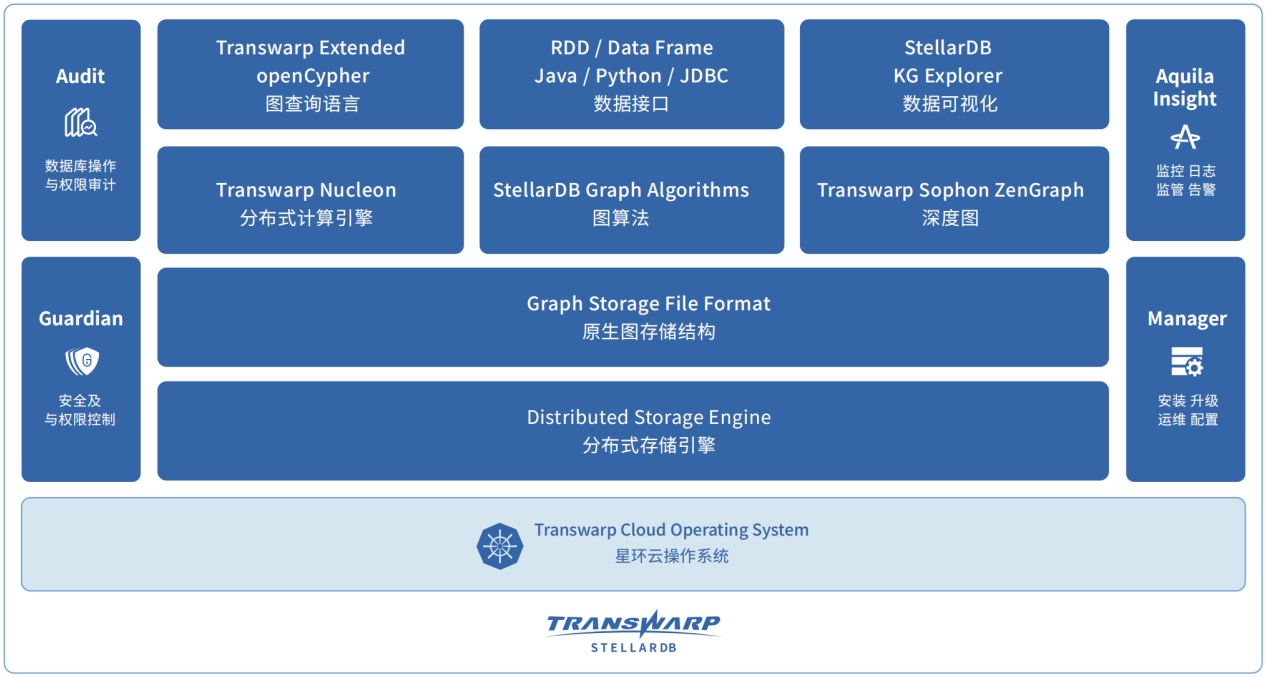
- 拥有灵活可变的数据结构；

- 充分利用图的内联信息，可存储规模庞大的关系；
- 实时返回查询结果；

图数据库把边视作数据的一种，将关系型数据库的JOIN操作转换为图数据库的一次普通查询。在数据量增加时，JOIN操作会急剧增加查询的开销，但图数据库仅会增加少量开销。随着图数据数量增加，单机系统在计算和存储上存在明显瓶颈，分布式图数据库是未来的趋势。

星环科技StellarDB是一款完整的分布式图数据库产品，支持企业级图数据的存储、查找、分析和展现。结合原生存储引擎和计算引擎，StellarDB可以轻松实现数千亿边规模的海量图存储、实时数据插入更新、10层以上深度链路查询，以及复杂算法分析。

3.2. 产品架构



3.3. 产品组成

作为星环大数据平台的重要组件，StellarDB架构主要由以下部分组成：

3.3.1. 存储引擎

图数据以高效的压缩格式存储于星环分布式存储引擎中。借助图分区算法，图数据可按策略分散存储于集群中，拥有良好的可扩展性，并具备存储任意规模图的理论能力。

存储引擎架构为Master-Worker结构，多个Master组成的Master Group负责元信息管理、任务调度、负载均衡等功能；Worker存储图数据，并提供数据读取、更新和删除功能。存储引擎通过Raft协议来保证数据一致性和高可用性。

3.3.2. 计算引擎

借助星环分布式计算引擎Quark的计算分析能力，计算能力随着节点数目增长线性扩展。StellarDB可同时为用户提供实时图查询和离线算法分析，进而支持海量边点的大图分析。

计算引擎和存储引擎同机部署。利用数据locality特性加速图计算和分析任务。计算引擎内置了部分常用图算法，并以RDD的方式提供数据和计算的接口。

3.3.3. 可视化引擎

StellarDB提供网页可交互分析工具KG Explorer。用户可以在界面上输入并执行查询语句，并基于查询结果做进一步的数据分析，或者通过业务数据和图谱模型来构建新图谱。

4. 快速上手

本章节将引导您快速熟悉StellarDB，并为您初步介绍如何通过KG Explorer和beeline客户端操作StellarDB。其中，"StellarDB初探"一节通过构建一张人物关系图，从零介绍如何在StellarDB进行基本操作；"StellarDB进阶"一节为您提供了内置于StellarDB的《哈利·波特》人物关系图，帮助您进一步探索StellarDB。

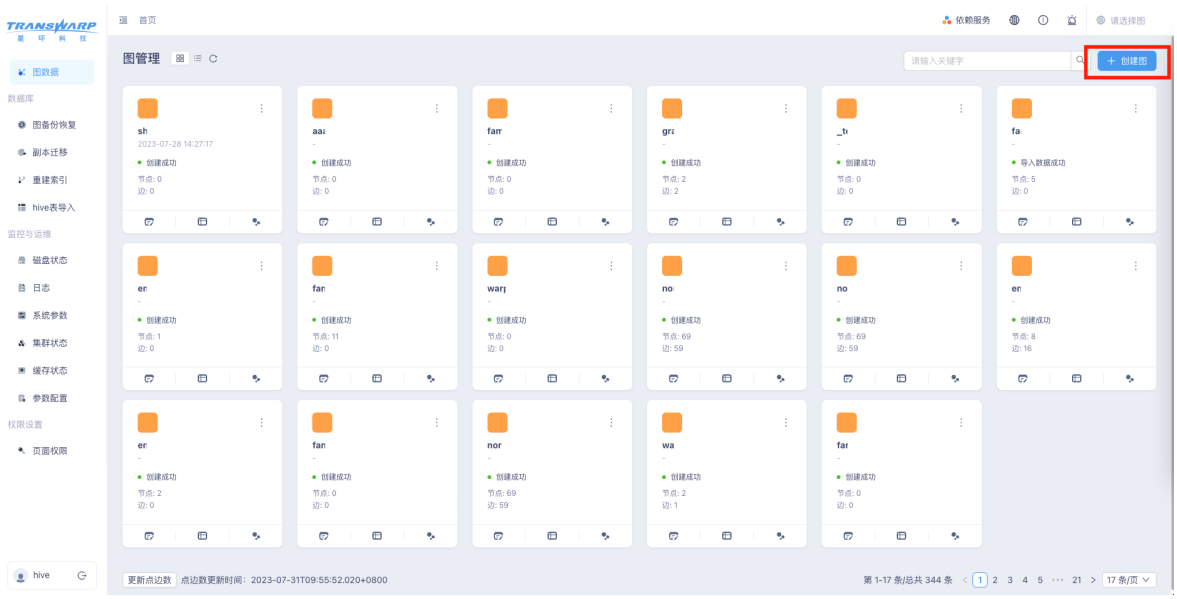
4.1. StellarDB初探

4.1.1. 使用KG Explorer构建图

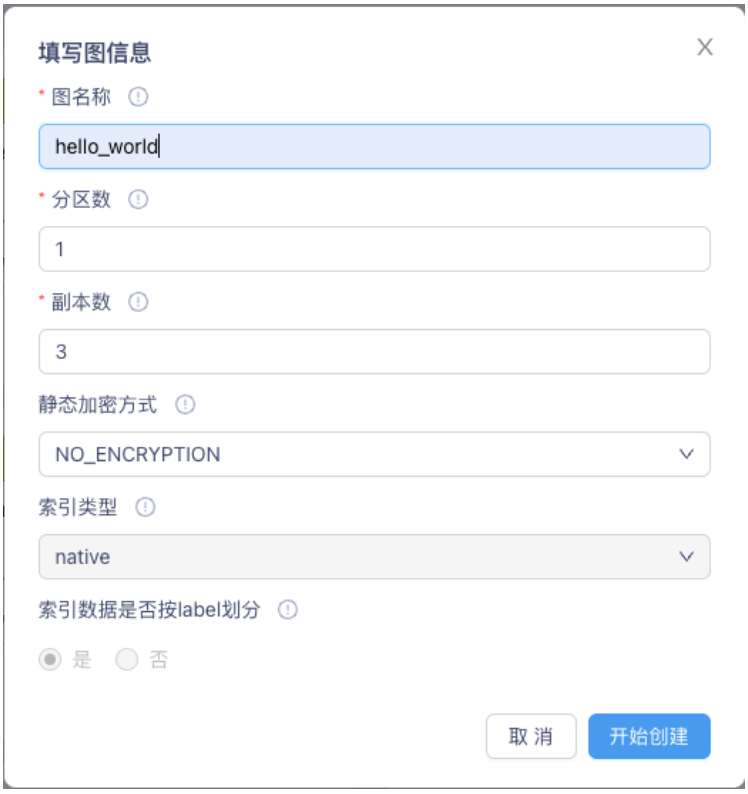
1. 从Manager页面进入KG Explorer页面。若KG Explorer开启了单点登录，会自动跳转Federation登录页面，按如图方式登录：

KG Explorer用户开启方法以及详细使用说明请查看章节《KG Explorer使用文档》。

2. 点击 **登录** 后进入KG Explorer主页面。我们首先需要构建图名为"hello_world"的图。
   a. 在主页面右上角点击创建图按钮开始图谱schema的构建。



b. 按照引导填写图基本信息后点击确定进入构建页面。



c. 在画布中，我们为"hello_world"图创建Boy和Girl两种类型的点，两种类型的点均包含name、salary、age、single四个属性，数据类型分别为string、double、int、boolean；图中创建Friend和Likes类型的边，边属性均为since，数据类型为int。

3. 创建"hello_world"的schema后，返回“图管理”页面，点击"hello_world"图名下方的“图探索”按钮进入图数据交互页面。
4. 在"hello_world"图中插入点数据。请在查询语句输入框中分别输入、执行下列语句。

```
# 插入一个类型为Girl的点，并对属性赋值，__uid为1
create (:Girl {name:"Kitty", age:23, salary:4250.95, single:true, __uid:"1"});

# 插入两个点，类型分别为Girl和Boy，不对属性赋值，__uid分别为2和1
create (:Girl {__uid: "2"}), (:Boy {__uid:"1"});

# 插入一个点，类型为Girl，对name属性赋值Lisa，__uid为3，设置点标签为human和student
create (:Girl {__uid: "3", name: "Lisa", __tags: ["human", "student"]});
```

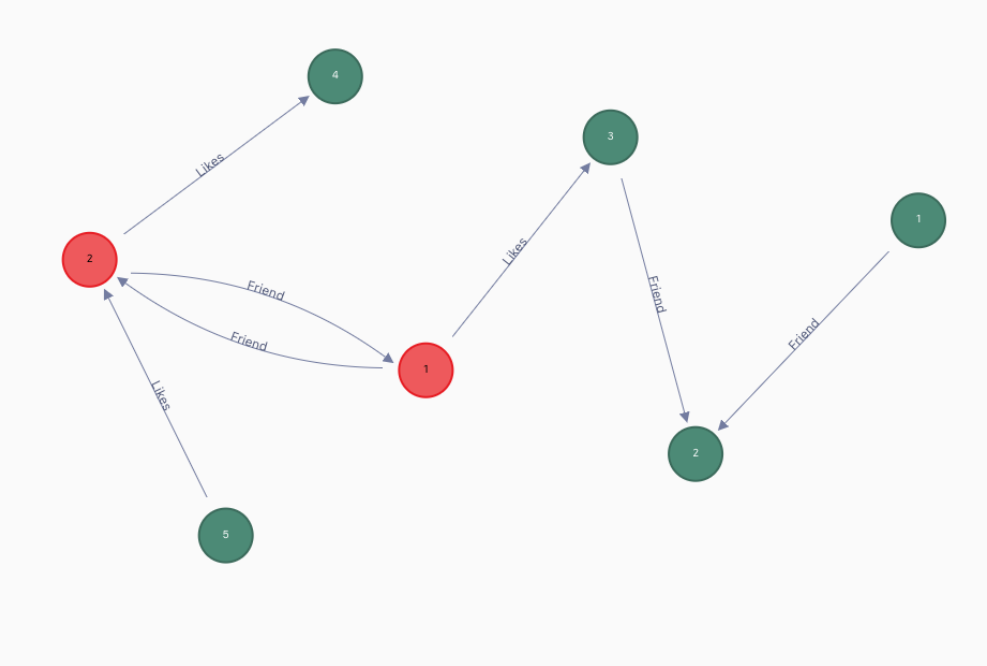
5. 在"hello_world"图中插入边数据。请在查询语句输入框中分别输入、执行下列语句。

```
# 插入一条类型为Likes的由John指向Rose有向边，插入边的同时插入两个点，类型分别为Boy和Girl，并且对name
属性赋值，__uid分别为2和4
create (:Boy {__uid:"2", name:"John"})-[:Likes]->(:Girl {__uid: "4", name: "Rose"});

# 插入一条类型为Likes的由Amy指向John有向边，插入边的同时插入一个点，类型为Girl，并且对name属性赋值Amy，__uid
为5
match (a:Boy {__uid:"2"}) create (a)<-[:Likes]-(:Girl {__uid:"5", name: "Amy"});

# 插入一条类型为Friend的连接John和__uid为1的Boy节点的无向边
match (a:Boy {__uid:"1"}),(b:Boy {__uid:"2"}) create (a)-[:Friend]-(b);

# 插入两条类型为Friend的有向边，其中一条是从__uid为1的Girl类型节点指向__uid为2的Girl
类型节点，另外一条是从__uid为3的Girl类型节点指向__uid为2的Girl类型节点
match (a:Girl {__uid:"1"}),(b:Girl {__uid:"2"}),(c:Girl {__uid:"3"}) create (a)-[:Friend]->(b),
(b)<-[:Friend]-(c);

# 直接创建uid为1的Boy到uid为3的Girl的单向边,并指定边的__uid为relation_1
create [:Likes {__uid:"relation_1",__usid:"1",__udid:"3",__uslabels:["Boy"],__udlabels:["Girl"]
}];
```

6. 图查询：在查询输入框中输入并执行下列查询语句。
   a. 查询图中所有相连的点，并返回所有点和边。

```
match (m)-[f]-(n) return m,f,n;
```

   查询结果如图示：



b. 查询John喜欢的人的名字。

```
match (a:Boy {name:"John"})-[:Likes]->(b:Girl) return b.name;
```

查询结果如图示：

match (a:Boy {name:"John"})-[:Likes]->(b:Girl) return b.name;

ID	_a_c0
0	Rose

c. 查询John的朋友的朋友的名字。

```
match (a:Boy {name:"John"})-[:Friend*..2]->(b) return b.name;
```

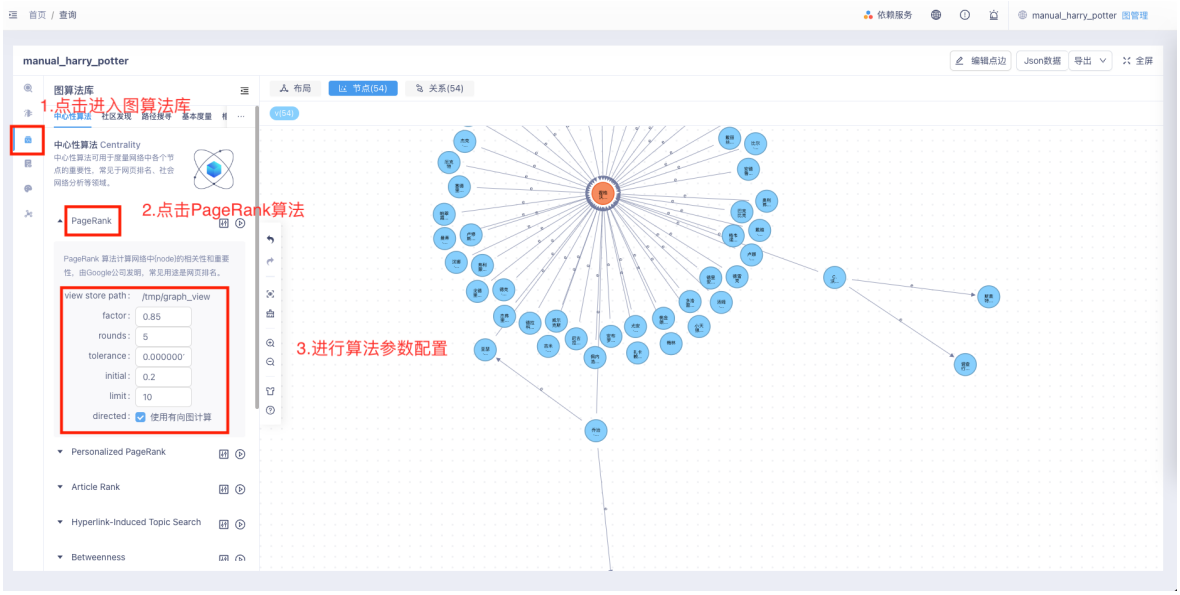
查询结果如图所示。（该语句执行结果返回两个Boy类型的点，是因为在创建__uid为1的Boy类型点时没有对name属性赋值，所以显示null）

match (a:Boy {name:"John"})-[:Friend*..2]->(b) return b.name;

ID	_a_c0
0	null
1	John

7. 算法调用（以PageRank为例）

a. 使用KG Explorer图算法库调用：如图示，在KG Explorer页面进入图示图算法库界面。点击 **PageRank** 算法，并且配置相应参数，点击 **执行** 按钮执行算法。



b. 或使用TEoC语句调用：在查询输入框中输入并执行下列语句。

```
create query temporary graph view hello_world_sample as (v) [e] with graph_pagerank
(@hello_world_sample,"/tmp/hello_world_view", '{factor: 0.85, rounds: 10, limit: 10}') as
unapply(vertex, rank) return uid(vertex), rank;
```

执行结果如图示。

ID	_a_c0	rank
0	5	0.06873759177429462
1	4	0.14975618309870203
2	2	0.1909064373948816
3	2	0.2396306388940223
4	1	0.14975618309870203
5	3	0.13247537396510292
6	1	0.06873759177429462

目前KG Explorer提供部分图算法的可视化调用，其他内置图算法需要使用语句调用图算法，请参考章节《图算法》了解如何使用语句调用图算法。

8. 删除数据

在查询语句输入框中输入并执行下列语句进行删除操作。

a. 删除name属性为Amy的点。

```
match (a) where a.name="Amy" delete a;
```

b. 删除label为Likes的边。

```
match [f:Likes] delete f;
```

9. 删除图

在图管理页面找到"hello_world"图，点击图名右上角"三个点"的图标，点击删除图，删

除"hello_world"图。

4.1.2. 使用beeline构建图

在进入命令行之前，请用户确保已经在当前的操作节点上已安装TDH Client。具体步骤请参考 《TDH安装手册》。

1. 进入命令行

   在本小节中，将使用 **server_ip|hostname** 来指代Quark Server所在的节点名称或ip。Quark Server所在节点可以通过管理界面的Quark角色页面查看。

   用户可以选择用不同的安全认证方式在任意一台服务器上登录beeline客户端，登陆方式如下：

安全认证方式	登陆指令
没有安全认证	beeline -u "jdbc:hive2://<server_ip/hostname>:10000/" 例如： beeline -u "jdbc:hive2://172.18.20.35:10000/"
LDAP认证	beeline -u "jdbc:hive2://<server_ip/hostname>:10000/<database_name>" -n <username> -p <password> 例如： beeline -u "jdbc:hive2://172.18.20.35:10000/" -n admin -p 123456
Kerberos认证	beeline -u "jdbc:hive2://<server_ip/hostname>:10000/<database_name>;principal=<princpal_name>" 例如： beeline -u "jdbc:hive2://172.18.20.35:10000/;principal=hive/idc36@TDH"

   (1). 使用客户端前请执行source /<tdh-client_install_path>/TDH-Client/init.sh

   (2). 关于安全认证的详细介绍请参考 Transwarp Guardian手册

2. 登陆beeline客户端成功后，执行下列语句将查询语言切换为TEoC。

```
config query.lang cypher;
```

3. 执行下列语句查看是否有"hello_world"同名的图，若有，请在后续操作中按需修改查询语句，或删除原"hello_world"图，再进行后续操作。

```
show stellargraphs;
```

4. 参照前文步骤2-4进行图数据库创建和图数据的填充。这里不再赘述。
5. 执行下列查询语句，校验"hello_world"点数据是否正确填充。

```
match (n) return n.__uid,labels(n),n.name;
```

   查询结果如图示：

```
+--------+----------+--------+
| _a_c0  |   _a_c1  | _a_c2  |
+--------+----------+--------+
| 1      | ["Boy"]  | NULL   |
| 1      | ["Girl"] | Kitty  |
| 4      | ["Girl"] | Rose   |
| 3      | ["Girl"] | Lisa   |
| 2      | ["Boy"]  | John   |
| 2      | ["Girl"] | NULL   |
| 5      | ["Girl"] | Amy    |
+--------+----------+--------+
```

6. 执行下列查询，校验"hello_world"边数据是否正确填充。

```
match [f] return f.__uid,labels(f),startuid(f),enduid(f);
```

7. 图查询
   a. 查询John喜欢的女孩的名字。

```
match (a:Boy {name:"John"})-[:Likes]->(b:Girl) return b.name;
```

   结果如图示：

   b. 查询John的朋友的朋友的名字。

```
match (a:Boy {name:"John"})-[:Friend*..2]->(b) return b.name;
```

   结果如图示：

8. 算法调用（以PageRank算法为例）

   在beeline客户端中输入并执行下列语句调用PageRank算法，其他算法使用方法和所示方法相同。

```
create query temporary graph view hello_world_sample as (v) [e] with graph_pagerank
(@hello_world_sample,"/tmp/hello_world_view", '{factor: 0.85, rounds: 10, limit: 10}') as
unapply(vertex, rank) return uid(vertex), rank;
```

   结果如图示：

```
+--------+----------------------+
| _a_c0  |         rank         |
+--------+----------------------+
| 3      | 0.13247537396510292  |
| 4      | 0.14975618309870203  |
| 1      | 0.06873759177429462  |
| 1      | 0.14975618309870203  |
| 2      | 0.2396306388940223   |
| 5      | 0.06873759177429462  |
| 2      | 0.1909064373948816   |
+--------+----------------------+
```

9. 删除数据
   a. 删除name属性为Amy的节点。

```
match (a) where a.name="Amy" delete a;
```

   b. 删除label为Likes的边。

```
match [f:Likes] delete f;
```

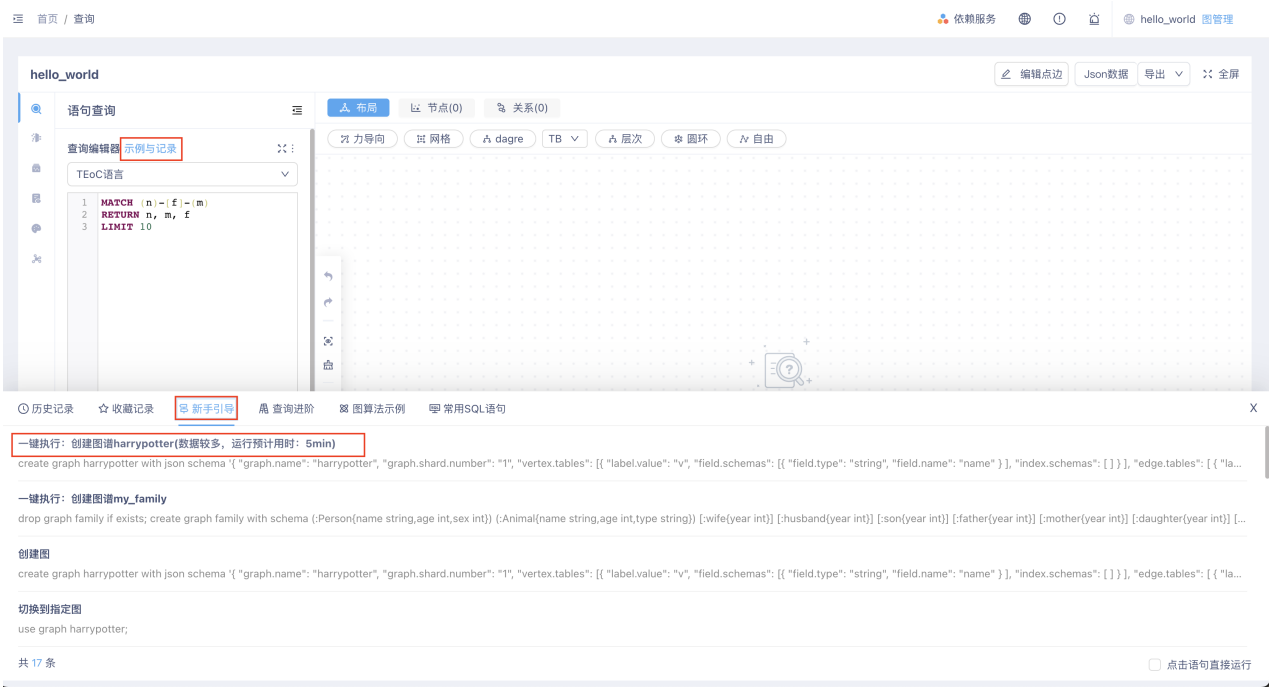
10. 删除图"hello_world"

```
drop graph hello_world;
```

4.2. StellarDB进阶

4.2.1. 使用KG Explorer交互

在上一小节的2D页面，点击“示例与记录”按钮，在弹出窗口中点击“新手引导”按钮，选择一键执行创建harrypotter图谱。如下图所示：



在图管理页面找到"harrypotter"图，点击图名下方”图探索“按钮进入图数据分析查询页面，在查询框中输入查询语句并执行。

+ 查询语句：

a. 查询哈利·波特与赫敏·格兰杰的关系

```
match p=(a)-[]->(b) where a.name="哈利·波特" and b.name starts with"赫敏" return p;
```

b. 查询马尔福家族人员的从属组织

```
match p=(a)-[f]->(b) where f.relation="从属" and a.name ends with"马尔福" return p;
```

c. 查询凤凰社的全体成员

```
match p=(a)-[f where f.relation="从属"]->(b where b.name="凤凰社") return p;
```

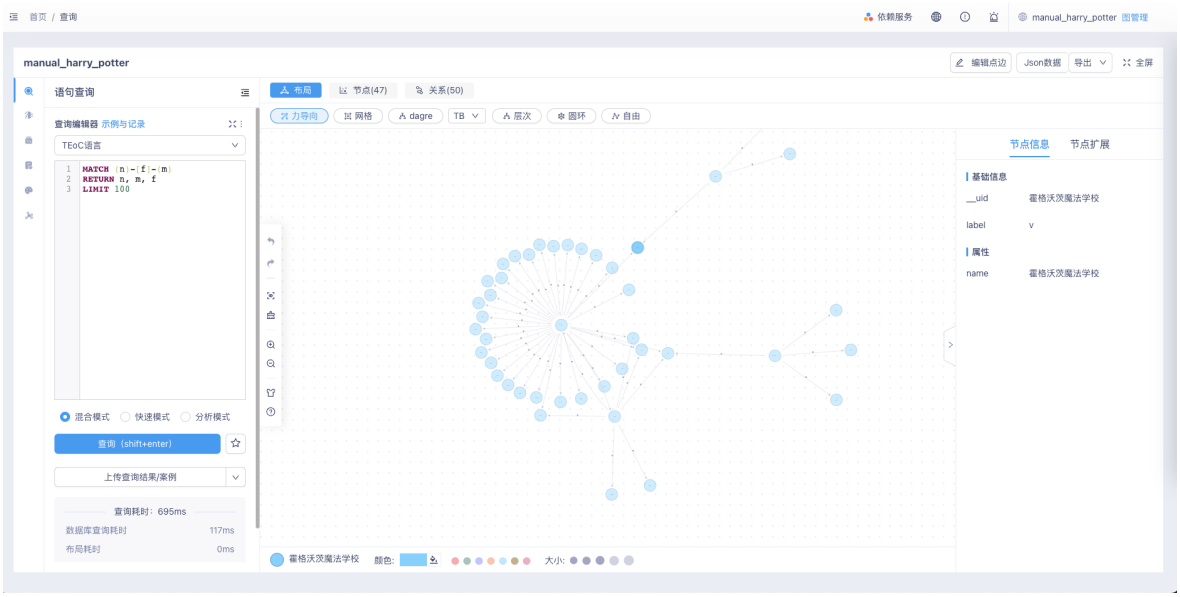
d. 查询阿不思·邓布利多的学生的职业

```
match p=(a)<-[f]-(b)-[]->(c) where c.name ends with "邓布利多" and f.relation="职业" return p;
```

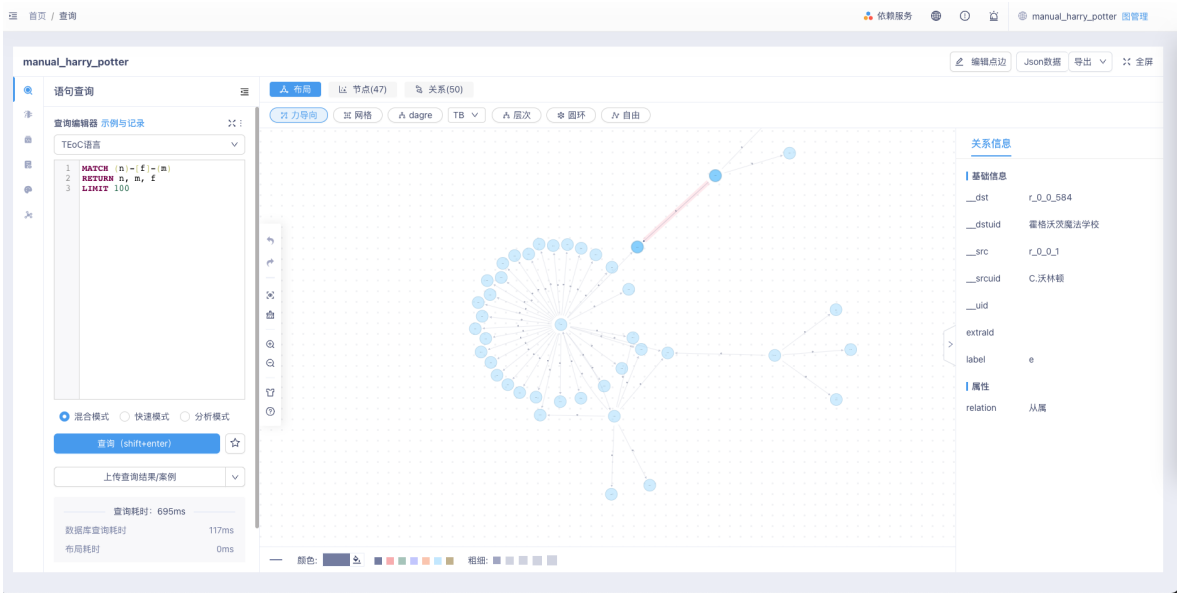
e. 查询伏地魔与哈利·波特的4层以内人际关系网

```
match p=(a)-[f*..4]->(b) where a.name="哈利·波特" and b.name="伏地魔" return p;
```

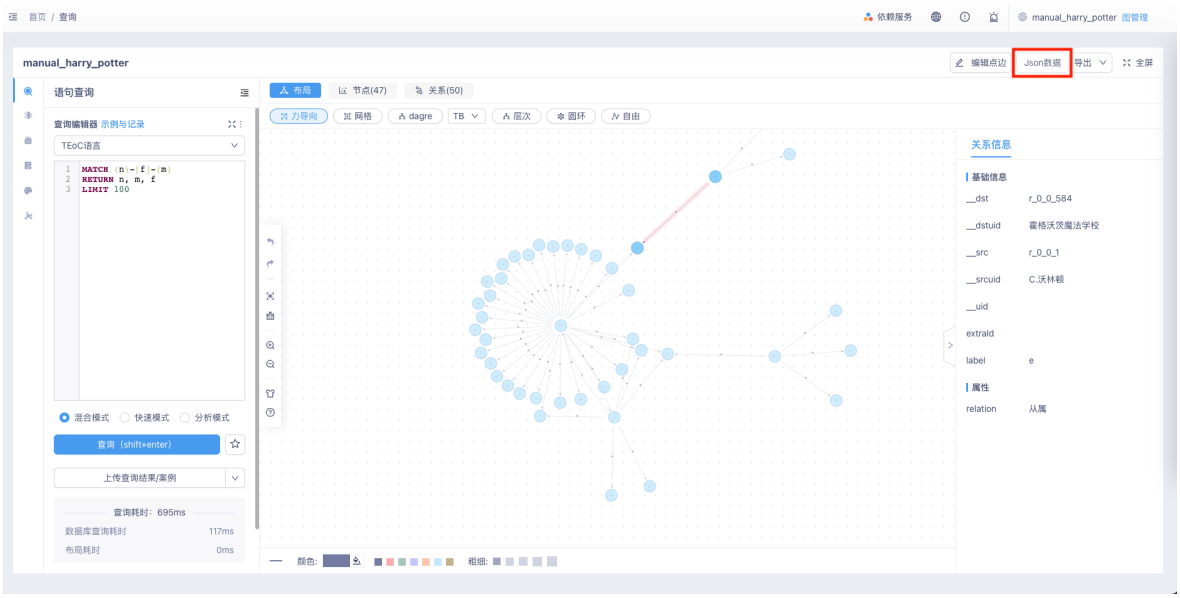
1. 图形化展示界面中单击某点可以显示该点的详细信息。

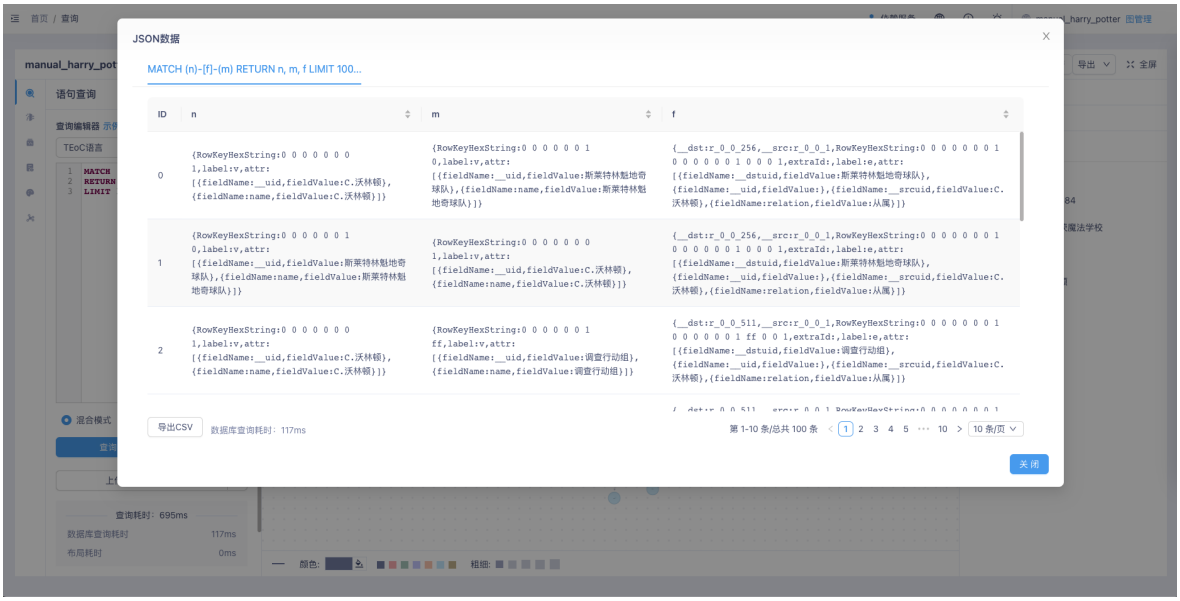


2. 点击边可以查看该边的详细信息。



3. 点击JSON数据可以查看查询结果。





关于KG Explorer的详细介绍在后续章节《KG Explorer使用文档》中查看

4.2.2. 使用beeline交互

1. 登录成功后执行下面指令切换语言为TEoC模式。

```
config query.lang cypher;
```

2. 执行下面指令查看已存在的图。

```
show stellargraphs;
```

执行结果如下图示：

```
+-----------------------------------+
|               _a_c0               |
+-----------------------------------+
| __system_meta_graph_stellarweb2   |
| harrypotter                       |
+-----------------------------------+
```

上图中__system_meta_graph_KGExplorer2是存放KG Explorer元数据信息的图，"harrypotter"是我们使用KG Explorer创建的图。

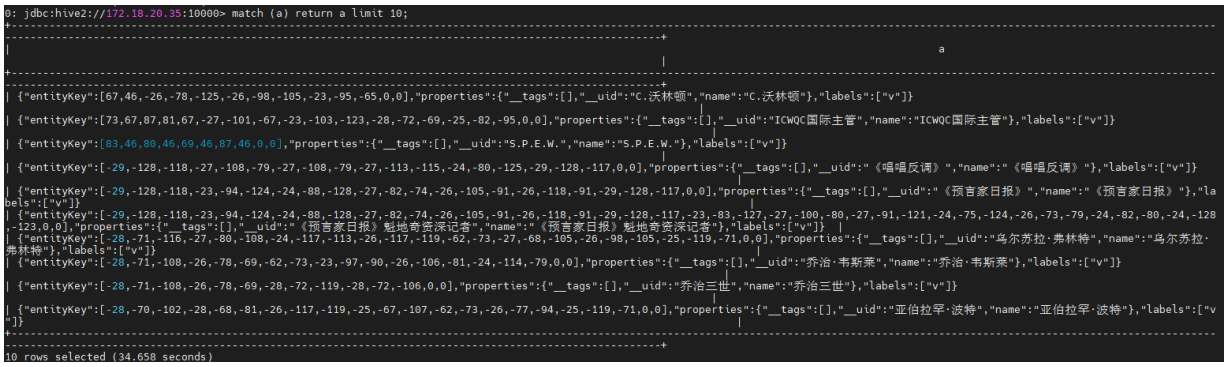
3. 执行下面指令选择使用KG Explorer创建的图"harrypotter"。

```
use graph harrypotter;
```

4. 执行下面指令查找图中的点，并显示其中的10条内容。

```
match (a) return a limit 10;
```

执行结果如下图示：



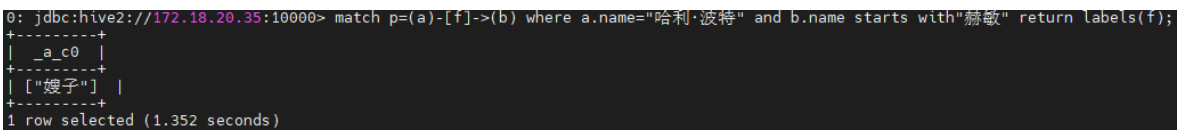
其中entitykey为点的rowkey的特殊编码形式，properties为点的属性信息。

5. 执行下列语句查询哈利·波特与赫敏·格兰杰的关系，其中”->”代表方向，查询结果会随着查询方向的改变而改变。

   a. 查询语句1：

```
match (a)-[f]->(b) where a.name="哈利·波特" and b.name starts with"赫敏" return f.relation;
```

   执行结果如下图示：



   b. 查询语句2：

```
match (a)<-[f]-(b) where a.name="哈利·波特" and b.name starts with"赫敏" return f.relation;
```

   执行结果如下图示：

```
0: jdbc:hive2://172.18.20.35:10000> match (b)-[f]->(a) where a.name="哈利·波特" and b.name starts with"赫敏" return labels(f);
+---------+
|  _a_c0  |
+---------+
| ["从属"]  |
| ["妹夫"]  |
+---------+
2 rows selected (0.569 seconds)
```

6. 执行下列语句查询马尔福家族人员的从属组织。

```
match (a)-[f]->(b) where f.relation="从属" and a.name ends with"马尔福" return a.name as 成员,b
.name as 组织;
```

执行结果如下图示：

```
0: jdbc:hive2://172.18.20.35:10000> match (a)-[f:从属]->(b) where a.name ends with"马尔福" return a.name as 成员,b.name as 组织;
+-------------+-------------+
|     成员      |     组织      |
+-------------+-------------+
| 德拉科·马尔福     | 布莱克家族       |
| 德拉科·马尔福     | 德拉科·马尔福一伙人  |
| 卢修斯·马尔福     | 斯莱特林        |
| 纳西莎·马尔福     | 斯莱特林        |
| 阿布拉克萨斯·马尔福  | 斯莱特林        |
| 德拉科·马尔福     | 斯莱特林学院      |
| 德拉科·马尔福     | 斯莱特林魁地奇球队   |
| 德拉科·马尔福     | 格林格拉斯家族     |
| 德拉科·马尔福     | 调查行动组       |
| 卢修斯·马尔福     | 霍格沃茨魔法学校    |
| 德拉科·马尔福     | 霍格沃茨魔法学校    |
| 阿布拉克萨斯·马尔福  | 霍格沃茨魔法学校    |
| 卢修斯·马尔福     | 食死徒         |
| 德拉科·马尔福     | 食死徒         |
| 卢修斯·马尔福     | 马尔福家族       |
| 德拉科·马尔福     | 马尔福家族       |
| 纳西莎·马尔福     | 马尔福家族       |
| 阿布拉克萨斯·马尔福  | 马尔福家族       |
| 卢修斯·马尔福     | 魔法部         |
| 塞普蒂默斯·马尔福   | 魔法部         |
| 德拉科·马尔福     | 魔法部         |
| 卢修斯·马尔福     | 鼻涕虫俱乐部      |
+-------------+-------------+
22 rows selected (0.554 seconds)
```

7. 执行下列语句查询凤凰社的全体成员。

```
match (a)-[f {relation:"从属"}]->(b where b.name="凤凰社") return a.name as 凤凰社成员;
```

执行结果如下图示：

```
0: jdbc:hive2://172.18.20.35:10000> match p=(a)-[f:从属]->(b where b.name="凤凰社") return a.name as 凤凰社成员;
+-------------+
|   凤凰社成员    |
+-------------+
| 乔治·韦斯莱      |
| 亚瑟·韦斯莱      |
| 卢娜·洛夫古德     |
| 吉迪翁·普威特     |
| 哈利·波特       |
| 小天狼星·布莱克    |
| 尼法朵拉·唐克斯    |
| 巴克比克        |
| 弗雷德·韦斯莱     |
| 查理·韦斯莱      |
| 比尔·韦斯莱      |
| 爱米琳·万斯      |
| 米勒娃·麦格      |
| 罗恩·韦斯莱      |
| 芙蓉·德拉库尔     |
| 莉莉·伊万斯      |
| 莱姆斯·卢平      |
| 西弗勒斯·斯内普    |
| 西比尔·特里劳尼    |
| 詹姆·波特       |
| 费比安·普威特     |
| 赫敏·格兰杰      |
| 金妮·韦斯莱      |
| 阿不思·邓布利多    |
| 阿不福思·邓布利多   |
| 阿拉斯托·穆迪     |
| 马琳·麦金农      |
| 鲁伯·海格       |
+-------------+
28 rows selected (0.695 seconds)
```

8. 执行下列语句查询阿不思·邓布利多的学生的职业。

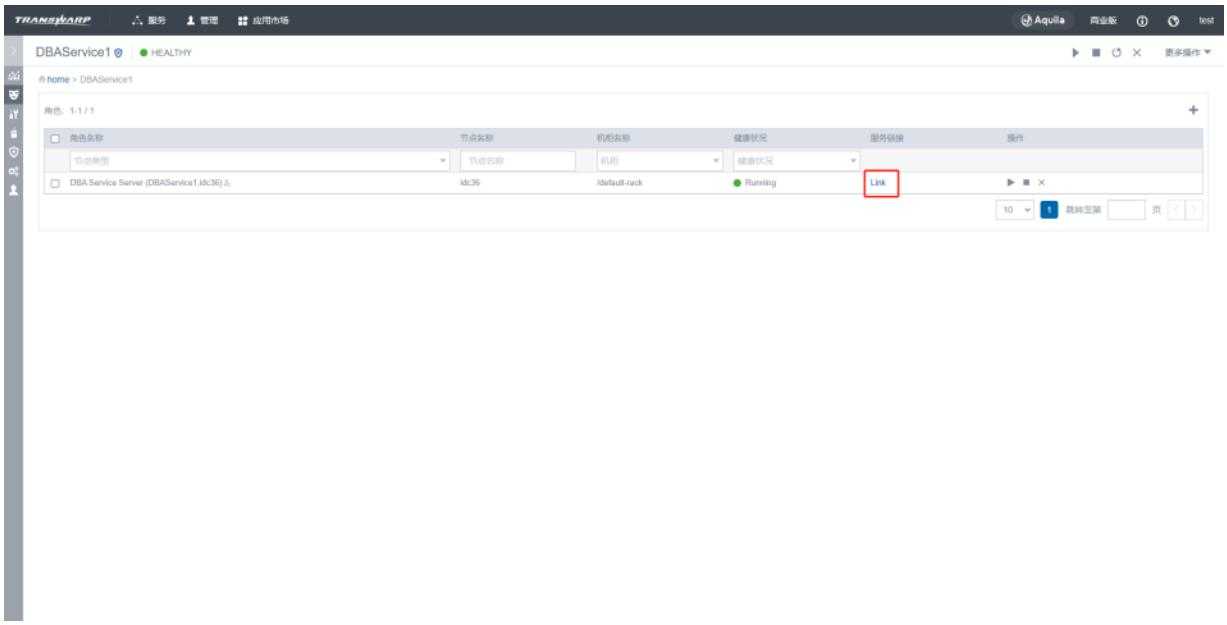
```
match (a)<-[f]-(b)-[:e {relation:"学生"}]->(c) where f.relation="职业" and c.name ends with
"邓布利多" return b.name as 学生姓名, a.name as 职业;
```

执行结果如下图示：

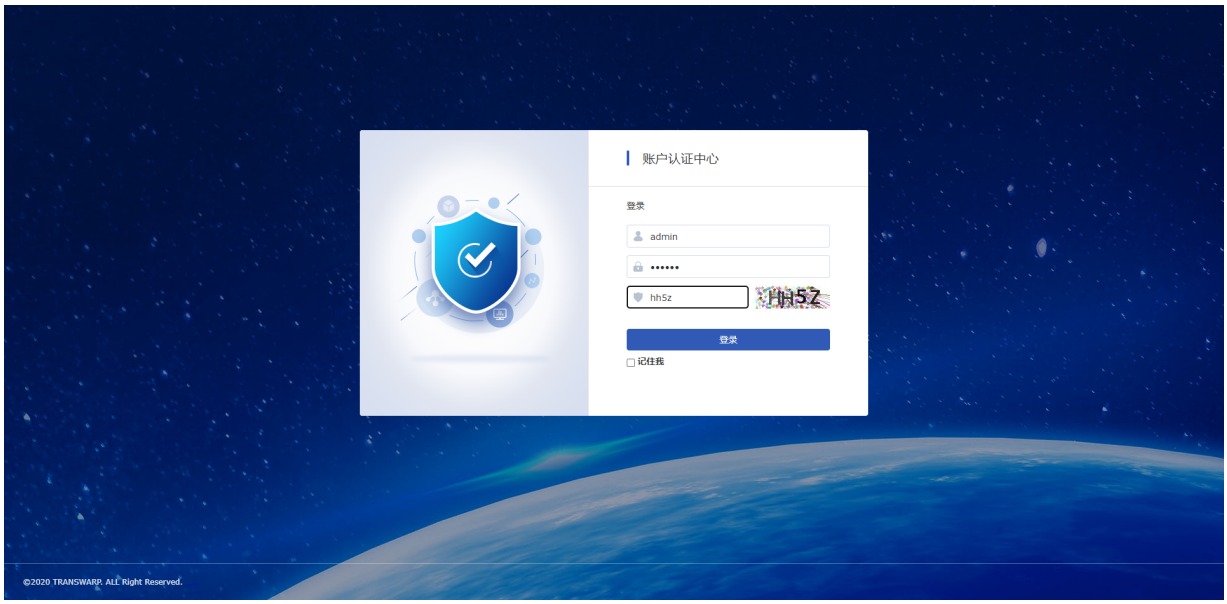
```
0: jdbc:hive2://172.18.20.35:10000> match (a)<-[f:职业]-(b)-[:学生]->(c) where c.name ends with "邓布利多" return b.name as 学生姓名, a.
name as 职业;
+---------+----------------+
|  学生姓名  |       职业       |
+---------+----------------+
| 哈利·波特  | 傲罗办公室主任        |
| 哈利·波特  | 黑魔法防御术兼职讲师     |
| 赫敏·格兰杰 | 魔法法律执行司副司长     |
| 赫敏·格兰杰 | 魔法生物管理控制司      |
| 金妮·韦斯莱 | 《预言家日报》魁地奇资深记者 |
| 金妮·韦斯莱 | 霍利黑德哈比队魁地奇球员   |
+---------+----------------+
6 rows selected (0.923 seconds)
```

4.2.3. 使用DBAService查看任务信息

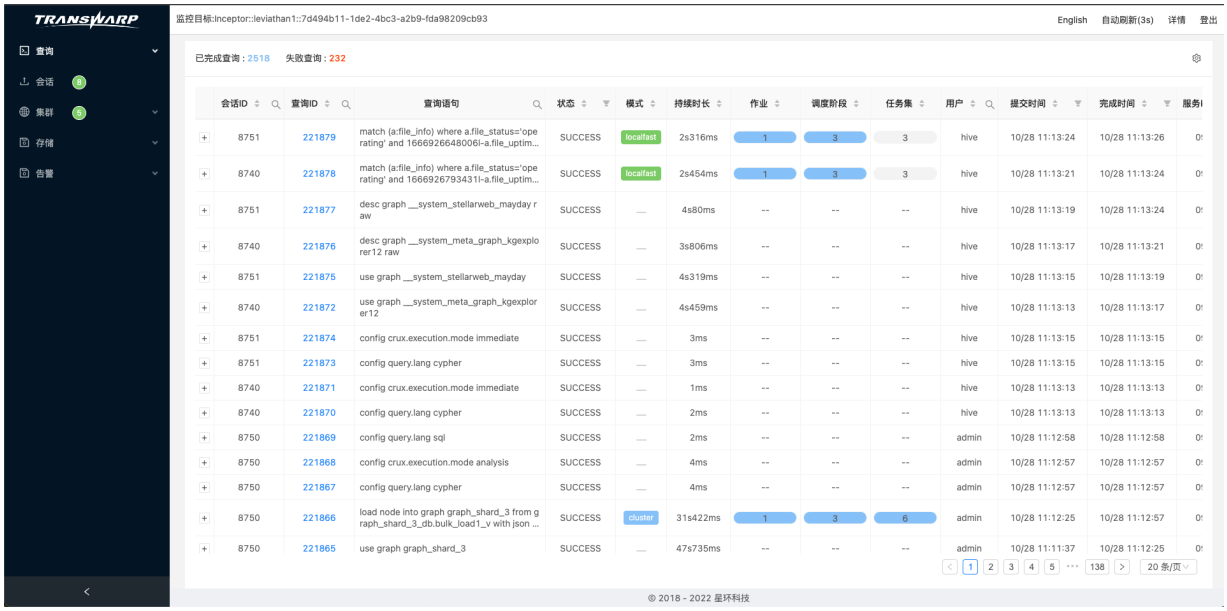
1. 通过Manager界面进入DBAService。

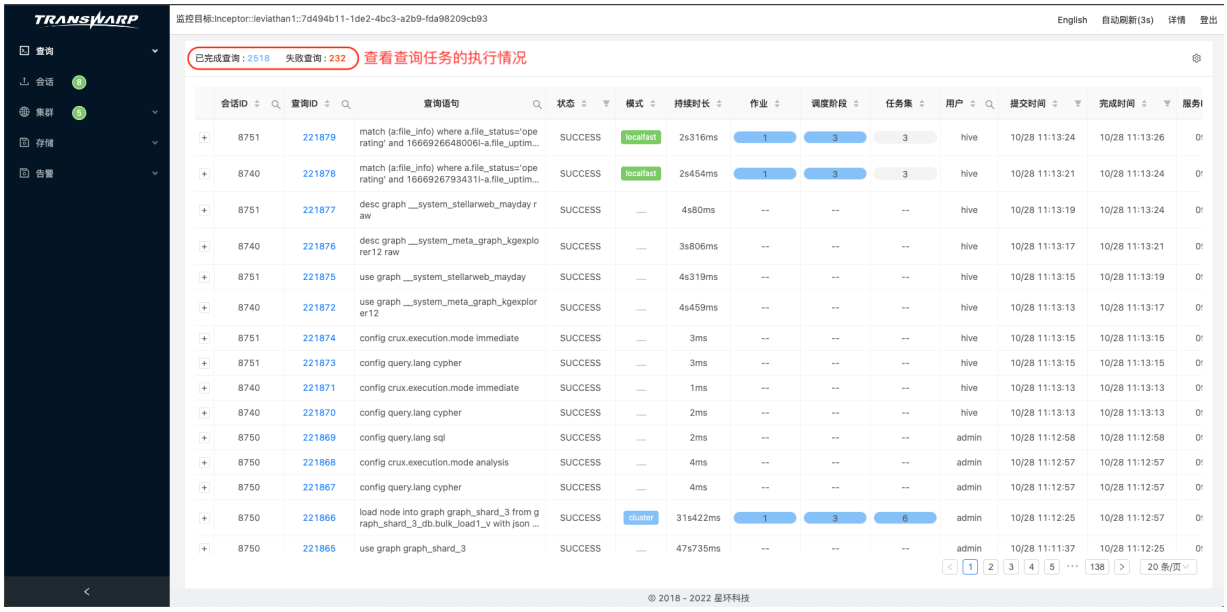


若DBAService配置了单点登录会自动跳转Federation登录页面，按如图方式登录：

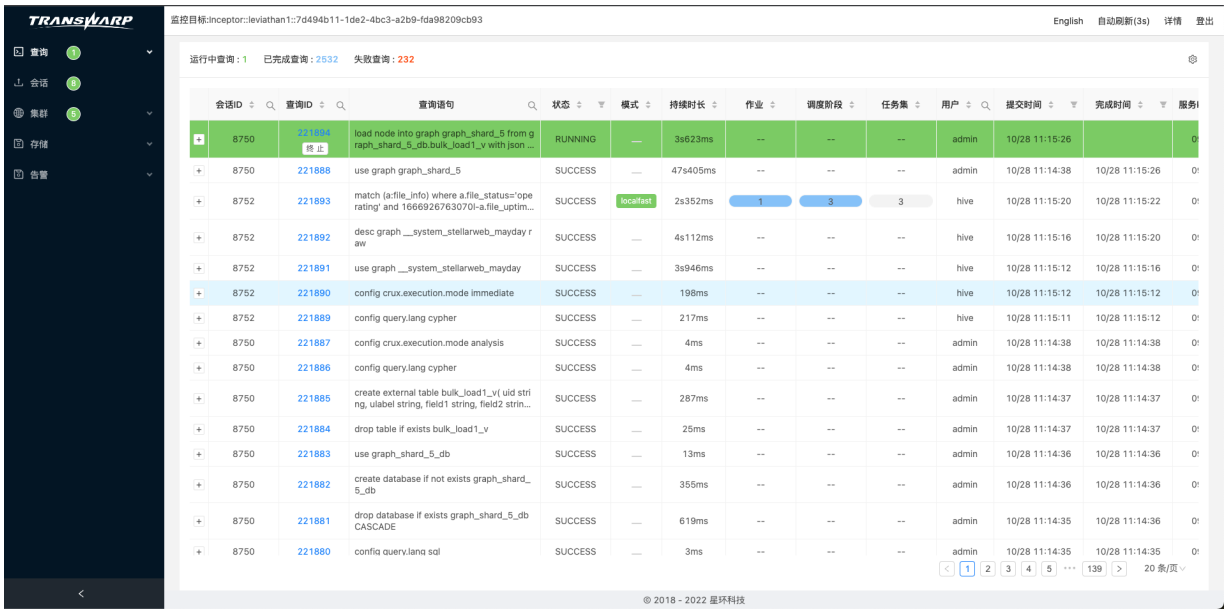


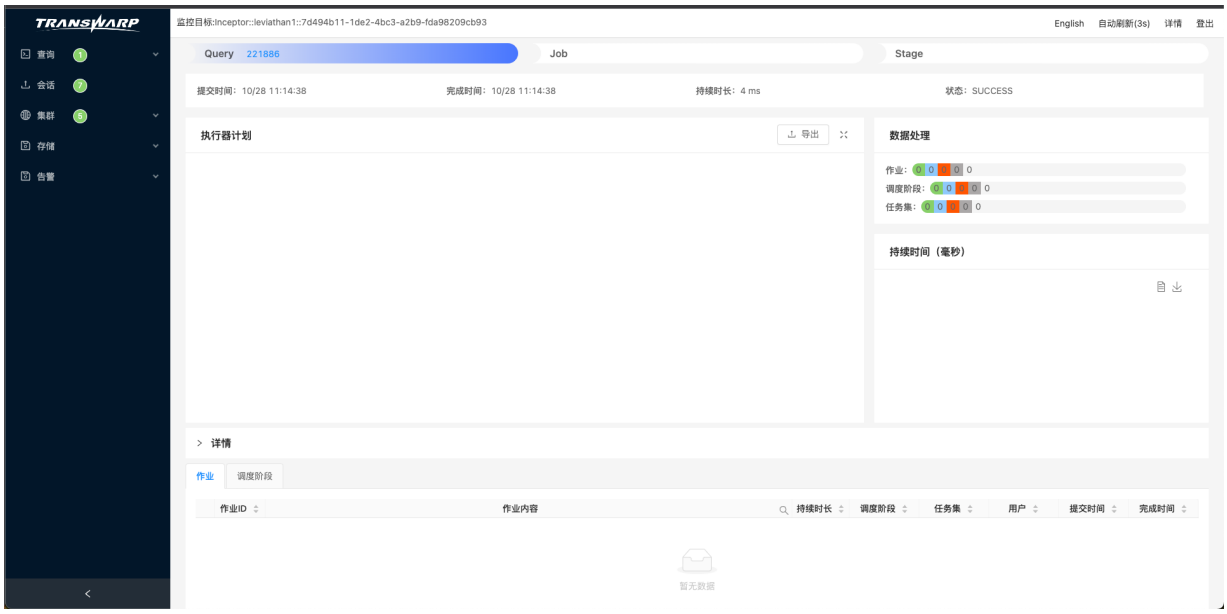
2. 点击左侧 **查询** 菜单，再点击 **查询**（Query） 按钮可以查看当前任务总览情况。





3. 点击一条任务的id可以查看该查询的详细信息。


TRANSWARP
监控目标:Inceptor::leviathan1::7d494b11-1de2-4bc3-a2b9-fda98209cb93
English
自动刷新(3s)
详情
登出
查询
会话
集群
存储
告警
运行中查询：1
已完成查询：2532
失败查询：232
会话ID
查询ID
查询语句
状态
模式
持续时长
作业
调度阶段
任务集
用户
提交时间
完成时间
8750 221894 终止 load node into graph graph_shard_5 from graph_shard_5_db.bulk_load1_v with json ... RUNNING 3s623ms admin 10/28 11:15:26
8750 221888 use graph graph_shard_5 SUCCESS 47s405ms admin 10/28 11:14:38 10/28 11:15:26
8752 221893 match (a:file_info) where a.file_status='operating' and 1666926763070l-a.file_uptim... SUCCESS localfast 2s352ms 1 3 3 hive 10/28 11:15:20 10/28 11:15:22
8752 221892 desc graph __system_stellarweb_mayday raw SUCCESS 4s112ms hive 10/28 11:15:16 10/28 11:15:20
8752 221891 use graph __system_stellarweb_mayday SUCCESS 3s946ms hive 10/28 11:15:12 10/28 11:15:16
8752 221890 config crux.execution.mode immediate SUCCESS 198ms hive 10/28 11:15:12 10/28 11:15:12
8752 221889 config query.lang cypher SUCCESS 217ms hive 10/28 11:15:11 10/28 11:15:12
8750 221887 config crux.execution.mode analysis SUCCESS 4ms admin 10/28 11:14:38 10/28 11:14:38
8750 221886 config query.lang cypher SUCCESS 4ms admin 10/28 11:14:38 10/28 11:14:38
8750 221885 create external table bulk_load1_v(uid string, ulabel string, field1 string, field2 strin... SUCCESS 287ms admin 10/28 11:14:37 10/28 11:14:37
8750 221884 drop table if exists bulk_load1_v SUCCESS 25ms admin 10/28 11:14:37 10/28 11:14:37
8750 221883 use graph_shard_5_db SUCCESS 13ms admin 10/28 11:14:36 10/28 11:14:36
8750 221882 create database if not exists graph_shard_5_db SUCCESS 355ms admin 10/28 11:14:36 10/28 11:14:36
8750 221881 drop database if exists graph_shard_5_db CASCADE SUCCESS 619ms admin 10/28 11:14:35 10/28 11:14:36
8750 221880 config query.lang sql SUCCESS 3ms admin 10/28 11:14:35 10/28 11:14:35
20 条/页
© 2018 - 2022 星环科技


TRANSWARP
监控目标:Inceptor::leviathan1::7d494b11-1de2-4bc3-a2b9-fda98209cb93
English
自动刷新(3s)
详情
登出
查询
会话
集群
存储
告警
Query 221886
Job
Stage
提交时间：10/28 11:14:38
完成时间：10/28 11:14:38
持续时长：4 ms
状态：SUCCESS
执行器计划
导出
数据处理
作业：
调度阶段：
任务集：
持续时间（毫秒）
详情
作业
调度阶段
作业ID
作业内容
持续时长
调度阶段
任务集
用户
提交时间
完成时间
暂无数据

5. Transwarp Extended openCypher

5.1. TEoC 基础

Transwarp Extended openCypher（简称TEoC）是一种基于Cypher语言的开源版本openCypher进行高度支持，并且拓展了openCypher的语法，从而TEoC可以支持更多的功能且允许高效地查询更新图数据。TEoC和SQL有着类似的语法和相同的灵活性，有助于用户完成复杂场景的图查询。在本文档中，将使用TEoC指代部分Cypher或者openCypher。

5.1.1. 图数据简介

TEoC面向的数据是一种描述点/边的图，图中的点和边都可以拥有多个属性值。

- **类型（Label）** 属性图中可以存在多种类型的点和边，例如社交网络中的人、公司、学校等实体，分别拥有不同属性。通过 **label** 来区分不同类型的实体。
- **点（Node）**

  图中的每一个点拥有全局唯一的ID、唯一的类型（label）、多个标签（Tags），以及一组属性K-V键值。一个点通常对应了模型中的一个实体（Entity）。

  需要注意的是，StellarDB中的label与Cypher的label语义不同，而StellarDB中的Tag与Cypher的label语义相同。Tag是一种特殊的属性，由于StellarDB中的点的label是唯一的，点的多重类型可以用Tag来描述。

- **边（Relationship）**

  图中每一个边都是有向边，拥有全局唯一的ID。如果起点和终点间存在多个边，用户需要提供一个额外ID来保证该边的唯一性。每个边有唯一的类型（label），可以有多个标签（Tags），以及一组属性K-V键值。

5.1.2. 存储模式

在上一小节提到的点或者边都拥有全局唯一ID、唯一类型label、多标签Tags以及实体Entity等概念。因此，StellarDB在存储模式上内置了一些系统字段和实体id的计算规则。

- 唯一ID的系统字段为__uid，类型为String，插入数据时需要显式指定。
- 点实体ID的entityKey由点的__uid和label自动计算而来，类型为binary。因此不需要显式指定entityKey，但__uid和label必须显式指定。entityKey在存储层也视为rowKey，来标识数据的唯一性。
- 边实体ID的entityKey由边的__uid和Label以及起点和终点的entityKey计算而来，类型为binary。因此，在创建边数据时，起点entityKey通过显示式地设置系统字段__usid和__uslabels来自动计算得出。终点entityKey通过显示式地设置系统字段__udid和__udlabels来自动计算得出。
- 点或边实体的标签系统字段为__Tags，类型为Array<T>，属于可选项。

点的数据样例：

```
{"entityKey":[49,0,0],"properties":{"__tags":[],"__uid":"1"},"labels":["v"]}
```

边的数据样例：

```
{"startKey":[49,0,0],"endKey":[49,49,56,48,53,50,0,0],"entityKey":[49,0,0,49,49,56,48,53,50,0,0,49,4
9,49,56,48,53,50,1,0,3,0,0,0,8,0,0,0,7,0,0,0],"properties":{"__tags":[],"__uid":"1118052"},"labels":
["e"]}
```

5.1.3. 数据模式

模式（Patterns）作为Cypher语言的核心，可以直接对数据进行描述。它在TEoC中同样也不可或缺，本节将介绍模式在TEoC中的应用。

模式可在CREATE，MATCH等关键词的子句中使用。它有以下几种经典的使用方式：

- **点模式（Patterns for nodes）**

  一个点通常用一个字符变量和一对圆括号进行描述，如下所示：

  ```
  (a)
  ```

  上面的模式描述是一个名称为字符变量a的点。

- **相关点模式（Patterns for related nodes）**

  模式还可以通过箭头来表示多个点以及它们之间的边，如下所示：

  ```
  (a)- ->(b)
  ```

  此模式描述了点a和点b，以及从点a到点b的单向边。

  还可以从两个点扩展到任意数量的点，例如：

  ```
  (a)- ->(b)<- -(c)
  ```

  上面多个点和它们之间的边被统一称作路径(Path)。一条路径同样也可以用一个字符变量来进行描述，如下所示：

  ```
  p=(a)- ->(b)<- -(c)
  ```

  箭头用于描述点之间的方向，也可以选择将其省略。

  ```
  (a)- -(b)
  ```

- **边模式（Patterns for relationships）**

  与点类似，边也可以用字符变量进行描述，不同的是边需要用方括号进行表示：

  ```
  (a)-[f]->(b)
  ```

- **类型模式（Patterns for labels）**

  模式还可以用来描述label，其中点的label可以用以下方式进行描述：

  ```
  (a:Human)- ->(b)
  ```

  也可以用同样的方式描述边的label：

  ```
  (a)-[f:REL_LABEL]->(b)
  ```

  需要注意的是，命名只有在一个TEoC查询的其他地方需要再次引用该点或者边时才必须指定，如果没有必要，也可以进行省略，如下所示：

  ```
  ()-[:REL_LABEL]->()
  ```

- **指定属性（Specifying properties）**

  点和边是图的两个基本结构，它们具有各自的属性。可以用大括号来表示这些属性，如：

  ```
  (a{name:'Jude',gender:'female'})
  ```

  大括号中用逗号分割的两个表达式分别表示点a两个不同的属性name和gender。

  边的属性同样也可以用大括号来表示:

  ```
  (a)-[{friend: false}]->(b)
  ```

- **变长模式匹配（Variable-length pattern matching）**

  可以通过在边中指定长度来描述多个边以及对应的点，例如：

  ```
  (a)-[*2]->(b)
  ```

  上述模式描述了一个长度为2的路径中三个点和两个边的图，它等价于：

  ```
  (a)-->()-->(b)
  ```

  也可以指定长度的范围，如：

  ```
  (a)-[*1..3]->(b)
  ```

  上述模式的最小路径长度为1，最大路径长度为3。它描述了一个从点a出发，到点b的所有长度为1、2和3的路径，该路径也可以被更具体地描述为包含了2个点和1个边，3个点和2个边以及4个点和3个边的图。要描述长度为3或更小的路径，可使用：

  ```
  (a)-[*..3]->(b)
  ```

  因为最小路径长度默认为1，所以上面的模式本质上与最小路径长度为1，最大路径长度为3的模式等价。

5.2. TEoC 前置参数

- 通过beeline或JDBC时，设置参数 `config query.lang cypher;` 将查询语言切换为TEoC模式。
- 根据使用场景选择查询模式（默认为 **immediate** 模式）
  1. **immediate** 模式通常用于并发及短查询场景，查询结果和中间结果通常不超过百万。通过 `config crux.execution.mode immediate;` 切换。
  2. **analysis** 模式通常用于分析场景，创建图、插入数据以及图算法相关的语句必须在该模式下进行。通过 `config crux.execution.mode analysis;` 切换。

5.3. 创建图

5.3.1. 使用TEoC创建图

```
create graph (if not exists) <graph_name> with schema [node_schema] | [rel_schema]
graphproperties:{<k1:v1, ..., kN:vN>};
```

node_schema

```
(:<label_name> {[field1_description],[field2_description],...})
```

rel_schema

```
[:<label_name> {[field_description],[field2_description],...}]
```

field_description

```
fieldName fieldType [index:DEFAULT]
```

创建图语法解释如下：

1. **node_schema** 和 **rel_schema** 都由一个label_name和多个field_description组成。每个field_description包含字段名、字段属性以及是否需要建立索引。
2. 至少指定一个 **node_schema** 。
3. StellarDB 5.0.0 版本将**uid和tags**作为内置属性。其中uid默认占用，tags不支持在schema中修改，__tags只有在有值存在的时候会显示在match语句的结果中，且显示顺序为正序。
4. **index:DEFAULT** 是一个可选项，用于为field添加索引。
5. **graphproperties** 中可以设置的属性有：
   a. `graph.shard.number` 用于指定图的分区数，**必须设置**。分片数的选择对于图数据库许多操作的性能有影响。一般而言，图预期存储的数据量越大，分片数应该越多。下面是推荐方案（仅适用普通场景）：
      i. (点数+边数)小于等于1百万，推荐 `graph.shard.number` 的值为1
      ii. (点数+边数)小于等于1千万，推荐 `graph.shard.number` 的值为集群中节点数量
      iii. (点数+边数)小于等于1亿，推荐 `graph.shard.number` 的值为集群中磁盘总数
      iv. (点数+边数)约等于N亿，推荐 `graph.shard.number` 的值为max(N,集群中磁盘总数)
   b. `graph.replication.number` 可以指定图的数据副本数量。数据副本占用的总存储空间与此值成正比。此值须设置为不小于1，不大于StellarDB Worker数目的整数；为了保证数据库可用性和数据安全，此值应当不小于3。
   c. `graph.encryption.type` 可以指定图数据是否需要静态加密。若设为"NO_ENCRYPTION"（默认值），则不加密；若设为"SM4_CTR"，则使用国标SM4分组加密。启用加密会导致数据库读写性能下降。
   d. `graph.index.type` 可以指定索引的类型，目前只支持native模式，即使用StellarDB内置的索引，后续会加入其他索引类型。
6. **node_schema** 和 **rel_schema** 的label_name必须彼此不重合，且不能为空字符串。

7. **if not exists** 关键字是一个可选项。表示当图不存在时执行创建图逻辑，否则不做任何操作。

以下为语句详例：

```
# 创建分区数为3，副本数为1的图abc，定义A类型的点，包含一个string类型的name属性
create graph abc with schema (:A {name string})
graphproperties:{`graph.shard.number`:3,`graph.replication.number`:1};

# 定义A类型的点，包含一个string类型的name属性,并指定label和属性的comment信息
create graph abc with schema (:A comment "A_info" {name string comment "name_info"})
graphproperties:{`graph.shard.number`:3,`graph.replication.number`:1};

# 创建分区数为3，副本数为3的图abc，定义类型A、B两种类型点，定义C类型边。B类型中的age属性建立索引
create graph abc with schema (:A {name string}) (:B {age int index:DEFAULT, city string}) [:C
{salary double}] graphproperties:{`graph.shard.number`:3};

# 创建分区数为3的图abc，定义A类型的点，包含一个decimal类型的name属性（不指定精度默认为decimal(10,2)
）和一个精度为(38，10)的decimal类型的num属性
create graph abc with schema (:A {name decimal, num decimal(38,10)})
graphproperties:{`graph.shard.number`:3};

# 创建分区数为3的图abc，定义A类型的点，包含一个array数组类型的string_array属性（元素为string类型）和一个
array数组类型的time_array属性（元素为localdatetime类型）
create graph abc with schema (:A {string_list array <string>, time_list array<localdatetime>})
graphproperties:{`graph.shard.number`:3};

# 创建分区数为3，副本数为3的图my_graph，定义Boy和Girl类型的点，均包含string类型的name属性，double类型的
salary属性，int类型的age属性，boolean类型的single属性，localdatetime类型的birthday属性，long类型的reserve属性
，精度为(38,10)的decimal类型的rate属性和array类型的hobby属性（元素类型为string类型），定义Friend和Likes两种类型
的边，均包含int类型的since属性
create graph my_graph with schema (:Boy {name string, salary double, age int, single boolean,
birthday localdatetime,
reserve long, rate decimal(38,10),hobbys array<string>}) (:Girl {name string, salary double, age
int, single boolean,
birthday localdatetime, reserve long, rate decimal(38,10),hobbys array<string>}) [:Friend {since
int}] [:Likes {since int}]
graphproperties:{`graph.shard.number`:3,`graph.replication.number`:3};
```

5.3.2. 使用JSON创建图

- 使用JSON文件创建图

```
create graph <graph_name> with file schema 'hdfs:///etc/conf/example.json';
```

以下是example.json模板：

```
{
  "graph.name":"my_graph",
  "graph.shard.number":"3",
  "graph.replication.number":3,
  "graph.encryption.type":"NO_ENCRYPTION",
  "vertex.tables":
  [
    {
      "label.value":"Boy",
      "field.schemas":
      [
        {
          "field.type":"string",
          "field.name":"name"
        },
        {
          "field.type":"double",
          "field.name":"salary"
        },
        {
          "field.type":"int",
          "field.name":"age"
        },
        {
          "field.type":"boolean",
          "field.name":"single"
        },
        {
          "field.type":"local_datetime",
```

```
          "field.name":"birthday"
        },
        {
          "field.type":"long",
          "field.name":"reserve"
        },
        {
          "field.extra.attr": {"field.type": "STRING"},
          "field.name": "hobbys",
          "field.type": "ARRAY"
        },
        {
          "field.type": "decimal",
          "field.name": "rate",
          "field.extra.attr": {"precision": 10,"scale": 2}
        },
        {
          "field.type":"geo",
          "field.name":"geoPoint"
        },
        {
          "field.type":"time_series",
          "field.name":"ts",
          "field.extra.attr":{"real.type":"int","size.limit":"10"}
        }
      ],
      "index.schemas": [salary, age]
    },
    {
      "label.value":"Girl",
      "field.schemas":
        [
        {
          "field.type":"string",
          "field.name":"name"
        },
        {
          "field.type":"double",
          "field.name":"salary"
        },
        {
          "field.type":"int",
          "field.name":"age"
        },
        {
          "field.type":"boolean",
          "field.name":"single"
        },
        {
          "field.type":"localdatetime",
          "field.name":"birthday"
        },
        {
          "field.type":"long",
          "field.name":"reserve"
        },
        {
          "field.extra.attr": {"field.type": "STRING"},
          "field.name": "hobbys",
          "field.type": "ARRAY"
        },
        {
          "field.type": "decimal",
          "field.name": "rate",
          "field.extra.attr": {"precision": 10,"scale": 2}
        },
        {
          "field.type":"geo",
          "field.name":"geoPoint"
        }
      ],
      "index.schemas": []
    }
  ],
  "edge.tables":
  [
    {
      "label.value":"FRIEND",
      "field.schemas":
      [
        {
          "field.type":"int",
          "field.name":"since"
        }
      ],
      "index.schemas":[]
    },
```

```
      {
        "label.comment": "LIKES_info",
        "label.value":"LIKES",
        "field.schemas":
        [
          {
            "field.comment": "since_info",
            "field.type":"int",
            "field.name":"since"
          }
        ],
        "index.schemas":["__EXTRAID"]
      }
    ]
  }
```

- 使用JSON字符串创建图

```
create graph (if not exists) my_graph with json schema '{...}';
```

其中，'{…}'中为example.json中的JSON字符串。

- 在使用JSON文件或字符串创建图时，有以下几点需要注意：
  1. __EXTRAID字段用于标识两点间多条边的唯一性。如果希望快速过滤该字段的话，需要添加__EXTRAID字段的索引；否则不需要添加。
  2. "vertex.tables" 与 "edge.tables" 中的 "label.value" 必须彼此不重合，且不可为空字符串。
  3. "label.comment" 与 "field.comment" 分别为label和属性的comment注释信息，可以不指定。
  4. ARRAY类型的内部元素(field.extra.attr)支持String，Boolean，Integer，Long，Double，Localdatetime和Decimal类型。
  5. 对于建图时不含时区的时间戳类型，StellarDB 3.0.X版本的拼写为"local_datetime"，StellarDB 4.0修改为"localdatetime"，并兼容3.0.X的拼写方式("local_datetime")，推荐使用"localdatetime"。

5.3.3. 使用已有图的schema创建新图

语句	说明
create graph copy_graph with schema from graph my_graph;	创建图copy_graph与已有的图my_graph有相同的schema
create graph if not exists copy_graph with schema from graph my_graph;	当图copy_graph不存在时，创建与已有的图my_graph有相同的schema
create graph if not exists copy_graph with schema from graph my_graph graphproperties:{`graph.shard.number`:3,`graph.replication.number`:3,`graph.encryption.type`:'NO_ENCRYPTION'};	创建与已有的图相同schema图的同时可以指定graphproperties

5.3.4. 查看已创建的图

语句	说明
show stellargraph(s);	展示stellardb中的抽象图，括号中的s可加可不加。
show graph(s);	括号中的s可加可不加。

5.3.5. 查看图schema信息

语句	说明
desc graph bank;	展示图bank的schema（**推荐Beeline调用**）
desc graph bank all;	展示图bank的cypher建图语句（**推荐Beeline调用**）
desc graph bank node alter/delete;	展示图bank中创建/删除点的cypher语句
desc graph bank relationship alter/delete;	展示图bank中创建/删除边的cypher语句
desc graph bank label "${label name}" alter/delete;	展示图bank中指定label的添加/删除语句
desc graph bank raw;	展示图bank的schema，返回一个json字符串（**推荐JDBC调用**）
desc graph bank meta;	展示图bank的schema中一些元信息
show labels [NODE|EDGE];	显示图中包含的label信息，执行前需要 use graph <graph_name>; 指定图名
desc label <labelName>;	显示指定label的具体信息，执行前需要 use graph <graph_name>; 指定图名

5.3.6. 选择/切换图

```
use graph abc;
```

其中abc为所选图名，选择/切换到图abc，成功后可以对该图进行增、删、改、查等操作。

5.4. 删除图

输入并且执行下列语句删除图my_graph。

```
drop graph my_graph;
```

在当前版本中执行上述图删除语句后，被删除的图会被默认放到回收站。回收站的具体使用方式详见《回收站功能》章节。

5.5. 批量数据导入

图数据库支持通过TEoC，将关系型数据批量导入图中，本手册中有时使用bulkload表示。

当前支持的格式有：**Text**、**ORC**、**CSV** 以及 **Parquet**。批量数据导入时需要在 **analysis** 模式下进行。切换查询模式可以通过在命令行模式执行 `config crux.execution.mode analysis;` 或在KG Explorer切换**分析模式** 实现。

5.5.1. 批量数据导数约束（新增）

StellarDB 5.0.0版本增加了批量数据导入的约束，此约束要求在使用bulkload进行点边数据批量导入的过程中，必须先批量导入点数据，然后再批量导入边数据。在边数据批量导入的过程中，图数据库还会对需要导入的边数据进行起始点和终点的交验，对于不存在起点终点的边不再进行导入。

5.5.2. 准备数据

StellarDB中的点和边数据需要从关系型数据库中的对应表中分别导入。每个label对应一张关系型数据库中的表。导入过程首先创建source库，并在source库下创建点表v和边表e，包含点数据和边数据，v表和e表的信息分别如下图所示：

- v表结构如下所示，其中uid对应节点的__uid，ulabel对应节点的label，str代表该节点具有的一个属性。

```
+----------+-----------+---------------+----------+--------+---------+
| col_name | data_type | default_value | not_null | unique | comment |
+----------+-----------+---------------+----------+--------+---------+
| uid      | string    | NULL          | No       | No     | NULL    |
| ulabel   | string    | NULL          | No       | No     | NULL    |
| str      | string    | NULL          | No       | No     | NULL    |
+----------+-----------+---------------+----------+--------+---------+
```

- e表结构如下所示，其中usid对应边的起始节点的__uid，uslabel对应边的起始节点的label，udid对应边的终点节点的__uid，udlabel对应边的终点节点的label，uelabel对应边的label，str代表边的一个属性。

```
+----------+-----------+---------------+----------+--------+---------+
| col_name | data_type | default_value | not_null | unique | comment |
+----------+-----------+---------------+----------+--------+---------+
| usid     | string    | NULL          | No       | No     | NULL    |
| uslabel  | string    | NULL          | No       | No     | NULL    |
| udid     | string    | NULL          | No       | No     | NULL    |
| udlabel  | string    | NULL          | No       | No     | NULL    |
| uelabel  | string    | NULL          | No       | No     | NULL    |
| str      | string    | NULL          | No       | No     | NULL    |
+----------+-----------+---------------+----------+--------+---------+
```

5.5.3. 数据导入

5.5.3.1. 使用bulk upsert/create/delete导入数据

通过bulk upsert语句批量导数(推荐使用)

StellarDB 5.0.0 版本更新了bulk upsert语法，使用 `select … bulk upsert` 语法代替了 `select … upsert`语法。

例句	说明
select str, name from source_db.v bulk upsert (:XX{__uid: uid, prop1: str, prop2: name});	导入点。从source_db库的v表中导入label为XX的点。语句中点的__uid来源于v表的uid列，属性prop1和prop2分别来自表的str列和name列。
select * from source_db.e bulk upsert [:YY{__usid: usid, __udid: udid, __uslabels:['XX'], __udlabels: ['XX']}];	导入边。从source_db库的e表中导入label为YY的边。__uslabels和__udlabels分别指定起点和终点的label，__usid和__udid分别指定起点和终点的uid。__usid，__udid，__uslabels，__udlabels必须指定。
select * from source_db.e bulk upsert [:YY{__uid: uid,__usid: usid, __udid: udid, __uslabels:['XX'], __udlabels: ['XX']}];	导入边。从source_db库的e表中导入label为YY的边。__uid用于区分相同起点终点的平行边。本例通过e表中的uid列来提供__uid值。
select * from source_db.e bulk upsert [:YY{__uid: tostring(cluster_unique_id()),__usid: usid, __udid: udid, __uslabels:['XX'], __udlabels: ['XX']}];	导入边。从source_db库的e表中导入label为YY的边。__uid用于区分相同起点终点的平行边。本例由系统生成__uid值。
select usid, uslabel, udid, udlabel, uelabel, ueid from source_db.e bulk upsert [::uelabel {__usid: tostring(usid), __udid: tostring(udid), __uslabels: [uslabel], __udlabels: [udlabel], __uid:ueid}];	这里边的label为source库e表的uelabel列，边的属性__usid，__udid，__uslabels，__udlabels和str分别为e表的usid，udid，uslabel，udlabel列,支持tostring()、tointeger()等数据类型转换函数
select usid, udid from source_db.e where usid = '5' bulk upsert [:edge {__usid: usid, __udid: udid, __uslabels: ['XX'], __udlabels: ['XX']}];	选择source_db的e表中usid，udid两列，过滤出usid='5'的数据，并且指定边的值为edge,__uslabels和__udlabels的值为XX, __usid，__udid分别为e表的usid列和udid列
select * from source_db.v bulk upsert (:XX{__uid: uid, ts: to_timeseries(ts, "string")});	假设source_db库的v表中有uid，ts两列，创建label为XX的点包含uid，ts两种属性。点的__uid来源于v表的uid列，点的ts属性类型为timeseries<string>，且来源于v表的ts列

表 1. 通过bulk create语句批量导数

例句	说明
select * from source_db.v bulk create (:XX{*, __uid: uid, name: str});	从source_db库的v表中读出数据并以bulk create的方式插入数据，和bulk upsert不同之处见下文
select uid from source_db.e bulk create [:YY{__uid: uid,__usid: uid, __udid: uid, __uslabels:['XX'], __udlabels: ['XX']}];	
select usid, uslabel, udid, udlabel, uelabel, ueid from source_db.e bulk create [::uelabel {__usid: usid, __udid: udid, __uslabels: [uslabel], __udlabels: [udlabel], __uid:eid}];	

通过bulk delete语句批量删数

使用bulk delete删除时，请先删除边再删除点。

例句	说明
select uid from source.v bulk delete (:xx{__uid:uid});	从v表中读出数据并以bulk delete的方式删除点数据

`select usid, udid from source.e bulk delete [:XX{__uid: "", __usid: usid, __udid: udid, __uslabels:['XX'], __udlabels:['KK']}];`	从e表中读出数据并以bulk delete的方式删除边数据
`select edge_label, uid, usid, udid from source.e bulk delete [::edge_label{ __uid: uid, __usid: usid, __udid: udid, __uslabels:['XX'], __udlabels:['KK']}];`	从e表中读出edge_label列作为边label值，usid列作为起点uid值，udid列作为终点uid值，uid列作为边extraId值

- 起点终点的和边label都相同的边数据，导入时视作同一条边，后导入的数据会覆盖之前的数据。若需要区分两点之间多条相同label的边，需配置生成唯一的边extraid。bulk create或bulk upsert语句中通过__uid指定，load语句的方式通过ueid指定。见下文load.json示例。
- 导入时可能出现导入结果计数和source_db库中count计数不一致，可能原因有：1.存在错误数据，可查看导入结果的hdfs错误结果路径error_path；2.如果没有错误路径，则可能时存在重复数据。

5.5.3.2. 使用load语句进行批量数据导入（deprecated）

StellarDB 5.0.0版本不再对load语法进行支持，批量数据导入请参照《使用bulk upsert进行数据批量导入》小节进行相应修改。

5.5.3.3. 数据导入方法对比

1. bulk upsert语句的实际含义为对已有数据进行更新，对新数据进行插入操作，这里推荐使用bulk upsert进行导数。
2. bulk create语句指定批量导数的方式为插入，即新插入的数据会覆盖所有原数据。

5.5.4. 数据类型对应关系

使用StellarDB进行批量数据导入时，数据源为Quark中建立的表。下表中罗列了图数据中的属性的数据类型与表中字段的数据类型的对应关系：

StellarDB图中数据类型	Quark表的数据类型
boolean	boolean
int	int
long	int,bigint
double	int,long,double
string	string,char,varchar
localdatetime	timestamp
array	array
decimal	decimal
geo	[double,double]
timeseries<type>	map<timestamp, type>

上述数据类型当中，有以下几种数据类型需要特别解释：

1. localdatetime与timestamp：
   a. 原始数据必须严格符合格式：yyyy-MM-dd HH:mm:ss.SSS，否则Quark会识别为NULL。
   b. 导入数据库后查询出来的localdatetime的格式为：yyyy-MM-ddTHH:mm:ss.SSS。
   c. 原始Quark表q1包含字段d1:string/d1:date/d1:datetime想在StellarDB中存储为localdatetime类型，创建新表q2包含同名字段d1:timestamp且其余字段与q1保持一致。使用如下语句插入数据到q2表，再从q2导入StellarDB，其中TDH_TODATE(<date>[,<originalformat>,<targetformat>])，date为字段名，originalformat为原始格式，targetformat是转换后的格式，targetformat必须为“yyyy-MM-dd HH:mm:ss.SSS”。

```
insert into table q2 select xx1,tdh_todate(d1, "yyyy-MM-dd'T'HH:mm:ss.SSSZ", 'yyyy-MM-dd
HH:mm:ss.SSS'),from p1;
```

2. 关于地理空间数据（GEO）：
   a. 原始数据为长度为2的array[double]类型：[longitude,latitude]，其中第一位为经度，后一位为纬度。如：[20.5,30.5].
   b. 导入数据库后查询出来的GEO的格式为：[20.5, 30.5]。
3. 关于时序类型数据（timeseries）：
   a. 原始数据为map类型，其中key为timestamp类型，value为对应时序类型的原始类型。
   b. 导入时需要使用 **to_timeseries()** 函数将原始数据转换成对应类型，参考bulk upsert的最后一个示例。

5.5.5. 从分区表导入数据的注意事项

StellarDB支持使用分区表进行数据导入。但是在使用分区表进行数据批量导入的过程中，如果分区类型是string类型，需要注意在数据导入时进行强制类型转换，否则会导致数据导入出现错误。以下通过一个完整的例子对该注意事项进行说明：

1. 假设已经通过如下指令创建分区表，并且向分区表中插入过数据

```
# 创建分区表taa，通过字符串类型
CREATE TABLE taa (a string,b INT) partitioned by(c string) row format delimited fields
terminated by '\t';

# 分区表中插入数据
insert into table taa partition(c='cc')
select t.name, t.age
from xxx.ta t;
```

2. 使用 select … bulk upsert 语法进行数据导入

```
select a,b from xxx.taa where tostring(c)='cc' bulk upsert (:Boy{__uid:a});
```

这里在数据导入的where子句中使用tostring()函数进行强制类型转换。

5.5.6. 数据校验

TEoC新增了bulkload快速校验数据的功能。在打开数据校验开关，并且设置非法数据上限的前提下，如果bulkload过程中遇到的非法数据达到上限时，bulkload过程立刻退出，而不是等到bulkload结束退出。使用方式为：

1. 在客户端中设置参数开启bulkload数据校验：

```
set crux.bulkload.validate.enabled = true;
```

2. 设置非法数据上限，非法数据上限的数量通过以下两个参数的乘积确定。在下述例子中，非法数据上线数量为1000000 * 0.01 = 10000。

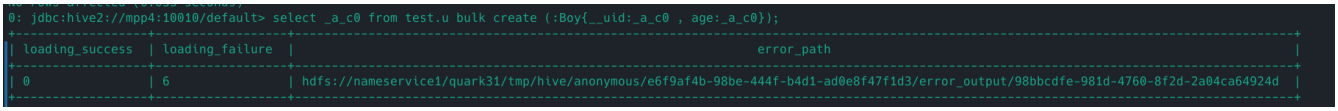
```
set crux.bulkload.partition.default.size = 1000000;
set crux.bulkload.failed.row.ratio = 0.01;
```

5.5.6.1. 错误数据输出

当bulkload引擎遇到错误数据时，除了将错误数据输出到日志中，也会把错误数据输出到HDFS目录。该机制默认不启用，如需启用，需设置如下参数：

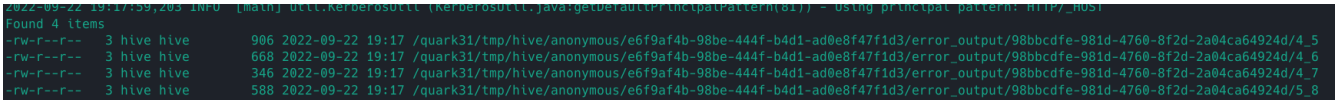
```
set crux.stellardb.bulkload.err.output.enabled = true;
```

错误数据输出至的HDFS目录会在bulkload结束时通过语句显示，如下图第三列所示：



5.5.6.2. HDFS错误数据目录结构

HDFS中错误数据目录位于Inceptor session目录下的 **error_output** 子目录。一条bulkload语句对于该子目录下的一个独立uuid的新目录。目录内文件名按${stage}_${task_id}命名，如下图所示：



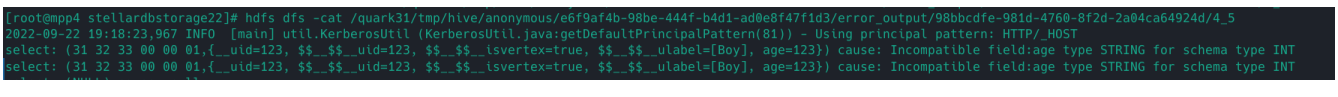
文件内格式按：

```
select: ${错误行数据} ${错误原因}
# 说明错误数据在select operator被发现并处理
```

或

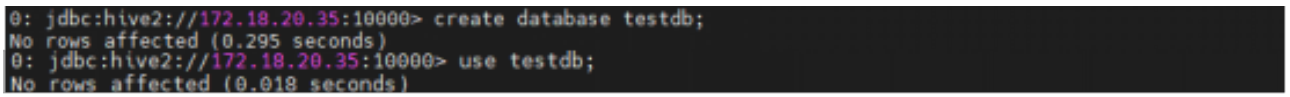
```
bulkload: ${错误行数据} ${错误原因}
# 说明数据在写入{stellardb}存储时被发现并处理
```

如下图所示：



5.5.7. 应用实例

1. 登录beeline客户端，如下图所示创建导入数据用的数据库。



2. 创建数据源表。

```
0: jdbc:hive2://172.18.20.35:10000> create external table ssca_source(type string, src string,dst string,weight int)
. . . . . . . . . . . . . . . . . . > row format delimited fields terminated by " "
. . . . . . . . . . . . . . . . . . > location 'hdfs://nameservice1/tmp/ssca2/data';
No rows affected (0.543 seconds)
```

3. 创建点表和边表。

```
0: jdbc:hive2://172.18.20.35:10000> create  table ssca2 as select * from ssca_source where type='a';
+-------+------+------+---------+
| type  | src  | dst  | weight  |
+-------+------+------+---------+
+-------+------+------+---------+
No rows selected (2.027 seconds)
0: jdbc:hive2://172.18.20.35:10000> create  table ssca2_vertex as select * from (
. . . . . . . . . . . . . . . . . . > select distinct src vertex from ssca2
. . . . . . . . . . . . . . . . . . > union
. . . . . . . . . . . . . . . . . . > select distinct dst vertex from ssca2
. . . . . . . . . . . . . . . . . . > );
+---------+
| vertex  |
+---------+
+---------+
No rows selected (5.528 seconds)
```

边表

点表

4. 使用JSON字符串创建图。

```
0: jdbc:hive2://172.18.20.35:10000> set query.lang=cypher;
No rows affected (0.006 seconds)
0: jdbc:hive2://172.18.20.35:10000> create graph ssca2_graph with json schema
. . . . . . . . . . . . . . . . . . > '{
. . . . . . . . . . . . . . . . . . > "graph.name":  "ssca2_graph",
. . . . . . . . . . . . . . . . . . > "graph.shard.number":  "23",
. . . . . . . . . . . . . . . . . . > "vertex.tables":
. . . . . . . . . . . . . . . . . . > [{
. . . . . . . . . . . . . . . . . . > "label.value":  "v",
. . . . . . . . . . . . . . . . . . >         "field.schemas":
. . . . . . . . . . . . . . . . . . >         [
. . . . . . . . . . . . . . . . . . >         ],
. . . . . . . . . . . . . . . . . . >         "index.schemas":
. . . . . . . . . . . . . . . . . . >         [
. . . . . . . . . . . . . . . . . . >         ]
. . . . . . . . . . . . . . . . . . >     }
. . . . . . . . . . . . . . . . . . > ],
. . . . . . . . . . . . . . . . . . > "edge.tables":
. . . . . . . . . . . . . . . . . . > [{
. . . . . . . . . . . . . . . . . . >         "label.value":  "e",
. . . . . . . . . . . . . . . . . . >         "src.label.value":  "v",
. . . . . . . . . . . . . . . . . . >         "dst.label.value":  "v",
. . . . . . . . . . . . . . . . . . >         "field.schemas":
. . . . . . . . . . . . . . . . . . >         [{
. . . . . . . . . . . . . . . . . . >               "field.type":  "int",
. . . . . . . . . . . . . . . . . . >               "field.name":  "weight"
. . . . . . . . . . . . . . . . . . >             }
. . . . . . . . . . . . . . . . . . >         ],
. . . . . . . . . . . . . . . . . . >         "index.schemas":
. . . . . . . . . . . . . . . . . . >         [
. . . . . . . . . . . . . . . . . . >         ]
. . . . . . . . . . . . . . . . . . >     }
. . . . . . . . . . . . . . . . . . > ]
. . . . . . . . . . . . . . . . . . > }';
+--------+
| _a_c0  |
+--------+
+--------+
No rows selected (7.023 seconds)
```

5. 导入点和边数据。

```
0: jdbc:hive2://172.18.20.35:10000> load node into graph ssca2_graph from testdb.ssca2_vertex with file schema "hdfs:///tmp/test/ssca2_load.json";
+------------------+------------------+
| loading_success  | loading_failure  |
+------------------+------------------+
| 16384            | 0                |
+------------------+------------------+
1 row selected (8.459 seconds)
0: jdbc:hive2://172.18.20.35:10000> load relationship into graph ssca2_graph from testdb.ssca2 with file schema "hdfs:///tmp/test/ssca2_load.json";
+------------------+------------------+
| loading_success  | loading_failure  |
+------------------+------------------+
| 1213774          | 0                |
+------------------+------------------+
1 row selected (11.511 seconds)
```

6. 查看导入失败数据。

```
0: jdbc:hive2://172.18.20.35:10000> load relationship into graph bad_data_graph from bad_dat
a.ssca2 with file schema "hdfs:///tmp/test/ssca2_load.json";
+-------------+---------+
|   _a_c0     | _a_c1   |
+-------------+---------+
| 124884324   | 2       |
+-------------+---------+
1 row selected (152.925 seconds)
```

导入失败数据

导入失败的具体数据可以在 **所有executor节点** 的 **/var/log/quarkX/quark-executor.log** 中通过过滤 **invalid** 字段查找。如果开启了错误数据输出，则可以直接到对应的HDFS目录查看：

```
2021-01-21 17:01:04,419 WARN io.transwarp.inceptor.execution.crux.CruxBulkLoadOperator: Foun
d invalid row in storage write PBulkloadOutputPlan-550:PBulkloadSinkOp-565: (null,null)
java.lang.IllegalArgumentException: Rowkey could not be null, the given data is null | null
```

如图示，此处未导入数据为null。

本案例使用beeline客户端进行数据的批量导入，也可以通过StellarDB的前端组件KG Explorer，使用点选的方式进行批量导入，操作方式详见 《KG Explorer使用文档》→《批量导入工具介绍》章节。

5.5.8. 常见错误

5.5.8.1. 客户端执行异常

通过客户端接入，比如使用JDBC或命令行方式接入，异常一般出现为执行语句拼写、数据类型、元数据解析错误等。

1. __uid的类型integer类型，导致rowKey函数入参校验失败。

```
0: jdbc:hive2://mpp16:10000> select uid from snap.v with tointeger(uid) as uid bulk create
 (:v{__uid:uid});

Error: Found exception during run crux query: Function unify_noderel with I,[LCruxString;,CruxMapType
 cannot be matched signature, the function args should be (string,array[string],graphprops),
 (string,array[string],string,array[string],string,array[string],graphprops), try to use type cast
 func : tostring(), array() or toarray() (state=,code=0)
```

解决方案：

__uid的类型必须是string类型，不可以是integer类型。可以使用tostring函数，将作为__uid的值转为string类型使用。

2. 使用了graph schema中未定义的vertex.table {xx}。

```
0: jdbc:hive2://mpp16:10000/default> select uid from snap.v bulk create (:xx{__uid:uid});
Error: Found exception during run crux query: [Query failed with] Un-defined vertex label[xx]
 (state=,code=0)
```

解决方案：

通过使用 `desc graph xxx` 来查看图名为xxx的图的schema，然后根据实际情况去调整schema定义或者cypher语句。

3. 语句执行结果显示有入库失败数据条数。

```
0: jdbc:hive2://mpp16:10000/default> select * from snap.e with concat_string(suid,duid) as
 uid,suid,duid,attr bulk create [:e {name:'xxxx',__uid:uid,__usid:suid,__uslabels:['v'],
 __udid:duid,__udlabels:['v']}];
+------------------+------------------+---------------------------------+
| loading_success  | loading_failure  |           error_path            |
+------------------+------------------+---------------------------------+
| 0                | 40               | Error output path not enabled.  |
+------------------+------------------+---------------------------------+
```

解决方案：

导致数据入库失败的情况较多，需要结合服务端日志来进一步排查。一般错误来源可能为使用未定义的field、__uid以及label等导致。

5.5.8.2. 服务端日志异常

StellarDB由计算和存储两部分构成，可以优先从计算侧server和executor两个角色的服务日志排查。

1. executor日志显示 `Found invalid row in storage write` 发现无效行数据。

```
2021-11-29 19:18:00,328 WARN io.transwarp.inceptor.execution.crux.CruxBulkLoadOperator: Found
 invalid row in storage write PBulkloadOutputPlan-28:PBulkloadSinkOp-39: (NULL)
java.lang.IllegalArgumentException: bulk load pre ser write binary could not null
```

解决方案：

此时需要检查所有参与rowKey计算的属性值是否为null，比如__uid、label以及非法field等。

2. executor日志显示 `Could not find label in vertex schema`，schema中没有找到点的label。

```
2021-11-30 17:27:32,803 WARN io.transwarp.stargate.CruxStellarDBStoreInfoGetter$GenericUDFNodeRK:
 Could not find label  in vertex schema
2021-11-30 17:27:32,803 INFO io.transwarp.inceptor.execution.crux.NewCruxSelectOperator: Found
 invalid row in op PSelectPlan-1308:PSelectOp-1315: ((31 37 30 35 30 37,{grisk=null, frisk=null,
 __uid=170507, $$__$$__uid=170507, $$__$$__isvertex=true, cid=null, uid=170507, $$__$$__ulabel=[],
  srisk=null}))
```

解决方案：

需要检查bulk create语句中label是否设置或者变量是否为null。

3. executor日志显示 `Found invalid row in op PSelectPlan`。

```
2021-11-30 18:03:05,492 INFO io.transwarp.inceptor.execution.crux.NewCruxSelectOperator: Found
 invalid row in op PSelectPlan-1395:PSelectOp-1402: (31 00 00,{__uid=1, $$__$$__uid=1,
 $$__$$__isvertex=true, name=xxx, $$__$$__ulabel=[v]})
```

解决方案：

看到这个日志信息，需要仔细检查语句中是否使用了在graph schema中未定义的field，通过 `desc graph xxx` 查看。

4. executor日志显示 `return one wrong row data when deserialize`。

```
2021-11-29 21:49:54,572 WARN io.transwarp.stargate.reader.HDFSTableReader: [HDFSReader] return one
 wrong row data when deserialize,now reset to null: java.lang.RuntimeException: Missing
 field/column for a row, error row: 99967
2021-11-29 21:49:54,572 WARN io.transwarp.stargate.reader.HDFSTableReader: [HDFSReader] return one
 wrong row data when deserialize,now reset to null: java.lang.RuntimeException: Missing
 field/column for a row, error row: 9997
```

解决方案：

HDFS reader读取数据异常，行数据对应的字段丢失。检查外部表字段定义和外挂目录下数据的对应关系，字段定义、列分隔符、换行符等，通过 `select * from ext_source return uid limit 10` 进行验证外部表数据是否读取正确。

5.6. 数据类型

5.6.1. 类型及表达式

1. TEoC支持如下基础数据类型：
   a. 布尔类型（boolean）
   b. 整数类型（int）
   c. 长整数类型（long）
   d. 浮点数类型（double）
   e. 字符串类型（string）
   f. 不含时区的时间戳类型（localdatetime）
2. TEoC支持如下复杂类型：
   a. 数组类型array<T>：其中T表示array内部元素的类型，内部元素可以是以上六种基本数据类型。
   b. 精度类型decimal(Precision,Scale)：其中Precision和Scale分别代表精度（字段长度，包括小数点前和小数点后的位）和标度（小数位数）。精度最大不能超过38，标度不能超过精度。不指定精度和标度则默认为 **decimal(10,2)**。如果数据无法按照指定的精度和标度表示，将被认为是NULL值。
   c. 地理坐标点(point(longitude, latitude))：其中longitude和latitude都是浮点数(double)基本数据类型。经度的取值在[-180.0, 180.0]之间，纬度的取值[-90.0, 90.0]之间。在schema建图中用 **geo<double>** 表示。
   d. 时序类型(timeseries<T>(limit))：其中T表示时序类型的原始类型，可以是 **除时间戳外** 的五种基本数据类型。limit表示保留最新的多少个时间点的记录（建议不超过100）。
3. 类型表达式：数据类型的类型表达式，请参照《类型表达式》章节。

5.6.2. 地理坐标类型

5.6.2.1. 类型简介

1. 空间几何即具有以下特征的点：
   a. 每个点会与一个特定的坐标参考系统(CRS)关联，坐标中值的含义由该系统确定。
   b. 可以将Point实例分配给节点和关系属性。
   c. 可以使用空间索引对具有Point属性的节点进行索引。
   d. 距离函数将在所有两个点具有相同的CRS的点上起作用。
2. 坐标参考系统（CRS）：
   a. 支持一种坐标参考系统(CRS)：基于地球上任意点建模的地理坐标系。
   b. 地理坐标类型支持 **WGS-84** ，将经纬度组合成一个数对(x, y)。

5.6.2.2. 使用方法

在StellarDB中可以通过如下方式使用经度和纬度构造一个类型为地理数据类型的数据：

```
point(longitude,latitude)
```

其中，经度（longitude）和纬度（latitude）字段的单位为十进制，需要使用TEoC中浮点数 **double** 类型数据。不能使用其他任何如度、分、秒等格式的数据型。计算 **distance** 的函数的计算单位始终是以 **km(千米)**

作为计量单位。

在大多数情况下，不必明确指定CRS，CRS类型可以从输入数据中进行推断，且推断遵循以下两条规则：

- 如果使用经度和纬度指定坐标，则CRS将被认为是地理坐标系，因此是WGS-84；
- 如果坐标中只有两个维度（x和y或经度和纬度），则CRS将为2D CRS。

下列语句示范如何使用地理坐标类型数据：

```
match (n) where n.point = point(longitude,latitude);
match (m) where m.point = point(longitude,latitude);
return pt_distance(n.point, m.point);
```

该语句用于计算两个坐标点之间的距离，单位为千米。其中，节点n和节点m的point属性是地理坐标类型的，它们的longitude和latitude部分都可以用 **double** 类型的数据替换。如下图所示：

```
match (n) where n.point = point(20.56789,30.56789);
match (m) where m.point = point(20.56891,30.56891);
return pt_distance(n.point, m.point) as distance;
```

返回结果下图所示：

```
+---------------------+
|      distance       |
+---------------------+
| 0.14966836824720137 |
+---------------------+
```

特别注意 **point** 函数调用坐标的顺序，以及这些值在结果中的显示方式，输入并执行下列语句：

```
return point(20.5678,30.7891) as Point;
```

返回结果如下所示：

```
+--------------------+
|       point        |
+--------------------+
| [20.5678,30.7891]  |
+--------------------+
```

5.6.2.3. 查询方法

可以将组件作为实例的属性进行访问：

类型	访问表达式	描述	类型	范围/格式
longitude	longitude(point)	地理CRS坐标的第一个元素，正数为本初子午线以东的度数	double	[-180.0,180.0]，数字文字

latitude	latitude(point)	地理CRS坐标的第二个元素，正数为赤道以北的度数	double	[-90.0,90.0]，数字文字

- 获得经度：

```
match (n) return longitude(n.point);
```

- 获得纬度：

```
match (n) return latitude(n.point);
```

下述语句为获取经纬度语句实例：

```
return longitude(point(20.5678,30.7891)) As longitude, latitude(point(20.5678,30.7891)) As latitude;
```

该语句执行结果如下所示：

```
+-----------+----------+
| longitude | latitude |
+-----------+----------+
| 20.5678   | 30.7891  |
+-----------+----------+
```

5.6.2.4. 空间索引

StellarDB空间索引基于 Apache Lucene Spatial 开发。可用来索引点和边的 **地理坐标** 类型的属性。空间索引可以解决附近点和区域内点的搜索问题。空间索引是将经度和纬度作为一个Spatial点存入索引，所以不支持单独进行经度或纬度的索引。在一定条件下距离函数查询也可以使用索引。

5.6.2.5. 可比性和可排序性

1. 可比性

   地理坐标数据之间不具可比性。也就是说，任何对两个地理类型点进行比较的函数操作（除了等式和不等式）返回值都为false。例如：

   ```
   return point(10.123,20.123) < point(20.5,30.5), point(10.123,20.123) > point(20.5,30.5);
   ```

   该语句执行结果如下所示：

   ```
   +---------+---------+
   | _a_c0   | _a_c1   |
   +---------+---------+
   | false   | false   |
   +---------+---------+
   ```

2. 可排序性

   a. 所有地理坐标类型的数据都是可排序的。可通过单独提取经度或者纬度后，根据对应的值使用 **ORDER BY** 进行排序。

例如：

```
match (n) return n ORDER BY <longitude(n.point) | latitude(n.point)>;
```

该语句执行结果如下所示：

```
+------------------------------------------------------------------------------------------+
|                                            n                                             |
+------------------------------------------------------------------------------------------+
| {"entityKey":[50,0,0],"properties":{"a":[20.5,30.5],"__tags":[],"__uid":"2"},"labels":["A"]}  |
| {"entityKey":[49,0,0],"properties":{"a":[20.59,50.99],"__tags":[],"__uid":"1"},"labels":["A"]} |
+------------------------------------------------------------------------------------------+
```

b. 通过经度或纬度的排序在返回n.point时需要在后面加上排序的属性。例如：

```
match (n) return n.point, longitude(n.point) ORDER BY longitude(n.point);
```

返回示例：

```
+-------------------+----------+
|       _a_c0       |   _a_c1  |
+-------------------+----------+
| [10.5,20.5]       | 10.5     |
| [20.55,30.55]     | 20.55    |
+-------------------+----------+
```

该语句使用纬度 **longitude(n.point)** 排序并且返回n.point，因此在返回结果中必须包含 **longitude (n.point)**。

地理空间类型更多用法，详见空间函数章节。

5.6.3. 时间序列类型

5.6.3.1. 时序类型特征

- 时序类型本质上是一个K-V集合，其中Key为时间戳，Value记录该时间点对应的字段值。每个时间点有且只有一个值与之对应；
- 时序类型字段的写入只允许完整创建、追加修改和完整删除。具体见后文使用方法一节；
- 时序类型支持根据时间点或时间段获取该字段值。适用于对字段值随时间变化进行分析的场景；
- 时序类型暂不支持作为过滤条件（即直接写在where语句中），不支持建索引；
- 时序类型作为字段时，单个字段内的K-V数量不宜过多。可参考下文在schema中设置字段的长度上限（推荐设为100以内），当超出上限时，自动舍弃时间最旧的一条记录。

5.6.3.2. 使用方法

以下例子演示如何在图中点上操作时序类型，结合章节创建边可类推在图中边上操作时序类型的方法。

1. schema中定义时序类型字段（使用schema创建名为ts的图，在schema中创建类型为v的点，点包含ts_field，其值类型为int且长度上限为10的时序类型字段）

```
create graph ts_graph with schema (:v{ts_field timeseries<int>(10)}) graphproperties:{...};
```

2. 或使用JSON schema创建：

```
create graph ts_graph with json schema '{
  "graph.name":"ts_graph",
  "graph.shard.number":"3",
  "graph.replication.number":3,
  "vertex.tables":
  [
    {
      "label.value":"v",
      "field.schemas":
      [
        {
          "field.type":"time_series",
          "field.name":"ts_field",
          "field.extra.attr":
            {
              "real.type":"int",
              "size.limit":"10"
            }
        }
      ]
    }
  ],
  "edge.tables":
  [
  ]
}';
```

3. 创建带有时序类型的点

```
create (:v{__uid:"1", ts_field:{localdatetime("2021-05-03T15:16:17")::200, localdatetime("2020-05-03T15:16:17")::100}});
```

时序类型的表达式为{time::value, …}，其中time需要使用localdatetime函数做类型转换。

5.6.3.3. 更新和删除方法

1. 为点a的ts_field字段追加一条记录，该记录的时间戳为语句执行时的当前系统UTC时间。

```
match (a) set a.ts_field = 300;
```

2. 为点a的ts_field字段追加一条记录，指定时间戳。

```
match (a) set a.ts_field = {localdatetime("2022-05-03T15:16:17")::300};
```

3. 删除点a的ts_field字段。

```
match (a) set a.ts_field = null;
```

5.6.3.4. 查询方法

1. 查询并返回点a完整的时间序列。

```
match (a) return to_timeseries(a.ts_field, "int");
```

2. 查询点a某一时间点对应的值。

```
match (a) return get_by_timestamp(to_timeseries(a.ts_field, "int"), localdatetime("2020-08-17T11:11:11"));
```

3. 查询点a某一时间段内的所有值。

```
match (a) return get_by_timestamp(to_timeseries(a.ts_field, "int"), localdatetime("2020-08-17T11:11:11"),localdatetime("2022-11-17T11:11:11"));
```

- 以上例句的返回结果均为{key:value, …}形式；
- 相关函数的详细说明参考手册中基础函数章节。

5.7. 数据操作语句

本章节的示例语句均可在示例图my_graph中执行，执行前请先创建示例图my_graph，建图语句如下:

```
create graph my_graph with schema (:Boy {name string, salary double, age int, single boolean,
birthday localdatetime,
reserve long, rate decimal(38,10),hobbys array<string>,geoPoint geo<double>}) (:Girl {name string,
salary double, age int, single boolean,
birthday localdatetime, reserve long, rate decimal(38,10),hobbys array<string>,geoPoint
geo<double>}) [:Friend {since int}] [:Likes {since int}]
graphproperties:{`graph.shard.number`:3,`graph.replication.number`:3};
```

5.7.1. 数据创建

5.7.1.1. 创建点

1. 创建一个uid为1，拥有name、age、salary和single等属性的点。

```
create (:Girl {name:"Kitty", age:23, salary:4250.95,
single:true,hobbys:["drawing","Photography","music"], birthday:localdatetime("1995-02-
07T14:10:38.459"),reserve:200L,rate:decimal(20000.00,10,2),__uid:"1"});
```

2. 创建两个不同类型的点。

```
create (:Girl {__uid: "2"}), (:Boy {__uid: "1"});
```

3. 创建一个点，指定name和tags。

```
create (:Girl {__uid: "3", name: "Lisa", __tags: ["human", "student"]});
```

创建点时，需要注意以下几点：

1. 创建点时，必须指定__uid。
2. 图的schema中必须包含指定的label和属性名，没有指定的属性默认为空。
3. 创建点/边时，若包含decimal类型的属性，需显式指定数据类型，如salary: decimal(10000.00,7,2)。
4. localdatetime插入数据必须严格符合格式：yyyy-MM-ddTHH:mm:ss.SSS。
5. 地理坐标类型（GEO）数据插入格式为：point(longitude,latitude).lon和lat是double的基本类型数据，顺序不能变。如 geoPoint:point(20.5,30.5)。
6. __tags为StellarDB图中的内部属性，数据类型为array<String>，默认为空，可用于表示点边多种标签。

5.7.1.2. 创建边

1. 创建uid为"2"、"4"，姓名为John和Rose两个点，并创建John指向Rose的单向边。

```
create (:Boy {__uid:"2", name:"John"})-[:Likes]->(:Girl {__uid: "4", name: "Rose"});
```

2. 创建uid为"2"、"5"，姓名为John和Rose两个点，并创建Rose指向John的单向边。

```
create (:Boy {__uid:"2", name:"John"})<-[:Likes]-(:Girl {__uid:"5", name: "Rose"});
```

3. 创建uid为"2"、"6"，姓名为John和Rose两个点，并创建Rose和John相互指向对方点的双向边。

```
create (:Boy {__uid:"2", name:"John"})-[:Likes]-(:Girl {__uid: "6", name: "Rose"});
```

4. 创建uid为"2"、"4"，姓名为John和Rose两个点，并创建Rose指向John的单向边，并指定边的extraid为"123"。

```
create (:Boy {__uid:"2", name:"John", age:23})-[:Likes {__uid: "123"}]->(:Girl {__uid: "4",
name: "Rose"});
```

5. 根据查询结果，创建John指向Rose的单向边，若a和b结果不唯一，则根据笛卡尔积结果创建边。

```
match (a:Boy {__uid:"2", name:"John"}), (b:Girl {name: "Rose"}) create (a)-[:Likes]->(b);
```

6. 直接创建uid为'1' 的Boy指向uid为'3' 的Girl的单向边，并指定边的extraid为"newyear_1"。

```
create [:Likes {__uid:"newyear_1", __usid:"1", __udid:"3", __uslabels:["Boy"],
__udlabels:["Girl"]}];
```

创建边时，需要注意以下几点：

1. 在StellarDB 5.0中，不能创建“悬挂边”。使用上述最后一种直接创建单向边的方式时，如果起点/终点不存在，则创建无效。
2. 直接创建单向边时，必须指定__uid、__usid、__udid、__uslabels以及__udlabels，并且__usid和__udid不能为空，这种语法相对于openCypher是额外功能。
3. 如果在某些情况下，实在不方便在创建边时提前创建其起点与终点，则可以设置StellarDB存储配置项 `graph.create.vertex.for.single.edge.insert` 为 `true` （也需要重启Quark服务，令其更新本配置项），这样在创建边数据时，如果起点或终点不存在，则会自动创建之。但是这种方式会导致插入数据性能略微下降，因此不推荐使用。本功能默认关闭，且对批量导入无效。
4. 起点、终点以及边的label都相同的边数据，在数据创建时视作同一条边。后导入的数据会覆盖之前的数据。若需要区分两点之间多条相同label的边，需配置生成唯一的边extraid。

5.7.1.3. 使用SQL表中的列创建点和边

当使用SQL表中的数据创建点和边时，点和边对应的是SQL表中相应的列。以下示例中，source数据库为存放原数据SQL表的关系型数据库。

1. 从source库的v表中选取uid列和name列创建点。

```
select name, uid from source.v create (:Boy {__uid: uid), name: name});
```

2. 从source库的e表中选取a列和b列，分别作为起点和终点创建边，其中起点和终点的类型为Boy和Girl,边类型为Friend。

```
select a, b from source.e create [:Friend {__usid: a, __udid: b, __uslabels: ["Boy"],
__udlabels: ["Girl"]}];
```

3. 从source库的e表中选取a列和b列，分别作为起点和终点创建边，其中起点和终点的类型为Boy和Girl，并选取c字段作为每条边的extraid(__uid)，用于两点间存在多条边的情况。

```
select a, b, c from source.e create [:Likes {__uid: c, __usid: a, __udid: b, __uslabels:
["Girl"], __udlabels: ["Boy"]}];
```

5.7.2. 更新属性值

1. 更新点a的name属性的值。

```
match (a) where uid(a) = "2" set a.name = "TOM";
```

2. 更新点a的name和age属性的值。

```
match (a) where uid(a) = "2" set a.name = "TOM", a.age = 25;
```

3. 更新点a的with属性的值（由于 **with** 是TEoC的保留关键字，所以这里必须要加`）。

```
match (a) where uid(a) = "2" set a.\`with` = "TOM";
```

4. 更新点a的__tags属性的值， 由["student","human"]更新为["woman","student","human"]。

```
match (a:Girl) where uid(a)="3" set a.__tags=add_list(toarray(a.__tags,"string"),"woman");
```

5. 更新点a的rate属性的值（rate属性为decimal类型，需需显式声明）。

```
match (a) where uid(a) = "2" set a.rate = decimal(10000);
```

6. 更新点a的geoPoint属性的值（geoPoint为地理坐标(geo)类型时，需使用point()）。

```
match (a) where uid(a) = "2" set a.geoPoint = point(lon,lat);
```

使用 **set** 语法将属性值设置为null等价于删除该属性值。

5.7.3. 删除点或边

1. 删除uid为1的点.

```
match (a) where uid(a) = "1" delete a;
```

2. 删除起点uid为1，终点uid为2的边。

```
match (a)-[f]->(b) where uid(a) = "1" and uid(b) = "2" delete f;
```

3. 删除label为Likes且since值为2010的边。

```
match [f] where f:Likes and f.since = 2010 delete f;
```

在删除点后，与其相关联的边会失效（称为悬挂边），这些边仍然会占用存储空间，并在边数统计中记入总数，但无法通过常规查询访问。

如果要删除这些悬挂边，请参考下面批量清除数据中的清理悬挂边章节。

5.7.4. 批量清除数据

对于清除全图/某个label的所有数据这些操作，StellarDB提供了便捷的方法，能够提升操作效率。

5.7.4.1. 清除label数据（新）

可以通过下列TEoC语句清除某图指定label的所有数据。当需要清除的包含该label的数据的数据量较大时，使用如下所示的操作的效率比逐条执行 **delete** 操作高。

需要注意的是，`truncate_label` 语法不支持在只有一种label的图中进行操作。此时 `truncate_label` 的操作含义为清空图数据，因此可以使用 `truncate graph` 替代。

另外需要注意的是，在使用`truncate_label` 语法清空数据时，应当先清空边数据，然后清空点数据。

```
manipulate graph <graphName> truncate_label <"vertex" | "edge"> <labelName>;
```

1. 清空label为Likes的边。

   ```
   manipulate graph my_graph truncate_label edge Likes;
   ```

2. 清空label为Girl的点。

   ```
   manipulate graph my_graph truncate_label vertex Girl;
   ```

   返回示例：

   ```
   {
          "message":"SUCCESS",     # 表示操作结果提示；如果操作失败，则会给出错误提示
          "isSuccess":true         # 表示操作是否成功
   }
   ```

清除label数据时，需要注意的是，使用truncate 点label的时候，相关联的边数据失效。

5.7.4.2. 清除全图数据

可以通过下列TEoC语句清除指定图的所有数据。本语法实际上为 **删除图** 加 **以相同schema创建新图** 两种操作合并的语法糖。

```
truncate graph <graphName>;
```

示例如下：

```
# 将my_graph的数据全部删除
truncate graph my_graph;
```

5.7.4.3. 清理悬挂边（新）

可以通过下列TEoC语句清理指定图中所有的悬挂边。

```
match [f] where is_hanging_edge(id(f)) bulk delete f;
```

5.8. 数据查询语句

查询语句遵循如下格式：

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT];
```

对Cypher较为熟悉的使用者在查询时可能会使用 `match (a)--(b)` 这类省略边细节的语法，由于在TEoC中 `"--"` 会被识别成注释，使用时需要在两个"-"间增加一个 **空格**。

本章节的示例语句均可在示例图my_graph中执行，执行前请先创建图my_graph，创建该图所使用的语句详见《数据操作语句》章节。在执行查询语句前请先在KG Explorer或beeline客户端创建数据，语句如下：

```
create (:Girl {name:'Kitty', age:23, salary:4250.95,
single:true,hobbys:["drawing","Photography","music"],
birthday:localdatetime("1995-02-
07T14:10:38.459"),reserve:1L,rate:decimal(20000.004,14,3),__uid:'1'});

create (:Girl {__uid: '2',name: 'Sherry', age: 15}), (:Boy {__uid: '1',name:'Shawn'}),(:Boy
{__uid:'8', name:'Mike', age:21});

create (:Girl {__uid: '3', name: 'Lisa', __tags: ["human", "student"]});

create (:Boy {__uid:'2', name:'John',salary:9000.44})-[:Likes {since: 2018}]->(:Girl {__uid: "4",
name: 'Rose'});

create (:Boy {__uid:'3', name:'Tom'})<-[:Likes]-(:Girl {__uid:'5', name: 'Marry'});

match (a:Boy {__uid:'3', name:'Tom'}),(b:Girl {__uid:'1'}) create (a)-[f:Likes]-(b);

create (:Boy {__uid:'4', name:'Jerry',age:21})-[:Friend {since:2020}]-(:Girl {__uid: '6', name:
'Jane',salary:7632.55});

match (a:Boy {__uid:'4'}),(b:Girl {__uid:'1'}) create (a)-[f:Likes{since:2020}]-(b);

create (:Boy {__uid:'5', name:'Mike', age:23})-[:Likes {__uid: "123"}]->(:Girl {__uid: '4', name:
'Rose'});

create [:Friend {__uid:'newyear_1', __usid:'1', since:2018, __udid:'2', __uslabels:['Boy'],
__udlabels:['Girl']}];

create (a:Boy {__uid:'6', name: 'Peter',age:23,reserve:1})-[f1 :Friend {since:2020}]->(c
:Boy{__uid:'7',name:'Jack',single:true,age: 25})-[f2:Likes {since: 2018}]->(b
:Girl{__uid:'7',name:'Julia'});

match (c :Boy{__uid:'7',name:'Jack'}) create (:Boy{__uid:'8',name:'Zhou'})-[:Friend {since: 2020}]-
(c);

match (c :Boy{__uid:'7',name:'Jack'}),(d:Boy {__uid:'2', name:'John'}) create (c)-[:Friend{since:
2018}]-(d);
```

5.8.1. 查找点

5.8.1.1. 按照属性和类型查找

1. 返回类型为Girl的点。

   ```
   match (a :Girl) return a;
   ```

2. 返回类型为Girl或Boy，年龄为23的点。

   ```
   match (a :Girl | :Boy {age: 23}) return a;
   ```

3. 返回类型为Girl的点。

```
match (a) where "Girl" in labels(a) return a;
```

4. 返回姓名为Mike，年龄为23的点。

```
match (a {name: "Mike", age: 23}) return a;
```

5. 返回类型为Girl，年龄为23的点。

```
match (a :Girl {age: 23}) return a;
```

6. 若按照decimal类型的属性值来查找，需显式指定该数值为decimal类型。

```
match (a) where a.rate = decimal(10000) return a;
```

7. 地理坐标类型的属性需通过距离来查找。如需要返回地理坐标为(20.5，30.5)的点。

```
match (a) where pt_distance(n.geoPoint,point(20.5,30.5)) = 0 return a;
```

5.8.1.2. 按照复杂条件查找

1. 返回年龄大于10岁，姓名以Tom开头的点。

```
match (a) where a.age > 10 and a.name starts with "Tom" return a;
```

2. 返回年龄等于15岁，姓名以Sherry开头，且类型为Girl的点。

```
match (a :Girl {age: 15} where a.name starts with "Sherry") return a;
```

对于pattern内部写where语句的这种写法，比如 `match (a where a.age > 1)`，该语句中的where内部不可以出现除了当前pattern定义的变量以外的其他变量。例如：

```
match (a where a.age = 23)-[f]-(b where b.age > 3)  return a, f, b;
```

这个例子中第一个where内部只能出现变量a，第二个where内部只能出现变量b。下述例句为非法例句：

```
match (a where a.age = 23)-[f]-(b where a.age > 3)  return a, f, b;
```

5.8.1.3. 按照UID查找

返回uid为’5’的点。

```
match (a) where uid(a) = "5" return a;
```

5.8.1.4. 查找精确点数量（较慢）

返回图中点的数量，返回结果为一个数值。

```
match (a) return count(a);
```

5.8.1.5. 查找近似点数量（较快）

使用 `config crux.execution.mode immediate;` 语句切换至immediate查询模式。返回图中点数量的近似数，返回结果为一个数值（点近似值在数据创建、更新、删除后有浮动，在一定时间后收敛成精确值）

```
return countapproxivertexs();
```

5.8.2. 查询边

StellarDB 5.0.0 版本中，查询边数据的全量返回值中，默认包含srcuid和dstuid属性，分别表示边对应的起点的uid和终点 的uid。

5.8.2.1. 按照属性和类型查找

1. 返回类型为Friend，属性since为2020年的边。

   ```
   match [f :Friend {since: 2020}] return f;
   ```

2. 多label查询。返回类型为Friend或Likes，属性since为2018年的边。

   ```
   match [f :Friend | :Likes {since: 2018}] return f;
   ```

3. 返回类型为Friend，属性since为2021的边。

   ```
   match [f :Friend where f.since = 2021] return f;
   ```

4. 返回since为2018的边。

   ```
   match [f where f.since = 2018] return f;
   ```

5.8.2.2.（新）查找精确边数量（较慢）

查询图中边的数量。

```
# 统计悬挂边
match [f] return count(f);

# 不统计悬挂边
match (a)-[f]->(b) return count(f);
```

5.8.2.3. 查找近似边数量（较快）

使用 `config crux.execution.mode immediate;` 语句切换至immediate查询模式，查询图中边数量的近似数。

```
return countapproxiedges();
```

- 边的近似值在数据创建、更新、删除后有浮动，在一定时间后收敛成精确值。
- `match [f where f.year = 20] return f` 这种pattern内部加where语句的写法，where语句不可以出现除了f以外的变量。
- 若按照decimal类型的属性值来查找，需显式指定该数值为decimal类型，如

```
match [f where f.length = decimal(10.00,4,2)] return f;
```

- `match [f] return count(f)` 不检查点的合法性，用户需自己保证边的起点和终点的存在性。

5.8.2.4. 单层路径查找

1. 查找一个Marry喜欢的男生名字。

```
match (a :Girl {name: "Marry"})-[f]->(m :Boy) return m.name limit 1;
```

2. 查找Shawn点和Marry点之间边的类型;

```
match (a :Boy {name: "Shawn"})-[f]-(m :Girl{name:"Marry"}) return labels(f);
```

3. 查找一个与uid为1的点有朋友或恋人关系的人名。

```
match (a)-[f :Friend | :Likes]->(b) where uid(a) = "1" return b.name limit 1;
```

4. 返回Tom点关联，并满足属性year为10的有向边的time。

```
match (a)-[f {year: 10}]->(b) where uid(a) = "Tom" return f.time;
```

5. 返回Tom点关联的人名。

```
match (a)-[f]-(b) where uid(a) = "Tom" return b.name;
```

5.8.2.5. 多层路径查找

1. 返回一个Peter点的2层有向关联点的名字。

```
match (a:Boy {name: "Peter"})-[f1 :Friend]->(c :Boy)-[f2 {since: 2018}]->(b :Girl) return b.name
limit 1;
```

2. 返回一个Peter点的2层有向关联点的名字。

```
match (a:Boy {name: "Peter"})-[ ]->()<-[ ]-(b :Boy) return b.name limit 1;
```

3. 返回一个Peter点的2层无向关联点的名字。

```
match (a:Boy {name: "Peter" })-[f :Friend *2]-(b :Boy) return b.name limit 1;
```

4. 返回Peter点的2层无向关联点的路径长度。

```
match p = (a:Boy {name: "Peter"})-[f :Friend *2]-(b :Boy) return length(p);
```

5. 返回Peter点出发的1层、2层和3层的路径长度。

```
match p = (a:Boy {name: "Peter"})-[f :Friend *1..3]-(b :Boy) return length(p);
```

6. 返回Peter点出发的1层、2层和3层的路径长度。

```
match p = (a:Boy {name: "Peter"})-[f :Friend *..3]-(b :Boy) return length(p);
```

5.8.2.6. 多层路径查找（中间点边过滤)

1. 返回从John点出发的1层、2层和3层路径，且这些路径的中间点边必须满足：中间边的since为2018。

```
match p = (a:Boy {name: "John"})-[f*1..3 `[r {since: 2018}]`]-(b) return p;
```

2. 返回从Rose点出发的1层、2层和3层路径，且这些路径的中间点边必须满足：中间点的类型为Boy且age大于20。

```
match p = (a:Girl {name: "Rose"})-[f*1..3 `(n: Boy where n.age > 20)`]-(b :Boy) return p;
```

3. 返回从Jerry点出发的1层、2层和3层路径，且这些路径的中间点边必须满足：(1)中间点的类型为Girl且age大于20；(2)中间边的类型为Likes，属性since为2020。

```
match p = (a:Boy {name: "Jerry"})-[f*1..3 `[r: Likes where r.since = 2020] (n: Girl where n.age
> 20)`]-(b :Boy) return p;
```

4. 返回从Peter点出发的1层、2层和3层路径，且这些路径的中间点边必须满足：(1)中间点的类型为Boy且single为true且age大于20；(2)中间边的类型为Friend，属性since为2020。

```
match p = (a:Boy {name: "Peter"})-[f*1..3 `[r: Friend  where r.since = 2020] (n: Boy {single:
true} where n.age > 20)`]-(b:Boy) return p;
```

5. 返回从Jerry点出发的1层、2层和3层路径，且这些路径的中间点边必须满足：(1)中间点的类型为Girl且age大于20；(2)中间边的类型为Friend或者Likes，且终点为Boy类型。

```
match p = (a:Boy {name: "Jerry"})-[f*1..3 `[r :Friend | :Likes where r.since = 2020] (n: Girl
where n.age > 20)`]-(b :Boy) return p;
```

5.8.2.7. （新）多层路径查找（无上界）

StellarDB 5.0.0支持变长路径查询中无上界的场景。

1. 返回从Jerry出发指向所有Boy节点的路径，路径中满足存在一条或者多条边

```
match p = (a:Boy {name: "Jerry"})-[f*1..]-(b :Boy) return p;
```

2. 返回从Jerry出发指向所有Boy节点的所有路径，路径中可能存在一条或者多条边

```
match p = (a:Boy {name: "Jerry"})-[f* ..]-(b :Boy) return p;
```

常见错误

1. 上述语法例子中 **r** 对应的where语句中只可以出现 **r** ， 不可以引用任何其他的变量，比如下面这两个语句为非法语句：

```
match p = (a:Boy{name: "Peter"})-[f*1..3 [`r` : Friend where `a`.age > 15]]-(b :Girl) return p;
match p = (a:Boy{name: "Peter"})-[f*1..3 [`r` : Friend where `f`.since > 20]]-(b :Girl) return
p;
```

2. 上述语法例子中 **f** 在pattern内部不可以添加任何filters ， 比如下面这两个语句非法：

```
match p = (a:Boy {name: "Peter"})-[f `:Friend`*1..3 [r: Friend {since: 10}]]-(b :Boy) return p;
match p = (a:Boy {name: "Peter"})-[f *1..3 `{f.since > 15}`[r: Friend {rank: 10}]]-(b :Person)
return p;
```

3. 上述语法例子中，变长语句内部定义的的点和边的变量名(**r** 和 **n**)不能和外部的点和边的变量名(**a** 、 **f** 等)相同，比如下面这两个语句非法：

```
match p = (a:Boy {name: "Peter"})-[f*1..3 [r: Friend where r.year = 25 and r.rank = 10] (`n`:
Girl where n.age > 20)]-(`n` :Boy) return p;
match p = (a:Boy {name: "Peter"})-[f*1..3 [`r`: Friend where r.year = 25 and r.rank = 10] (n:
Girl where n.age > 20)]-(`r` :Boy) return p;
```

4. 上述语法例子中不能直接返回变长语句中定义的点 **n** 和边 **r** ，比如下面这两个语句非法：

```
match p = (a:Boy {name: "Peter"})-[f*1..3 [`r`: Friend where r.since = 2020] (n: Person where
n.age > 20)]-(b :Person) return `r`;
match p = (a:Boy {name: "Peter"})-[f*1..3 [r: Friend where r.since = 2018] (`n`: Girl where
n.age > 20)]-(b :Person) return `n`;
```

5.9. 高级查询

5.9.1. 多match查询

返回与a同名的b点的数量。

```
match (a) where uid(a) = '3' match (b) where a.name = b.name return count(b);
```

5.9.2. OPTIONAL MATCH

在StellarDB 5.0.0版本中，图数据库继续支持OPTIONAL MATCH语法。该语法的语义为“可选择性地进行模式匹配，如果有匹配结果，则返回匹配结果；如果没有匹配结果，返回NULL”。如下示例：

```
match (a)-[r]->(b:Person)
optional match (b)-[f]->(c:Company) where c.amount < 1000000.00
return b.name, c.name;
```

如果上述示例语句中，match语句匹配到了结果b.name = "Amy"，而 optional match 子句没有匹配到结果，那么StellarDB返回一行NULL值；如果 optional match 子句有匹配结果，那么返回完整结果。

此外，在一些使用场景中，我们并不希望 optional match 子句的空返回结果被结果返回，可以通过配置 `set crux.optional.fill.null.enabled=false;` 来隐藏这些结果记录。

5.9.3. K跳邻居（新）

返回某个节点的K跳范围的邻居节点。语法规则如下：

```
with graph_bsp_khop(起点uid, 起点label, 方向, 最大跳数, 点过滤条件, 边过滤条件) as unapply(num) return num;
```

参数说明：

- 起点uid，类型：string，必填
- 起点label，类型：string
- 扩层方向，值可以有in\out\both，类型：string，必填
- 最大跳数，类型：int，必填
- 点/边过滤条件，参考点/边表达式的表示方式，类型：string，例如"(a:person where a.gender=\"male\")

使用范例：

```
with graph_bsp_khop("21250581" ,"vertex" ,"out",3,"(:vertex)","[f where f.a=2]" ) as unapply(num)
return num;
```

5.9.4. UNION和UNION ALL

1. 返回年龄等于23的Boy或等于21的Boy点的name（会对name去重）。

```
match (a:Boy) where a.age = 23 return a.name as `name` union match (a :Boy) where a.age = 21
return a.name as `name`;
```

2. 返回年龄等于23的Boy或等于21的Boy点的name（不会对name去重）。

```
match (a :Boy) where a.age = 23 return a.name as `name` union all match (a :Boy) where a.age =
21 return a.name as `name`;
```

- union去除重复内容，多个return的列名以及类型必须相同。支持属性值、点、边和路径的去重合并。
- union all不去除重复内容，多个return的列名以及类型必须相同。

5.9.5. 返回所有非匿名内容

返回a，b点内容。

```
match (a)-[]-(b) return * limit 1;
```

5.9.6. 返回null

如果uid为1的点的name值是null则返回null否则返回空值。

```
match (a) where  uid(a) = '1' and a.name is null return null;
```

5.9.7. 结果列命名

TEoC支持三种列名输出：匿名列名、表达式列名和有效列名。

查询语句 `match (a) return a.name` 的返回值所使用的列名为匿名列名_a_c0；查询语句 `match (a)-[f]-(b) return a, f, b` 中的返回值的列名分别是a，f，b。匿名列名和表达式列名均为StellarDB的默认输出。

StellarDB默认不输出有效列名，需要通过设置参数 `set crux.rename.column.name = true; set crux.rename.column.name.like.cypher = true;` 启用。语句 `match (a) return a.name` 的有效列名为a.name。

TEoC还支持使用 **AS** 关键字对返回值的列命名。指定别名后，输出列名以别名为准。

1. 返回a的年龄，列名为b。

```
match (a) with a.age as b return b limit 1;
```

2. 返回a的名字和uid，列名为nameA和uidA。

```
match (a) return tostring(a.name) as nameA, uid(a) as uidA;
```

5.9.8. 跳过前N条结果

返回a的年龄，跳过前4条记录。

```
match (a) return a.age skip 4 limit 1;
```

5.9.9. 结果排序

当前版本中，排序仅支持数值类型和String类型字段。在必要时，需要使用tointeger()，tostring()，tolong()，tofloat()，todouble()等合适的UDF，转换返回字段类型。

返回a的姓名和年龄，并按照年龄排序。

```
match (a) return a.name, tointeger(a.age) as age order by age desc limit 1;
```

5.9.10. 模糊查询

返回n的name属性中带有“ry”的点的name。

```
match (n)  where n.name=~'.*ry.*' return n.name limit 10;
```

5.9.11. DISTINCT去重

1. 查询并返回点a去重复后的姓名。

   ```
   match (a) return distinct a.name;
   ```

2. 返回对a点姓名和b点年龄两列一起去重后的结果。

   ```
   match (a)-[f]-(b) return distinct a.name,b.age;
   ```

3. 返回a点姓名，f边去重since的数量，f边since的数量。

   ```
   match (a)-[f]->(b) return a.name, count(distinct f.since), count(f.since);
   ```

DISTINCT支持属性值、点、边和路径的去重；支持多列去重；

5.9.12. 聚合函数

1. 返回点的最大salary值。

   ```
   match (n) return max(tointeger(n.salary));
   ```

2. 返回点的最小salary值。

   ```
   match (n) return min(tointeger(n.salary));
   ```

3. 返回点的salary值的总和。

   ```
   match (n) return sum(tointeger(n.salary));
   ```

4. 返回图中点的个数。

```
match (n) return count(n);
```

5. 返回图中去重后点姓名的个数。

```
match (n) return count(distinct n.name);
```

聚合函数中使用todecimal()函数时，必须指定decimal的精度与graph schema中相同，否则默认精度(decimal(10,2))以外的数据将转换为null。

错误示例：

```
match (n:Girl) return n,count(n),sum(todecimal(n.rate));
```

正确示例：

```
match (n:Girl) return n,count(n),sum(todecimal(n.rate,38,3));
```

5.9.13. CASE WHEN语句

1. 根据name值返回值。

```
match (n) return case n.name when 'John' then 1 when 'Jack' then 2 else 3 end limit 1;
```

2. 根据uid为1的点的salary值返回值，name为null返回0，否则返回1。

```
match (a) where uid(a)='1' with a.salary as x return case x when NULL then 0 else 1 end as
result;
```

3. 根据uid为1的点的age值返回值，name为"Tom"返回0，否则返回1。

```
match (a) where uid(a)='1' with a.name as x return case when x = "Tom" then 0 else 1 end as
result;
```

4. 根据uid为2的点的age值返回值，name为null返回0，否则返回1。

```
match (a) where uid(a)='2' with a.age as x return case when x is NULL then 0 else 1 end as
result;
```

5.9.14. WITH的用法

1. 在参数传递时使用。下面所示例句将第一个match获取的查询结果num1和num2传递给第二个查询。

```
match (a)-[f]->(b) where uid(a) = "6" and uid(b) = "4" with count(a) as num1,b.name as
name1,count(b) as num2 match (c)-[f]->(b) where num1 = c.reserve return
name1,c.name,num2,labels(f);
```

2. 将点按照年龄分组，并计算每个年龄的点的个数。

```
match (a) with a.age as age, count(a.age) as count return age, count;
```

5.9.15. 唯一性语义

1. match的默认语义为路径边不重复，即返回结果path中每条路径上的边具有唯一性。

```
match path=(a)-[f*3..5]->(b) return path;
```

通过执行 `set crux.dedup.relationship.switch=false;` 语句关闭 `crux.dedup.relationship.switch` 参数，这样可以切换match语义成路径点不重。

2. traverse的语义为路径点不重复，即每条路径上的点具有唯一性，可用于过滤环路。

```
traverse path=(a)-[f*3..5]->(b) return path;
```

5.9.16. 简单跨图查询

在当前版本中，跨图查询只支持简单的match查询语句，语法结构如下：

```
[MATCH] {GRAPH_NAME}[.]{PATTERN} RETURN {PROPS};
```

使用示例如下：

1. 选择图graph1，跨图查询图graph2的点属性id。

```
use graph graph1;
match graph2.(a) return a.id;
```

2. 选择图graph1，跨图查询图graph2中uid为"x1"和"x2"的点及图graph3中uid为"x3"的点。

```
use graph graph1;
match graph2.(a{__uid:"x1"}),(b{__uid:"x2"}) match graph3.(c{__uid:"x3"}) return a,b,c;
```

3. 选择图graph1，跨图查询图graph2中uid为"x1"的点，根据该点的属性id加1，查询图graph1中uid为该点id加1的点。

```
use graph graph1;
match graph2.(a{__uid:"x1"}) match (b) where uid(b) = tostring(a.id + 1) return b;
```

5.9.17. （新）查询最短路径

在当前版本中，TEoC支持多种最短路径的计算方法，在功能满足的情况下，推荐使用BSP最短路径，可以获得更好的性能。

1. BSP最短路径（新）

   BSP最短路径的语法如下：

```
with graph_bsp_sssp(起点uid, [起点label], 终点uid, [终点label], 方向, 最大跳数, 返回值,
点过滤条件, 边过滤条件) return 返回值;
```

参数说明：

起点uid和终点uid的类型为string类型，不能为空

起点label和终点label的类型为string类型，可以为空

方向的可选值有 "out"\"in"\"both"，为string类型，不能为空

最大跳数为限制两点间最短路径的最大长度，为int类型，-1表示不作限制

返回值的可选值有，"length"\"count"\"length_count"\"path"\"path_single"，分别表示返回路径长度\路径条数\长度和条数\所有路径\一条路径。路径使用点的__uid属性表示。作为参数的返回值是需要双引号的，return后的不用加双引号。

点过滤条件和边过滤条件可仿照TEoC语法书写，并且放在一对双引号内，相应地要对过滤条件中的符号进行转义，例如 "(a:person where a.gender=\"male\")"

```
with graph_bsp_sssp( "39687658" ,"vertex", "47978366", "vertex", "out",-1, "length","","") as
unapply(length) return length;

with graph_bsp_sssp( "39687658" ,"vertex", "47978366", "vertex", "out",-1, "length","(a:person
where a.gender=\"male\")","[f:person_knows_person]") as  unapply(length) return length;
```

2. search … shortestPath

其语法如下所示。其他最短路径的计算方法，请参考图算法章节。

```
search ${路径变量名} = shortestPath((起始点变量名), (终点变量名), (经过点变量名), [经过边变量名],
是否为双向边, 遍历最短路径的最大路径长度) where 起点/终点/经过边的条件/路径条件 return 路径变量名相关的表达式
```

使用示例如下：

a. 查询起点为1，终点为10的最短路径。

```
search p = shortestPath((src),(dst),(passNodes),[passEdges]) where uid(src) = "1" and
uid(dst) = "10" return p;
```

b. 查询起点为1，终点为10的最短路径的长度。

```
search p = shortestPath((src),(dst),(passNodes),[passEdges]) where uid(src) = "1" and uid(dst) =
"10" return length(p);
```

a. 找出起点为1，终点为10，路径中边的amount值为1，最大查找长度为5的路径。

```
search p = shortestPath((src),(dst),(passNodes),[passEdges],true,5) where uid(src) = "1" and
uid(dst) = "10" and passEdges.amount = 1 return p;
```

b. 找出起点为1，终点为10，最大查找长度为5的路径，并输出其中两条最长的路径。

```
search p = shortestPath((src),(dst),(passNodes),[passEdges],true,5) where uid(src) = "1" and
uid(dst) = "10" with p return length(p), p order by length(p) desc limit 2;
```

c. 找出起点为a1，终点为a2，经过点label为N1，经过边label为E1的最短路径。

```
search p=shortestPath((a),(b),(c),[d],false) where uid(a) = 'a1' and uid(b) = 'a2' and
collection_contains(labels(c),["N1"]) and collection_contains(labels(d),["E1"]) return p;
```

在使用以上方法查询最短路径时，有以下几点需要注意：

在对最短路径经过的点的进行过滤的时候，过滤条件也会对起点和终点进行过滤；

对于两点之间存在多条关联边的复杂图，算法返回最短层级的所有路径；

若起点和终点不唯一，将对每对起终点运行算法。

5.9.18. 查询结果导出到Quark SQL表

结果导出语法结构如下：

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
STORE {COL0}, {COL1}, ..., {COLN} INTO TXT|CSV|ORC|HOLODESK TABLE {DATABASE_NAME}.{TABLE_NAME};
```

使用示例如下：

1. 将a的uid以及a.age转换成string和int类型存入test_db.test_tbl表。

   ```
   match (a) store toString(uid(a)), tointeger(a.age) into txt table test_db.test_tbl;
   ```

2. 将a的uid以及a.age转换成string和decimal类型存入test_db.test_tbl1表。这里使用 **todecimal()** 方法时需要在 **todecimal()** 中手动指定decimal类型字段的精度，该精度需要和表中该字段精度匹配。此外该精度还需要和图schema中定义的精度匹配，否则将会被识别为null。

   ```
   match (a) store toString(uid(a)), todecimal(a.age,3,0) into txt table test_db.test_tbl1;
   ```

3. 将a的uid以及a.salary转换成string和double类型存入test_db.test_tbl2表。

   ```
   match (a) store toString(uid(a)), todouble(a.salary) into csv table test_db.test_tbl2;
   ```

4. 将a的uid以及a.arrAttr转换成string类型存入test_db.test_tbl3表,其中mkstring函数的第二个参数为分隔符。

   ```
   match (a) store toString(uid(a)), mkstring(a.arrAttr, ",") into orc table test_db.test_tbl3;
   ```

5. 将a.geoPoint以及a.timeSeries转换成string类型存入test_db.test_tbl4表,其中maptostring函数的第二个参数为时间戳和值之间的分隔符，第三个参数为时间序列各元素之间的分隔符

   ```
   match (a) store mkstring(toarray(a.geoPoint, "double"), ","),
   maptostring(to_timeseries(a.timeSeries, "int"), ",", "\|") into orc table test_db.test_tbl4;
   ```

在使用上述方法导出数据时，需注意以下几点：

1. 当前版本中支持导出的数据类型string、integer、double、long以及decimal。其他类型通过UDF（如

toString()、toInteger()、toDouble()、toLong() 以及 todecimal()）转换，array、geo类型通过 mkstring() 转换后导出；timeseries类型通过 maptostring() 转换后导出。

2. 目标表不存在时会自动创建；若目标表存在，导出内容需要与表结构匹配。
3. 目标表若存在，导出字段名需要以as重命名为目标列名。(如 store toString(uid(a)) as x)。
4. HOLODESK 是星环科技分布式分析型数据库 ArgoDB 中所使用的表，在数据导入HOLODESK表时，必须有对应产品地 license。在使用时另需注意，HOLODESK表暂时不支持StellarDB当前版本中的 array 类型。
5. 导出数据至Quark的CSV表时，列名使用默认列名，如果需要指定列名，可参照下述语法替换相对应的NAME：

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
STORE {COL0} [as {NAME0}], {COL1} [as {NAME1}], ..., {COLN} [as {NAMEN}] INTO
TXT|CSV|ORC|HOLODESK TABLE {DATABASE_NAME}.{TABLE_NAME};
```

5.9.19. 查询结果导出到StellarDB其他图

语法结构如下：

```
[MATCH WHERE] [WITH] ...[MATCH WHERE]
SET {FIELD0}.xx = xx, ...,{FIELDN}.xx = xx
STORE {FIELD0}, {FIELD1}, ..., {FIELDN} INTO GRAPH {GRAPH_NAME}
[RETURN]
```

使用示例如下：

1. 将当前图中label为A的点的long_col属性赋值为int_col属性值+1，之后将a存入另一个名为store_sub_graph_sub1的图中，返回变更前a的属性long_col的原值。

```
match (a:A) set a.long_col = a.int_col + 1 store a into graph store_sub_graph_sub1 return
a.long_col;
```

2. 在当前图中查询(a)-[f]->(b)，并将结果中f的int_col属性赋值为常量31。之后将赋值后的f存入图store_sub_graph_sub1，返回变更前的a.long_col最小值和f.long_col最大值。

```
match (a)-[f]->(b) set f.int_col = 31 store f into graph store_sub_graph_sub1 return
min(a.long_col), max(f.long_col);
```

3. 若当前图为store_sub_graph，创建另一个与图store_sub_graph的schema一致的图store_sub_graph_sub1。将查询store_sub_graph_sub的结果计算变更后存入图store_sub_graph_sub1，之后跨图查询store_sub_graph_sub1。

```
use store_sub_graph;

drop graph store_sub_graph_sub1 if exists ;

# 根据已有原图的schema创建目标图
create graph if not exists store_sub_graph_sub1 with schema from graph store_sub_graph;

# 目标图的点为所有label添加新属性path_nodes
add_field store_sub_graph_sub1  ({path_nodes array<string>}) all;

# 目标图的边为所有label添加新属性hop_num
add_field store_sub_graph_sub1  [{hop_num int}] all;

# 指定新属性的值，并导出点边到目标图
match p=(a)-[f]->(b) set a.path_nodes =[uid(a)], b.path_nodes =  [uid(a), uid(b)],f.hop_num =  1
store a,b,f into graph store_sub_graph_sub1 return max(tostring(uid(f)));

# 跨图查询目标图结果
match store_sub_graph_sub1.(a)-[f]->(b) return a.path_nodes, b.path_nodes, f.hop_num;
```

使用上述方法时需要注意以下几点：

1. 若目标图不存在时需要手动创建。从当前图中导出的数据所对应的图结构需要与目标图结构匹配。这里推荐根据已有原图的schema创建目标图，之后再向目标图中添加需要的字段。
2. 支持导出点和边，**set** 语句赋值目标图独有属性值和修改原有属性值。

5.9.20. Session级WHILE循环查询

在当前版本中，TEoC支持简单的seesion级while循环结构的查询语句。循环查询语法结构如下：

```
[WHILE] [(] [CONDITION] [)]
[DO] [{] {QUERY1};{QUERY2}..{QUERYN} [}]
```

需要注意的是，若在JDBC中使用该语句，JDBC需关闭查询语句按分号切分的功能。若使用版本不支持该操作，请更新客户端。关闭该功能的方法是在JDBC连接串后面添加参数 `";splitSQLBySemicolon=false"`。

示例语句如下：

1. 循环求1到100的和。

```
drop all vars;

decl x: int assign x:= 1;

decl sum: int assign sum:= 0;

while (@x < 101) do {sum := @sum + @x; x := @x + 1;};
return @sum;
```

2. 反复查询15次业务语句并将带有每层信息的点存入目标图，之后对目标图执行查询操作。

```
use enb_test1;
# 关闭该参数对自定义参数的影响
set crux.queryplan.cache = false;
drop all vars;

# 起点id
decl sid:string assign sid:="125";

# 记录每次查询是否有结果
decl temp_result:string;
decl hop_num:int assign hop_num:= 1;
drop graph enb_test_dq1 if exists;
create graph enb_test_dq1 with schema from graph enb_test1;

# 为目标图中的所有点增加array<string>类型的属性path_nodes 用于记录从初始点到当前点经历过的所有不重复点的uid
add_field enb_test_dq1 ({path_nodes array<string>}) all;

# 新增属性hop_arr记录遍历每一跳后，遍历过的点所经历过的跳数
add_field enb_test_dq1 ({hop_arr array<int>}) all;

# 新增属性hop_num记录遍历每一跳后，该点最后一次经历的跳数
add_field enb_test_dq1 ({hop_num int}) all;

# 初始化
temp_result:= match (a:Substation)-[:Substation_Outflow_ACLineSegment]->(s:ACLineSegment)-
[:Substation_Inflow_ACLineSegment]->(b:Substation)
where uid(a) = @sid
set b.path_nodes = [uid(a), uid(b)],b.hop_arr = [@hop_num],b.hop_num=@hop_num store b into graph
enb_test_dq1 return max(tostring(uid(b)));

# 循环查询15层结果存目标图，没结果则中止查询
while (@temp_result is not null and @hop_num < 15) do { hop_num:= @hop_num+1; temp_result:=
match enb_test_dq1.(a) where a.hop_num=@hop_num-1 with uid(a) as id, toarray(a.hop_arr, "int")
as hops, toarray(a.path_nodes, "string") as dedup match (a:Substation)-
[:Substation_Outflow_ACLineSegment]->(s:ACLineSegment)-[:Substation_Inflow_ACLineSegment]-
>(b:Substation) where not uid(b) in dedup and uid(a) = id set b.path_nodes = dedup + uid(b),
b.hop_arr = hops + @hop_num, b.hop_num=@hop_num store b into graph enb_test_dq1 return
max(tostring(uid(b))); };

# 跨图查询每层结果
match enb_test_dq1.(a) return a;
```

- 关闭参数splitSQLBySemicolon之后，还需将整个while语句写在同一行。
- **decl** 为变量声明关键字，使用该关键字的详细介绍，请参阅《变量声明》章节。

5.9.21. 查询级别参数配置

如果需要某些参数配置只在本次查询中生效，那么就可以进行查询级别的参数配置。其使用应用场景大致有：

- 若在KG Explorer端进行查询操作，可能存在复用session的情况。此时需要查询级别的具有明确的并且独占的配置信息的上下文。
- 对于bulkload等特殊的查询，需要设置独有的参数。比如查询模式，或者一些数据清理时间的设定，便可以设置生命周期为查询级别生效的配置信息可以避免反复调整配置。

该方法的语法如下：

```
using context /* use graph XXX; config crux.execution.mode XXX; set XXX=XXX;*/  match (a) return a;
```

示例语句如下所示：选择图my_graph，设置查询模式为analysis，查询点的个数。

```
using context /* use graph my_graph; config crux.execution.mode analysis;*/  match (a) return
count(a);
```

/**/ 中只能有 use graph，config crux.execution.mode 以及 set 三种形式。

5.10. 更新Schema

5.10.1. 更新点属性（新）

更新点属性包含对点属性的增加、删除和修改。语法如下所示：

```
<alter_type> <graph_name> (:<label_name> {<field_name> <field_type> [index:DEFAULT]});
```

1. 新增属性示例如下所示：
   a. 为图bank，EMPLOYEE类型的点添加emp_addr字段，不建立索引。

      ```
      add_field bank (:EMPLOYEE {emp_addr string});
      ```

   b. 为图bank，EMPLOYEE类型的点添加emp_addr字段，并添加索引。

      ```
      add_field bank (:EMPLOYEE {emp_addr string index:DEFAULT});
      ```

   c. 为图bank，所有类型的点添加emp_addr字段，不建立索引。

      ```
      add_field bank (:EMPLOYEE {emp_addr string}) all;
      ```

2. 删除属性示例如下所示：
   a. 为图bank，EMPLOYEE类型的点删除emp_addr字段。

      ```
      delete_field bank (:EMPLOYEE{emp_addr string});
      ```

   b. 为图bank，所有类型的点删除emp_addr字段。

      ```
      delete_field bank (:EMPLOYEE{emp_addr string}) all;
      ```

3. （新）修改属性示例如下所示：为图bank，EMPLOYEE类型的点修改emp_addr字段，将其类型改为int。

```
## 低版本
alter_field bank (:EMPLOYEE {emp_addr int});

## StellarDB 5.0版本
delete_field bank (:EMPLOYEE {emp_addr int});
add_field bank (:EMPLOYEE {emp_addr int});
```

StellarDB 5.0.0将不再支持alter_field语法，原有的alter_field语句可以统一使用delete_field和add_field语句替代。可参考如上所示例子

5.10.2. 更新点label

更新点label包含新增和删除点label。

1. 新增点label的语法如下所示：

```
alter_graph_schema <graph_name> (:<label_name>{<field_name1 field_type1 [index:DEFAULT], ... ,
field_nameN field_typeN [index:DEFAULT]>});
```

使用示例如下所示：为图bank添加一个EMPLOYEE类型的点，其中emp_addr字段建立索引。

```
alter_graph_schema bank (:EMPLOYEE {emp_addr string index:DEFAULT, age int, salary double});
```

2. 删除点label的语法如下所示：

```
delete_label <graphname> (:<vlabel_name>);
```

使用示例如下所示：为图my_graph删除f_1类型的点。

```
delete_label my_graph (:f_1);
```

删除点label的语法仅支持一次性删除一个label，删除多个label需要多次删除。

5.10.3. 更新边属性（新）

更新边属性包含对边属性的增加、删除和修改。语法如下所示：

```
<alter_type> <graph_name> [:<label_name>{<field_name> <field_type> [index:DEFAULT]}];
```

1. 新增属性示例如下所示：
   a. 为图bank，BORROW类型的边添加emp_addr字段，不建立索引。

   ```
   add_field bank [:BORROW{emp_addr string}];
   ```

   b. 为图bank，BORROW类型的边添加emp_addr字段，并添加索引。

   ```
   add_field bank [:BORROW {emp_addr string index:DEFAULT}];
   ```

   c. 为图bank，所有类型的边添加emp_addr字段。

   ```
   add_field bank [{emp_addr string}] all;
   ```

2. 删除属性示例如下所示：
   a. 为图bank，BORROW类型的边删除emp_addr字段。

   ```
   delete_field bank [:BORROW{emp_addr string}];
   ```

   b. 为图bank，所有类型的边删除emp_addr字段。

   ```
   delete_field bank [{emp_addr string}] all;
   ```

3. （新）修改属性示例如下所示：为图bank，BORROW类型的边修改emp_addr字段，将其类型改为int。

```
## 低版本
alter_field bank [:BORROW {emp_addr int}];

## StellarDB 5.0版本
delete_field bank [:BORROW {emp_addr int}];
add_field bank [:BORROW {emp_addr int}];
```

5.10.4. 更新边label

更新边label包含新增和删除边label。

1. 新增边label的语法如下：

```
alter_graph_schema <graph_name> [:<label_name>{<field_name1 field_type1 [index:DEFAULT], ... ,
field_nameN field_typeN [index:DEFAULT]>}];
```

使用示例如下所示：

a. 为图bank添加一个BORROW类型的边，其中emp_addr字段建立索引。

```
alter_graph_schema bank [:BORROW{emp_addr string index:DEFAULT, age int, salary double}];
```

b. 为图bank添加一个BORROW类型的边，其中emp_addr字段建立索引，__EXTRAID字段建立索引。

```
alter_graph_schema bank [:BORROW{emp_addr string index:DEFAULT, age int, __EXTRAID string
index:DEFAULT}];
```

__EXTRAID字段用于标识两点间多条边的唯一性，如果希望快速过滤该字段的话，需要添加__EXTRAID字段的索引。

2. 删除边label语法如下所示：

```
delete_label <graphname> [:<elabel_name>];
```

使用示例如下所示：为图my_graph删除f_1类型的边。

```
delete_label my_graph [:f_1];
```

删除边label的语法仅支持一次性删除一个label，删除多个label需要多次删除。

5.10.5. 更新comment注释信息

更新label注释信息和字段注释信息。语法如下所示：

```
alter_label_comment <graphName> (:<label> comment <label_comment>);
alter_field_comment <graphName> [:<label> {<field_name> <field_type> comment <field_comment>}];
```

1. 修改图abc的label名为A的label注释信息

```
alter_label_comment abc (:A comment "Abc_info");
```

2. 修改图abc的label名为A的点中的属性name的注释信息

```
alter_field_comment abc (:A {name string comment "nameAbc_info"});
```

5.11. 表达式

5.11.1. 类型表达式

类型	例子
十进制型整数	10, -213
十进制小数	1.25, 3.604E-14, -2.31
十进制型长整数	1993458435921, -12381543923L
任意精度的有符号十进制数	123bd,123.31BD
八进制整数(0开头)	084, -096
字符串	"星环", '信息科技'
布尔类型	true, false, TRUE, FALSE
数组类型	[1, 2, 3],["星环","信息科技"],[decimal(10.2,3,1),decimal(100.2,3,2)],[localdatetime("2021-01-18T09:50:12.627"),localdatetime("2021-11-18T03:50:12.113")]
时间类型	localdatetime("2021-01-18T09:50:12.627")
Decimal类型	decimal(10.2,3,1)
地理空间类型	point(20.5,30.5),point(-20.5,-30.5)
时序类型	{localdatetime("2023-01-01T15:16:17")::"nice"}, {localdatetime("1997-01-01T15:16:17")::1997, localdatetime("1998-01-01T15:16:17")::1998}

5.11.2. 指代型表达式

类型	例子
变量	x, rel, myVar
属性	a.prop, x.prop, rel.'surprise that this is a kind of property'
函数调用	lenght(p), sum(a.salary)
路径	(a)-[]->(b)-[]->(c)

5.11.3. 混合型表达式

类型	例子
符号运算	1*2+3, 1<2, -x
正则表达式	transwarp.name =~'[a-z]'
简单字符串匹配	a.name starts with "trans"
逻辑控制	case n.category when 'food' then 1 when 'car' then 2 else 3 end

5.11.4. 命名规则

类型、属性名和变量名，遵循如下命名规则：

规则	正确用例	错误用例
必须以字母开头	accustom	_accustom
可以有数字，但不能作为开头	name1	1name
不能包含符号，下划线除外	name_g	name*
大小写敏感	School，sChool，school是三个不同的类型；x和X是不同的变量	

单词间多余的空格会被自动删去：match (a) return a 和 match (a) return a 等价。

5.11.5. 保留关键字

在当前版本中，TEoC不区分保留关键字的大小写，部分留作未来使用。

ADD • ALL • ALTER • AND • AS • ASC • ASCENDING • ASSERT • BULK • BY • CALL • CASE • COMMIT • CONSTRAINT • CONTAINS • COPY • CREATE • CSV • DATABASE • DECL • DELETE • DESC • DESCENDING • DESCRIBE • DETACH • DISTINCT • DO • DROP • ELSE • END • ENDS • EXISTS • FILE • FOR • FOREACH • FROM • IN • INDEX • INSERT • INSTALL • INTO • IS • JOIN • JSON • KEY • LIMIT • LKJOIN • LOAD • MANDATORY • MATCH • MERGE • NODE • NOT • OF • ON • OPTIONAL • OR • ORDER • PERIODIC • REDUCE • RELATIONSHIP • REMOVE • REQUIRE • RETURN • RUN • SCALAR • SCAN • SCHEMA • SEARCH • SET • SHOW • SKIP • START • STARTS • THEN • TRUNCATE • UNION • UNIQUE • UNWIND • UPDATE • UPSERT • USING • WHEN • WHERE • WITH • XOR • YIELD • false • null • true

在查询语句中使用保留关键字作变量名或者类型名等场景，TEoC 与很多其他语言一样，需要使用反引号"`"将关键字括起来，例子如下：

```
match (`true`) return `true`;
```

5.11.6. 符号

类型	符号	示例
属性符号	.	match (a) return a.name;
数学符号	+, -, *, /, %, ^	return 1 + 2;
比较符号	=, <>, !=, <, >, , >=, IS NULL, IS NOT NULL	详见《比较符号》小节
字符串匹配符号	STARTS WITH, ENDS WITH, CONTAINS, =~(正则表达式)	详见《字符串匹配符号》小节
逻辑符号	AND, OR, XOR, NOT	详见《逻辑符号》小节
字符串运算符号	字符串合并	return "Shanghai" + ", China";
集合符号	集合操作	详见《集合符号》小节

比较符号

查询	返回

return 1 = 2	false
return 1 < 2	true
return 1 is null	false
return 1 is not null	true
return 1 <> 1	false
return 1 != 2	true

字符串匹配符号

查询	返回
match (a) with tostring(a.name) as name where name **starts with** "abc" return name	abcTest
match (a) with tostring(a.name) as name where name **ends with** "abc" return name	testabc
match (a) with tostring(a.name) as name where name=**~"t[a-z]+"** return name	test

逻辑符号

查询	返回
match (a) where a.prop1 = 1 and a.prop2 = 2 return uid(a)	uid1
match (a) where a.prop1 = 1 or a.prop2 = 2 return uid(a)	uid2
match (a) where a.prop1 = 1 xor a.prop2 = 2 return uid(a)	uid3
match (a) where not a.prop1	uid4

集合符号

查询	返回
return 1 in [1, 2]	true
unwind [1, 2, 3] as p return p	1 2 3

5.11.7. 转义字符

转义字符	十六进制	含义
\a	0x07	响铃
\b	0x08	退格，将当前位置移到前一列
\f	0x0C	换页，将当前位置移到下页开头
\n	0x0A	换行，将当前位置移到下一行开头
\r	0x0D	回车，将当前位置移到本行开头
\t	0x09	水平制表
\v	0x0B	垂直制表
\\	0x5C	反斜线

\’	0x27	单引号
\″	0x22	双引号
\?	0x3F	问号
\uhhhh		4位十六进制数据所表示的字符，如\u0014
\Uhhhhhhhh		8位十六进制数据所表示的字符，如\U0002A6A5

TEoC与很多其他语言一样，当字符串中含有转义字符时，将对所指符号进行转义，例子如下所示，其返回结果是一个双引号。

```
return "\"";
```

如果TEoC语句中的字符串含有转义字符，会自动将转义字符转义。如不需要转义，可以通过 `set crux.escape.sequence.enabled = false` 来关闭自动转义功能。

5.12. Explain语句结构

使用explain语句可以查看TEoC语句的逻辑执行计划，能够帮助用户排查问题、优化TEoC语句。简单地讲，explain语句的使用方法为**explain + TEoC** 语句，详细语法遵循如下格式。该语句的返回结果为语句的逻辑执行计划树型结构。

```
[EXPLAIN]
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT];
```

示例：

```
explain match (a:Boy) return a
```

返回结果：

```
| Normal Query
Log_POutputPlan(id: 69)
| output: {_a_C0: GraphNode}
| Column Prune State: null
| physicalOP extreme optimize: disabled
| COptimization: success
|_Log_PSelectOutputPlan(id: 70)
  | output: {_a_C0: GraphNode}
  | Column Prune State: (0:All)
  | attributes: make_node(_a_C0,_a_C1,_a_C2)
  |_Log_PFilterPlan(id: 68)
    | output: {_a_C0: LCruxBinary;, _a_C1: Array[String], _a_C2: Map[String,Dynamic]}
    | Column Prune State: null
    | filter: collection_contains(_a_C1,array(const: Boy))
    |_Log_ScanPlan(id: 67)
        output: {_a_C0: LCruxBinary;, _a_C1: Array[String], _a_C2: Map[String,Dynamic]}
        Column Prune State: (0:All,1:All,2:NeedAll-{})
        Scan Symbol: null
        Table Scan Signature: my_graph.__graph_node
        Attached Variable: a
        EmbeddedFilter: None
        scanPushDown: collection_contains(_a_C1,array(const: Boy))
        Statistics: null
```

explain l + TEoC语句、 **explain logical + TEoC语句** 与 **explain + TEoC语句** 的效果相同，都可以展示TEoC语句的逻辑执行计划。

5.13. 变量声明

5.13.1. 声明简介

声明是指为特定数据类型的变量分配一定的存储空间，并命名该变量以便引用它；必须先声明变量，然后才能引用它；对声明的变量可以进行赋值操作来改变它的值；声明的变量其作用域是Session级别的。

5.13.2. 变量声明

使用 **decl** 关键字声明一个变量必须为变量指定名称和类型，且名称不能与已有的变量名相同。声明但未赋值的变量的默认值为null。变量名声明对大小写敏感。变量声明的语句遵循如下格式：

```
DECL [<variable_name>:<variable_type>];
```

使用方法示例如下表所示：

语句	说明
decl x:int;	声明一个类型为 int 的变量 x
decl s:string;	声明一个类型为 string 的变量 s
decl l:long;	声明一个类型为 long 的变量 l
decl b:boolean;	声明一个类型为 boolean 的变量 b
decl d:double;	声明一个类型为 double 的变量 d
decl time:localdatetime;	声明一个类型为 localdatetime 的变量 time
decl d1:decimal;	声明一个类型为 decimal 的变量 d1
decl list1:list[int];	声明一个类型为 list[int] 的变量 list1
decl list2:list[double];	声明一个类型为 list[double] 的变量 list2

5.13.3. 变量赋值

对已经声明的变量可以进行赋值操作，并且在给变量赋值时必须保证赋值类型与变量类型相同。此外可以将声明语句和变量赋值语句连着写。变量赋值语句遵循如下格式：

```
[ASSIGN] [<variable_name>:=] [EXPRESSION | QUERY];
```

使用示例如下表所示：

1. 声明一个类型为int的变量x，给变量x赋值为1。

```
decl x:int
assign x:= 1;
```

2. 声明一个类型为decimal的变量d1，将变量d1赋值为12345.6549。

```
decl d1:decimal
assign d1:=12345.6549bd;
```

3. 声明一个类型为string的变量y，将uid为1的点的name赋值给y。

```
decl y:string
match (a) where uid(a)='1' assign y:= tostring(a.name);
```

4. 声明一个类型为double的变量d，将uid为1的点的weight赋值给d。

```
decl d:double
assign d:= match (a) where uid(a)='1' return todouble(a.weight);
```

5. 声明一个类型为int的变量z，并且赋值为1。

```
decl z:int assign z:= 1;
```

6. 声明一个类型为long的变量l，并将uid为1的点的salary赋值给l。

```
decl l:long assign l:= match (a) where uid(a)="1" return tolong(a.salary);
```

5.13.4. 变量取值

使用变量取值语句对已声明的变量取值，且该语句遵循如下格式：

```
[@<variable_name>];
```

使用示例如下所示：

1. 返回变量x的值。

```
return @x;
```

2. 返回age等于变量x的值的所有点。

```
match (a) where tointeger(a.age)= @x return a;
```

5.13.5. 删除变量

删除变量包含删除指定变量和删除所有变量，删除指定变量语句遵循如下格式：

```
[DROP]
[VAR | VARIABLE]
[<variable_name>];
```

删除全部变量语句遵循如下格式：

```
[DROP ALL]
[VARS | VARIABLES];
```

使用示例如下表所示：

语句	说明
drop var x;	删除变量 x
drop variable x;	删除变量 x
drop all vars;	删除所有的变量
drop all variables;	删除所有的变量
decl y:int assign y:= 1 return @y drop var y;	声明一个类型为 int 的变量 y 将 y 赋值为 1 返回变量 y 的值 删除变量 y

5.13.6. 显示所有变量

显示所有变量语句遵循如下格式：

```
[SHOW]
[VARS | VARIABLES];
```

使用方法如下所示：

1. 显示所有已声明的变量。

```
show vars;
```

2. 显示所有已声明的变量。

```
show variables;
```

5.14. 重命名图

可以使用 **rename** 对图进行重命名，且在重命名图的过程中，必须保证不存在新图。重命名后，旧图所有信息将被新图所继承，不再被保留。重命名图的语法如下所示，其中 **old_graph** 为需要重命名的旧图名称，**new_graph** 为需要生成的新图名称。

```
RENAME GRAPH <old_graph> AS <new_graph>;
```

使用示例如下所示：

```
rename graph old_graph_test as transwarp_graph;
```

在使用重命名图功能时，新图和旧图对应的数据库中不推荐混入SQL table。

5.15. 图拷贝

5.15.1. 使用COPY进行图拷贝

使用 **copy** 关键字进行图数据拷贝，可以将数据从原图中拷贝至指定的新图当中。如果新图不存在，则会创建该新图；若新图已经存在，必须确保新旧两个图的schema相同。

使用示例如下：

1. 将图old_graph的点和边拷贝到图new_graph中。

```
copy graph old_graph into graph new_graph;
```

2. 将图old_graph的点拷贝到已经存在的图new_graph中。

```
from old_graph match (a) copy to new_graph insert (a);
```

3. 将图old_graph的边拷贝到已经存在的图new_graph中。

```
from old_graph match [f] copy to new_graph insert [f];
```

5.15.2. 使用BULK COPY进行图拷贝（新）

StellarDB 5.0.0当前只支持在图数据库升级，即历史图数据迁移至新版本数据库时使用该语法，该语法暂不支持在其他任何场景中使用。

1. 将图old_graph的点和边批量拷贝到图new_graph中。

```
bulk copy graph old_graph into graph new_graph;
```

5.15.3. 图拷贝语法扩展

除了在 **copy** 中使用 **insert** 的导入方式之外，图拷贝语法还支持 **upsert** 和 **update** 两种导入方式（使用时只需要用对应的关键词替换上面的 **insert** 即可），它们的不同在于：

- **insert** 导入方式会让新数据覆盖所有的原始数据；
- **update** 导入方式仅更新已经存在的数据，对于不存在的数据，不进行任何操作。

进行图拷贝时需注意以下几点：

- 图拷贝建议在 **analysis** 模式下进行使用。

5.16. 函数

5.16.1. 基础函数

函数名	参数	功能	示例	备注
timestamp	/	1970.1.1到现在的毫秒数	return timestamp();	
id	类型为点或者边的实体	返回点或边的内部id	match (a) return id(a);	5.0.0版本返回值不可读，不推荐使用
startid	类型为边的实体	返回边的起点的内部id，StellarDB 5.0.0版本改进了数据结构，此函数返回结果不可识读	match [f] return startid(f);	5.0.0版本返回值不可读，不推荐使用
endid	类型为边的实体	返回边的终点的内部id，StellarDB 5.0.0版本改进了数据结构，此函数返回结果不可识读	match [f] return endid(f);	5.0.0版本返回值不可读，不推荐使用
hex_string_id	类型为点或边的实体	返回点或边内部id的hex编码	match (a) return hex_string_id(a);	
uid	类型为点或者边的实体	返回点或边的用户指定uid（如果类型为边，而且用户没有指定uid，那么返回空字符串""）	match (a) return uid(a);	
startuid	类型为边的实体	返回边的起点uid	match [f] return startuid(f);	
enduid	类型为边的实体	返回边的终点uid	match [f] return enduid(f);	
euid	类型为边的实体	返回边的用户指定uid（如果用户没有指定uid，那么返回空字符串""）	match [f] return euid(f);	
tolong	literal	转换为Long（如果可行）否则null		
toboolean	literal	转换为Boolean（如果可行）否则null		
todouble	literal	转换为Double（如果可行）否则null		
tolocaldatetime	literal	转换为localdatetime（如果可行）；否则报错，支持将毫秒时间戳转为localdatetime	return tolocaldatetime(16118182450561)	
tostring	literal	转换为String		
todecimal	literal,int,int	返回一定精度的数值（如果可行）；否则null。精度转换遵循HALF_UP四舍五入，默认精度为decimal(10,2)	todecimal(123.423,5,2); /// 返回123.42 todecimal(123.123); /// 返回123.12	

函数名	参数	功能	示例	备注
tofloat	literal	转换为float（如果可行）；否则null		
tointeger	literal	转换为integer（如果可行）；否则null		
tomillis	localdatetime	将localdatetime转换为毫秒时间戳(如果可行) 否则报错		
toarray	array literal,string	转换为Array格式，第二个参数表示array中的元素类型：支持double、int、long、string、boolean和localdatetime; toarray([1.2,2.3,3.5],"int")返回[1,2,3]		
to_timeseries	timeseries literal, string	转换成timeseries格式，第二个参数表示timeseries的原始类型：支持double、int、long、string和boolean		
add_day	localdatetime, Int	在指定的时间戳上加指定的天数		
add_hour	localdatetime, Int	在指定的时间戳上加指定的小时		
add_minute	localdatetime, Int	在指定的时间戳上加指定的分钟		
add_second	localdatetime, Int	在指定的时间戳上加指定的秒		
localdatetime	String, Literal	将特定格式的日期返回成localdatetime类型，目前支持"iso_local_date_time"、"basic_iso_date"等格式可以参考 **Java DateTimeFormatter** 对应的格式定义，也支持指定模式。如果为空值()或者null值会返回当前的系统时间。	1. localdatetime("20181011", "basic_iso_date") 2. localdatetime("2020-05-03T15:16:17", "iso_local_date_time") 3. localdatetime("2020-05-03T15:16:17") 默认返回iso_local_date_time 格式 4. localdatetime() 默认返回iso_local_date_time 格式下当前的系统时间 5. localdatetime("2022-12-01","yyyy-MM-dd") 指定yyyy-MM-dd模式	
timediff	localdatetime, localdatetime, Literal	返回两个时间戳的时间差，可指定单位为ns、ms、s、m、h或d	1. timediff(localdatetime("20181010", "basic_iso_date"), localdatetime("20181011","basic_iso_date"), "d") 2. timediff(localdatetime("2018-10-09T15:16:17", "iso_local_date_time"), localdatetime("20181011", "basic_iso_date"), "h")	

函数名	参数	功能	示例	备注
year	localdatetime	返回指定的时间戳中的年	return year(localdatetime());	
month	localdatetime	返回指定的时间戳中的月	return month(localdatetime());	
day	localdatetime	返回指定的时间戳中的日	return day(localdatetime());	
hour	localdatetime	返回指定的时间戳中的小时	return hour(localdatetime());	
minute	localdatetime	返回指定的时间戳中的分钟	return minute(localdatetime());	
second	localdatetime	返回指定的时间戳中的秒	return second(localdatetime());	
labels	node	返回点的所有类型（List）	match (a) return labels(a);	
mid_nodes	path	返回路径中的中间节点	match p=(a)-[f*2]→(b) return mid_nodes(p);	
nodes	path	返回路径中的所有点（List）	match p=(a)-[f]-(b) return nodes(p);	
relationships	path	返回路径中的所有边(List)	match p=(a)-[f]-(b) return relationships(p);	
length	path	返回路径的长度	match p=(a)-[f]-(b) return length(p);	
props	类型为点或边的实体	返回点或边的属性	match (a) return props(a);	
countapproxivertexs	/	快速查询图中点的数量	return countapproxivertexs();	
countapproxiedges	/	快速查询图中边的数量	return countapproxiedges();	
coalesce	(expr [, expr]*)	返回第一个非null的值，如果所有入参都是null则返回null	MATCH (a) WHERE a.idno = '12321321' RETURN coalesce(a.username, a.english_name);	
pt_distance	(point(lon,lat),point(lon,lat))	返回两个经纬度地理坐标的距离	match (n) return pt_distance(n.geoPoint,point(20.5,30.5));	
get_by_timestamp	timeseries,timestamp	返回一个集合，包含一个键值对，表示时序属性中对应时间点的值，如果匹配不到精确时间则会返回该时间点之前最近的值，如果该时间点之前没有值则返回空集合	match (n) return get_by_timestamp(to_timeseries(n.ts, "int"),localdatetime("2021-01-01T00:00:00"));	

函数名	参数	功能	示例	备注
get_by_timestamp	timeseries,timestamp,timestamp	返回一个键值对集合，包含时序属性中对应时间段内的所有值，如果该时间段内没有值则返回空集合	match (n) return get_by_timestamp(to_timeseries(n.ts,"int"),localdatetime("2021-01-01T00:00:00"),localdatetime("2022-01-01T00:00:00"));	

5.16.2. 聚合函数

函数名	参数	功能
count	structure	返回参数在结果集中的个数
min	values	返回参数在集合中的最小值
max	values	返回参数在集合中的最大值
sum	values	返回集合中数据的总和

collect	values	将多个同类型元素聚合在一个list中；支持输入一至四个参数：
		1.ARG需要聚合的元素；
		2.最多聚合元素的LIMIT数量。默认1000；
		3.聚合数量超出limit限制决定是抛异常还是静默。默认true，设成false静默不会超限报错；
		4.是否过滤null值，默认开启。当前版本不建议将LIMIT数量设为过大的值，同时强烈不建议关闭超限报错。
		例：
		match (a) return collect(uid(a));
		match (a) return collect(uid(a),10000);
		match (a) return collect(uid(a),10000,true);
		match (a) return collect(uid(a),10000,true,false);
		支持去重，在第一个参数前添加关键字"distinct"，例：match (a) return collect(distinct a,1000,true);

5.16.3. 数学函数

本表中三角函数皆为弧度制计算。

函数名	参数	功能
abs	double	绝对值
sign	double	负数返回-1，整数返回1，0返回0
sqrt	double	开平方
e	/	自然对数的底数e
exp	double	e的n次方
ln	double/int	自然对数
log10	double	以10为底对数
pi	/	圆周率 π
sin	double	正弦

tan	double	正切
cos	double	余弦
acos	double	反余弦
asin	double	反正弦
atan	double	反正切
degrees	double	输入弧度换算成角度制
radians	double	输入角度换算成弧度制
ceil/ceiling	double	大于等于参数的最小整数
floor	double	小于等于参数的最大整数
rand	/	产生一个0-1之间随机浮点数
pow	double,double	pow(x, y)返回x的y次方
round	double	返回四舍五入后的值

5.16.4. 字符串操作函数

函数名	参数	功能
left	String, Integer	left(y, x)返回字符串y最左边x个字符的子串（如果x大于字符串y的长度，则返回字符串y）
right	String, Integer	right(y, x)返回字符串y最右边x个字符的子串（如果x大于字符串y的长度，则返回字符串y）
ltrim	String	去除字符串左边的空格
rtrim	String	去除字符串右边的空格
trim	String	去除字符串左右两边的空格
replace	String(source), String(match), String(repl)	在source中将所有match替换为repl
substring/substr	String(source), Integer(index), Integer(length)	对第source从index开始取长度为length的子串
tolower	String	变为全小写
toupper	String	变为全大写
reverse	String	反转字符串
split	String,String	split(x, y)以字符串y为分隔符分割字符串x
split_regex	String,String	split_regex(x, y)以正则字符串y为分隔符分割字符串x
concat_string	String*	拼接所有的字符串参数
cruxregmatch	String, String	cruxregmatch(x, r) 验证字符串x是否满足正则r

hex_string	String	返回字符串的hex编码
length	String	返回字符串的长度

5.16.5. 集合操作函数

函数名	参数	功能
set_equals	Array, Array	两个集合是否相等（注意:如果输入Array中有重复字符，内部会转化为集合处理。比如set_equals(['a','b'],['a','a','b'])会返回true）
collection_contains	Array, Array	第一个集合是否包含第二个集合 （注意:如果输入Array中有重复字符，内部会转化为集合处理，比如collection_contains(['a','b'],['a','a','b'])会返回true）
unwind_udtf	Array	将一个数组拆解成多个元素，并逐行返回。当输入为null或者长度为零的Array时，返回零行结果（比如unwind_udtf([1,2,3])返回三行，分别是1,2,3）
crux_size	Array	返回数组的长度
add_list	Array[any],any	向一个数组添加元素
list_apply	Array[any],Int	返回list指定位置的元素。比如list_apply(['ab','2','we'],0)返回的是集合的第一个元素 'ab'
sort_list	Array[Any]	将list元素进行排序，比如输入return sort_list([3,1,2])，返回[1,2,3]

5.16.6. crux-cypher内部函数

函数名	参数	功能	备注
tags	node/relation	返回点或边的tags，是一个Array。比如 match(a) return tags(a); 返回所有点的tags	
split_path	path	将路径拆分成多个长度为一的路径，是一个Array。当p = (a)时，返回空Array。比如 p = (a)-[f]->(b)-[g]->(c)，那么split_path(p)是由(a)-[f]->(b)和(b)-[g]->(c)组成的Array	
tojson	node/relation/path	返回点、边或路径的JSON格式，是一个JSON串	
is_simple_path	path	返回过滤经过重复点（带环）的路径，比如match p = (a)-[f*2]→(b) where is_simple_path(p) return p;	
node_rk_to_uid	vertex	将算法运行时的内部ID映射回用户定义的userID	5.0.0版本不推荐使用，可用uid()函数替代

5.16.7. Reduce函数

Reduce 函数用于返回列表中的每个元素作用在表达式上所产生的最终值。其语法如下所示：

```
reduce(accumulator = initial , variable IN list | expression)
```

其中，参数 **accumulator** 用于保存结果和迭代列表时中间结果的变量；**initial** 用于初始化变量值，且只执行一次；**variable** 用于在迭代列表时，表示值的变量；**expression** 表达式将对列表中的每个值执行一次，并生成结果值，保存在accumulator中。需要注意的是，accumulator和expression的返回值类型必须相同。

使用示例和详解如下：

```
return reduce(i = 0 , j in [1,2,3] | i + j);
```

该示例中，accumulator为i且0作为i的初始值。variable为j，作为引用名称遍历数组中的元素。expression为i + j，那么第一次计算0 + 1 = 1，中间结果保存在i中，此时i = 1；第二次计算1 + 2 = 3，此时i = 3；第三次计算3 + 3 = 6，此时i = 6，返回6作为表达式结果。

5.16.8. 空间函数

空间函数是对地理坐标数据类型提供的函数，用于快速实现两坐标点距离求解、查询范围内是否存在某点等操作。

5.16.8.1. point()函数

point() 函数用于返回WGS-84坐标系中与给定坐标值相对应的2D点，该函数的返回值是double类型的数组Array[double]。语法如下：

```
point(longitude,latitude)
```

其中，longitude为WGS-84坐标系中的点的经度，其类型为double；latitude为WGS-84坐标系中的点的纬度，其类型也是double。

该函数使用示例如下：

1. 返回经纬度为（20.5678，30.7891）的点。

   ```
   return point(20.5678,30.7891) As point;
   ```

   返回结果如下图所示：

   ```
   +--------------------+
   |       point        |
   +--------------------+
   | [20.5678,30.7891]  |
   +--------------------+
   ```

2. 查询并返回点n，且n的属性a是经纬度（20.5678，30.7891）所指向的地理坐标类型的数据。

   ```
   match (n) where n.a = point(20.5678,30.7891) return n;
   ```

5.16.8.2. pt_distance()函数

pt_distance() 函数返回一个double类型的值。该值表示在WGS-84坐标系中两个点之间的测地距离，单位KM。函数语法如下：

```
pt_distance(point1,point2)
```

在该函数中，point1和point2均表示WGS-84坐标系中的点。

该函数的使用示例如下所示：

1. 假设有两个坐标点，经纬度分别为（20.7788，30.7788）和（20.5566，30.5566），返回两个地理坐标点之间的距离。

```
return pt_distance(point(20.7788,30.7788), point(20.5566,30.5566)) As distance;
```

语句执行结果如下图所示：

```
+--------------------+
|      distance      |
+--------------------+
| 32.589980777221506 |
+--------------------+
```

2. 接上例，在图中查找位于两个坐标的点n和m，并计算两个点之间的距离。

```
match (n) where n.point = point(20.7788,30.7788);
match (m) where m.point = point(20.5566,30.5566);
return pt_distance(n.point, m.point);
```

3. 假设已知point2为地理坐标类型数据，在当前图中查询点n，且n到point2之间的距离等于或者小于distance（distance可替换为具体数值）。

```
match (n) where pt_distance(n.point, point2) = distance return n;
或
match (n) where pt_distance(n.point, point2) < distance return n;
```

pt_distance(null, null)、pt_distance(null, point2)和pt_distance(point1, null)均会返回null。

5.16.8.3. withinBox()函数

withinBox() 函数用于判断目标查询点是否存在于由WGS-84坐标系中两个点组成的一个 **矩形边界框** 中，函数的返回值是一个布尔值。函数语法如下：

```
withinBox(boundPoint1,boundPoint2,targetPoint)
```

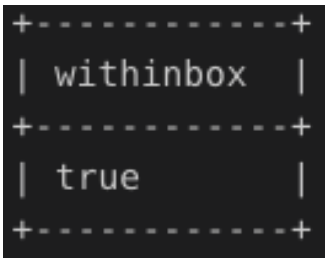
其中，参数boundPoint1和boundPoint2为同在WGS-84坐标系中的两个点，矩形范围由这两个点确定。targetPoint为目标查询点。

该函数使用示例如下所示：

1. 判断坐标点（20.60，30.60）是否在坐标点（20.7788，30.7788）和坐标点（20.5566，30.5566）所形成的矩形范围内。

```
return withinBox(point(20.7788,30.7788), point(20.5566,30.5566), point(20.60,30.60)) As
withinBox;
```

该例句的执行结果如下图所示：



2. 假设已知两坐标点boundPoint1和boundPoint2，在图中查询并返回在两个坐标点所形成的矩形范围内的点。

```
match (n) where withinBox(boundPoint1,boundPoint2,n.point) return n;
```

5.16.8.4. withinCircle()函数

withinCircle() 用于判定目标点是否存在于由WGS-84坐标系中一个坐标点作为圆心形成的一个 **圆形边界框** 中，函数的返回值是一个布尔值。 函数语法如下：

```
withinCircle(center,radius,targetPoint)
```

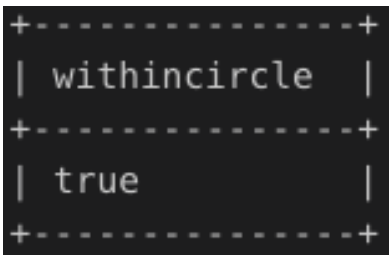
其中，函数中的参数center表示WGS-84坐标系中作为圆心的坐标点；radius表示圆形边界的半径，单位是KM，数据类型可为double或是int类型；targetPoint表示目标坐标点。

函数使用示例如下所示：

1. 判断坐标点（20.60，30.60）是否存在于以坐标点（20.7788，30.7788）为中心，半径为50KM的圆形范围内。

```
return withinCircle(point(20.7788,30.7788), 50.0, point(20.60,30.60)) As withinCircle;
```

该例句执行结果如下图所示：



2. 假设已知坐标中心坐标点center，半径为radius。在图中查询并返回以center为圆心，半径为radius的圆形范围内的点。

```
match (n) where withinCircle(center,radius,n.point) return n;
```

5.17. 断言函数

下述断言函数中，nodes(p)返回路径中所有的点的列表。

5.17.1. all()函数

all() 用于断言predicate是否适用于列表list中的所有元素，如果是则返回true；否则返回false。如果list为空返回true。函数语法如下所示：

```
all(variable IN list WHERE predicate)
```

其中，参数list表示列表或者返回列表的表达式；variable表示用于断言中的变量；predicate用于测试列表中所有元素的断言。

该函数使用示例如下：返回的路径中所有的点的age属性都大于18的路径。

```
match p=(a)-[f]->(b) where all(n in nodes(p) where n.age > 18) return p;
```

5.17.2. any()函数

any() 用于断言predicate是否至少适用于列表list中的一个元素，如果是则返回true；否则返回false。如果list为空返回false。函数语法如下所示：

```
any(variable IN list WHERE predicate)
```

其中，参数list表示列表或者返回列表的表达式；variable表示用于断言中的变量；predicate用于测试列表中所有元素的断言。

该函数使用示例如下：返回的路径中至少有一个点的age属性大于18。

```
match p=(a)-[f]->(b) where any(n in nodes(p) where n.age > 18) return p;
```

5.17.3. single()函数

single() 用于断言predicate是否只适用于列表list中的某一个元素，如果是则返回true；否则返回false。如果list为空返回false。函数语法如下所示：

```
single(variable IN list WHERE predicate)
```

其中，参数list表示列表或者返回列表的表达式；variable表示用于断言中的变量；predicate用于测试列表中所有元素的断言。

该函数使用方法如下所示：返回的路径中只有一个点的age属性大于18。

```
match p=(a)-[f]->(b) where single(n in nodes(p) where n.age > 18) return p;
```

5.17.4. none()函数

none() 用于断言predicate是否不适用于列表list中的任何元素，如果是则返回true；否则返回false。如

果list为空返回true。函数语法如下所示：

```
none(variable IN list WHERE predicate)
```

其中，参数list表示列表或者返回列表的表达式；variable表示用于断言中的变量；predicate用于测试列表中所有元素的断言。

该函数使用示例如下所示：返回的路径中所有的点的age属性都不大于18。

```
match p=(a)-[f]->(b) where none(n in nodes(p) where n.age > 18) return p;
```

5.18. 索引（新）

索引是数据库中某些数据的冗余副本，目的是使查询性能更优。作为代价，数据库需要额外存储空间和较慢写入速度，因此决定哪些字段需要索引是一项重要且不易的任务。

（新）StellarDB 5.0.0版本不再对旧版本使用的 `manipulate create_index` 和 `manipulate delete_index` 语法进行支持，在新版本中统一使用 `create index` 和 `drop index` 进行索引的创建和删除

5.18.1. 新增索引

```
CREATE INDEX [IF NOT EXISTS] FOR (LabelName) ON [f1, f2, ...];
CREATE INDEX [IF NOT EXISTS] FOR [LabelName] ON [f1, f2, ...];
```

- 不支持对 `TIME_SERIES` 类型的属性创建索引
- 默认情况下，对同一个Label的某个属性多次创建索引会报错；但如果带有 `IF NOT EXISTS`，则不会抛出任何错误
- 包裹点边 `LabelName` 的括号不同，注意区分

示例 1. 在点label `person` 的属性 `name` 和 `age` 上建立索引

```
CREATE INDEX IF NOT EXISTS FOR (person) ON [name, age];
```

示例 2. 在边label `ask` 的属性 `time` 上建立索引

```
CREATE INDEX IF NOT EXISTS FOR [ask] ON [time];
```

5.18.2. 删除索引

```
DROP INDEX FOR (LabelName) ON [f1, f2, ...] [IF EXISTS];
DROP INDEX FOR [LabelName] ON [f1, f2, ...] [IF EXISTS];
```

- 默认情况下，对某个Label没有被索引的属性进行删除索引的操作会报错；但如果带有 `IF EXISTS`，则不会抛出任何错误
- 包裹点边 `LabelName` 的括号不同，注意区分

示例 1. 删除点label `person` 的属性 `name` 上存在的索引

```
DROP INDEX FOR (person) ON [name] IF EXISTS;
```

示例 2. 删除边label `ask` 的属性 `time` 上存在的索引

```
DROP INDEX FOR [ask] ON [time] IF EXISTS;
```

5.18.3. 查看索引

```
USE GRAPH <GraphName>
SHOW INDEX;
SHOW INDEX FOR <LabelName>
```

5.18.4. 索引重建

StellarDB在对数据进行更新操作时，会在索引中慢慢积攒冗余信息。在长期运行过程中，冗余信息会对查询性能产生负面影响。进行索引重建可以从索引中去除冗余，提升查询性能。

索引重建任务在后台执行。任务的调度、执行过程在各shard各副本是独立的，同一时刻各shard的任务状态可能不同。

索引重建任务的执行过程主要分为两个阶段：前一阶段进行索引重建，时间较长，阶段中支持正常的读写，但不可与“图备份”功能、“批量导入”功能、“schema更新”功能同时执行； 后一阶段进行索引的切换，持续时间很短（最多持续数秒），阶段中的读写请求将会被阻塞，直到阶段完成再执行。

5.18.4.1. 提交索引重建任务

在beeline客户端中执行如下命令提交索引重建任务。StellarDB会尝试在指定shard提交索引重建任务，任务会在稍后被调度执行。

索引重建语法规则如下所示：

```
manipulate graph <graphName> rebuild_index <type> <shardIdList>;
```

其中：

- graphName，为需重建索引的图的图名。
- type，为重构索引数据范围，可选值包括 `vertex`、`edge` 和 `both`，分别表示重建点数据索引、重建边数据索引，重建两者的索引。
- shardIdList，为需重建的 `shardId` 列表。也可使用 `all` 值表示同时启动所有shard的重建索引任务。

调用示例如下：

```
manipulate graph g1 rebuild_index both 0,1,2;
manipulate graph g1 rebuild_index edge all;
```

返回值示例：

```
{
        "jobIdPairs":[
                {
                        "jobId":"RebuildIndex_g1_2_2020-12-15T03:14:00.910Z_43",
                        "shardId":2
                },
                {
                        "jobId":"RebuildIndex_g1_1_2020-12-15T03:14:00.857Z_52",
                        "shardId":1
                },
                {
                        "jobId":"RebuildIndex_g1_0_2020-12-15T03:14:00.831Z_51",
                        "shardId":0
                }
        ],
        "message":"success",
        "status":0,
        "failedShards":[]
 }
```


其中：

- status，用于表示各shard任务提交成功情况，全部成功/部分成功/全部失败时值分别为0/-1/-2.
- jobIdPairs，用于包含提交成功的任务信息。每个shard的任务具有单独的任务ID。
- failedShards，用于包含提交失败的shard的ID。

索引重建注意事项：

1. 同时启动所有shard的索引重建任务可能会使集群同时运行的索引重建任务过多，消耗较多计算资源，对读写请求的处理性能下降。建议仅在系统空闲时同时启动所有shard的索引重建任务；若需要在重建索引时使用系统的读写能力，建议对每次仅对一部分shard启动索引重建任务，检查任务执行完成后再进行其他shard的索引重建。
2. StellarDB会限制重建索引任务占用的计算资源来避免其对数据库一般服务产生过大影响，但也会使并发执行中的的重建任务过多时无法提交更多的重建任务。如果有特别的重建索引需求，可以调节StellarDB配置项 `graph.db.rebuild.index.threads`(默认为32），将数值增大则StellarDB可以同时执行更高并发的重建索引任务。

5.18.4.2. 查看索引重建任务状态

在beeline客户端中执行如下命令查看重建任务执行情况。

查看索引重建任务语法规则如下：

```
manipulate graph <graphName> show rebuild_index <shardId> <jobId>;
```

其中：

- graphName，为重建索引的图的图名。
- shardId，为重建索引任务所在shard的ID。
- jobId，为提交重建索引任务时返回的任务ID。

调用示例如下：

```
manipulate graph g1 show rebuild_index 0 RebuildIndex_g1_0_2020-12-15T03:14:00.831Z_51;
```

返回值示例：

```
{
        "validTasks":[
                {
                        "worker":"akka.tcp://storageWorker@host1:33815/user/GraphStorageWorker",
                        "status":{
                                "buildVertex":true,                     // 任务是否包含点数据的索引重建
                                "jobId":"RebuildIndex_g1_2_2020-12-15T03:14:00.910Z_43",    // 任务ID
                                "creationTime":1608002047187,           // 任务在本shard副本的启动时间戳
                                "vertexProgress":1.0,                   // 点数据索引重建进度
                                "state":3,                              // 任务执行状态
                                "buildEdge":true,                       // 任务是否包含边数据的索引重建
                                "edgeProgress":1.0                      // 边数据索引重建进度
                        }
                },
                {
                        "worker":"akka.tcp://storageWorker@host2:33815/user/GraphStorageWorker",
                        "status":{
                                "buildVertex":true,
                                "jobId":"RebuildIndex_g1_2_2020-12-15T03:14:00.910Z_43",
                                "creationTime":1608002040935,
                                "vertexProgress":1.0,
                                "state":1,
                                "buildEdge":true,
                                "edgeProgress":0.75
                        }
                },
                {
                        "worker":"akka.tcp://storageWorker@host3:33815/user/GraphStorageWorker",
                        "status":{
                                "buildVertex":true,
                                "jobId":"RebuildIndex_g1_2_2020-12-15T03:14:00.910Z_43",
                                "creationTime":1608002052141,
                                "vertexProgress":1.0,
                                "state":3,
                                "buildEdge":true,
                                "edgeProgress":1.0
                        }
                }
        ],
        "message":"success",
        "missedWorkers":[]
 }
```

返回结果中：

- 每个shard副本会分别返回自己的任务状态。如果无法从副本上查询到jobId对应的重建任务（可能是任务尚未被调度执行，或者任务已经经过了3天以上，历史记录被清除），则在 `missedWorkers` 中会包含shard副本Worker的地址。可查询到任务状态的都包含在 `validTasks` 中。
- 每个shard副本的任务返回状态如上所示，其中的state值表示任务执行状态：

  0 - 任务初始化中

  1 - 索引重建中（此状态中，进度才会处于0到1之间）

  2 - 切换索引中

  3 - 任务成功完成

  -1 - 任务等待shard其他互斥任务结束时间过长，自动取消索引重建任务，安全退出

  -2 - 任务异常退出

5.18.4.3. StellarDB相关配置项

StellarDB服务的以下配置项可影响重建索引过程行为：

1. `graph.rebuild.index.progress.track.enabled` 可以设置StellarDB是否应追踪索引重建阶段的进度，默认值为true，即启用追踪进度。若设为false， 将会使重建索引速度提升，但是在查询任务进度时，`vertexProgress` 与 `edgeProgress` 的值将不会以小数反映具体进度，值只会在阶段完成

时从0.0变为1.0。

2. `graph.rebuild.index.wait.start.max.time` 可以设置索引重建任务被调度执行时，等待目前的批量导入任务完成的最长等待时间。默认值为60000，即60秒。 索引重建不可与批量导入同时执行，但可以等待批量导入任务完成。如果系统频繁进行批量导入操作，且批量导入任务重要性较高，可以将本值设低，使重建索引任务更容易 等待超时、快速自行取消。

5.19. 全文索引

StellarDB全文索引基于 Apache Lucene 开发，可用来索引点和边中 **String** 类型的属性。允许使用被索引字段的部分内容进行查询。

5.19.1. 查看分词器

全文索引利用指定的分词器将一段字符串切分成多个词组(分词)，因此可以匹配到多个子字符串。分词的效果由用户指定的分词器决定。

可以使用以下语句查看系统支持的分词器类型：

```
SHOW FULLTEXT ANALYZER;
```

5.19.2. 创建全文索引

目前创建全文索引有两种方法：独立地为点或边的属性创建索引；或者在创建图的时候，在schema中指定创建索引。全文索引名只支持大小写字母、阿拉伯数字和下划线，并且在同一张图中全文索引名不可重复。

- 使用创建语句独立地对点或边创建索引，语法如下所示。

  a. 在点属性上创建全文索引。

  ```
  CREATE FULLTEXT INDEX [IF NOT EXISTS] <index_name> FOR (nodeLabel_1|nodeLabel_2|...) ON
  [propName1, propName2,...] [OPTIONAL "{" ANALYZER: analyzer_name "}"];
  ```

  b. 在边属性上创建全文索引。

  ```
  CREATE FULLTEXT INDEX [IF NOT EXISTS] <index_name> FOR [edgeLabel_1|edgeLabel_2|...] ON
  [propName1, propName2,...] [OPTIONAL "{" ANALYZER: analyzer_name "}"];
  ```

  创建全文索引时需要注意以下事项：

  `OPTIONAL` 为可选项，当前可选项为分词器类型。默认分词器为 `smart_cn`，可根据系统支持的分词器类型指定。

  目前不支持单个索引名对label指定field建全文索引。例如，label Movie 和Book有同名属性name，在同一个索引名下，不支持只索引Movie.name而不索引Book.name，同名属性均会被索引。

  使用该方法创建全文索引示例如下：

  a. 在点属性上创建全文索引：在点属性movie.(name, director)和person.name上建立名为ft_v的全文索引，默认分词器为smart_cn。

  ```
  CREATE FULLTEXT INDEX IF NOT EXISTS ft_v FOR (movie|person) ON [name, director];
  ```

  b. 在边属性上创建全文索引：在边属性acted_in.place上建立名为ft_e的全文索引，指定分词器cjk。

  ```
  CREATE FULLTEXT INDEX IF NOT EXISTS ft_e FOR [acted_in] ON [place] OPTIONAL {ANALYZER: cjk};
  ```

- 建图时在schema中指定创建全文索引。

```
CREATE GRAPH <graph_name> WITH JSON SCHEMA
'{
  "edge.tables": [...],
  "vertex.tables": [...],

  "fulltext.index.schema": {
    "vertex.fulltext.descriptors": [
      {
        "index.name": <index_name>,
        "index.analyzer": <analyzer>,
        "indexed.labels": <labels>,
        "indexed.fields": <fields>
      }
    ],
    "edge.fulltext.descriptors": [
      {
        "index.name": <index_name>,
        "index.analyzer": <analyzer>,
        "indexed.labels": <labels>,
        "indexed.fields": <fields>
      }
    ]
  }
}';
```

使用该方法创建全文索引示例如下：

```
CREATE GRAPH movies WITH JSON SCHEMA
'{
  "graph.cacheable": false,
  "graph.index.type": "NATIVE",
  "graph.name": "movies",
  "graph.replication.number": 3,
  "graph.shard.number": 10,
  "edge.tables": [
    {
      "compress.policy": "global",
      "field.schemas": [
        {
          "field.name": "place",
          "field.type": "STRING"
        }
      ],
      "label.value": "acted_in"
    }
  ],
  "vertex.tables": [
    {
      "compress.policy": "global",
      "field.schemas": [
        {
          "field.name": "name",
          "field.type": "STRING"
        },
        {
          "field.name": "director",
          "field.type": "STRING"
        }
      ],
      "label.value": "movie"
    },
    {
      "compress.policy": "global",
      "field.schemas": [
        {
          "field.name": "name",
          "field.type": "STRING"
        }
      ],
      "label.value": "person"
    }
  ],
  "fulltext.index.schema": {
    "vertex.fulltext.descriptors": [
      {
        "index.name": "ft_v",
        "index.analyzer": "smart_cn",
        "indexed.labels": [movie, person],
        "indexed.fields": [name, director]
      }
    ],
    "edge.fulltext.descriptors": [
      {
        "index.name": "ft_e",
        "index.analyzer": "cjk",
        "indexed.labels": [acted_in],
        "indexed.fields": [place]
      }
    ]
  }
}';
```

5.19.3. 指定图名查看全文索引

```
USE GRAPH <graph_name>;
SHOW FULLTEXT INDEX;
```

5.19.4. 查询全文索引

全文索引查询分为常规查询和基于Lucene语法的查询。

- 常规查询。

```
MATCH (n) WHERE FULLTEXT_QUERY("index_name", "query_string") RETURN n;
MATCH [f] WHERE FULLTEXT_QUERY("index_name", "query_string") RETURN f;
```

- 基于Lucene语法查询。

```
MATCH (n) WHERE FULLTEXT_QUERY("index_name", 'query_string') RETURN n; -- 单引号''包裹查询字符串
```

使用全文索引进行近似查询和精确查询的示例如下图所示：

- 字符串 ″克里斯″ 被分解为［克，里，斯］，包含这三个term的字符串都会被命中。若想要精确命中单条字符串，需要用单引号包裹字符串。
- 更多Lucene查询语法详见 Lucene query document 。

5.19.5. 删除全文索引

```
DROP FULLTEXT INDEX <index_name> [IF EXISTS];
```

5.20. 查看系统内置函数（新）

StellarDB 5.0.0提供简单的语法查看系统内置的函数。

1. 查看所有内置函数

```
show functions;
```

2. 通配符匹配查找函数，会匹配函数名，函数类型

```
## 尝试匹配split函数名的函数
show functions like "*split*";

## 函数匹配输入输出类型为array[node]
show functions like "array[node]";

## 匹配udaf
show functions like "udaf";
```

5.21. StellarDB 5.0.0 动态图(Dynamic Graph)模型（新）

5.21.1. 为什么引入动态图模型?

在实际应用过程中很容易可以发现，图数据在很多图数据的应用场景中并不是静态不变的，而是动态演进的，这些场景中包括例如金融反欺诈场景中金融交易网络随着时间的推进而发生的交易变化、交易社群变化等；又比如社交网络中新增用户、用户关注或者取消关注、更改账户信息等。

将图数据变化的历史记录下来，不仅可以用于历史数据规律的总结，还可以利用动态图数据进行动态图神经网络相关技术的研究，从而进一步挖掘数据中潜在的数据价值和更加灵活高效的业务场景，譬如预测某一个时刻某一事件是否会发生。

5.21.2. 动态图模型的动态变化

图数据的动态变化主要分为两类，一类是节点或边的属性的值的变化；另一类变化是子图（结构）的变化，如新增/删除点边。这两种图数据的动态变化可以单独发生，也可以同时发生。

从图数据的属性变化角度来看，StellarDB 5.0.0动态图模型可以记录图中节点或者边属性的所有历史版本（而非新数据覆盖旧数据）。在实际数据开发使用中，还可以结合诸如柱状图、趋势图等对历史数据进行可视化，更加直观、更加适合业务使用。

从图数据的子图（结构）的角度来看，StellarDB 5.0.0 动态图模型还可以返回不同时间子图的变化。

5.21.3. 创建动态图模型

由于动态图模型隐式地存储了图数据中相关数据的历史记录，那么很自然的，我们需要在创建图schema的过程中，显式地声明当前动态图需要记录的数据记录的版本个数，只需要在创建图schema的过程中，在graphproperties属性中配置 `graph.reserve.version: N` 即可，N为大于等于1的正整数，当N为1时，该图为普通图；当N大于1时，创建的图就是动态图，且途中会保留相应数据的N个历史版本。

5.21.4. 动态图模型的查询

目前，StellarDB 5.0.0主要支持图历史数据的查询，通常搭配下面两个内置函数使用：

- history(x, y)：返回x的y个版本的数据，其中x可表示点、边
- history(x)：返回x的所有历史数据，其中x可表示点、边

使用示例：

```
MATCH (a) WHERE uid(a)="xxx" RETURN history(a, x);
MATCH (a)-[f]->(b) WHERE uid(a) = "abc" RETURN history(f);
```

5.21.5. 动态图的其他操作

动态图的其他操作和普通图并无区别，请参考手册其他章节内容。

6. StellarDB图算法

6.1. 图计算

StellarDB 5.0.0版本对图算法场景进行了大规模改进和提升，内置算法性能得到较大提升。在语法方面，StellarDB 5.0.0 的内置图算法对于返回的节点，会直接以节点类型返回。因此可以直接使用uid(vertex)访问节点的uid，而不再需要node_rk_to_uid函数进行uid的转换。可以参考PageRank等函数。

另外，对于图算法返回的节点，我们也可以灵活的访问其其他属性作为返回值。

6.1.1. 图计算简介

StellarDB的图计算使用TEoC语句调用相应图算法。算法的输入数据为图的点、边数据。当前版本中图计算支持结果返回、结果导出和结果写回。在使用图算法时，使用 `config crux.execution.mode analysis;` 语句切换到分析模式下使用图算法语句。

6.1.2. 图数据视图

StellarDB支持创建一个可被持久化的视图，用于加速图算法执行过程。

1. 创建视图

   创建视图的语法如下所示：

   ```
   create query temporary graph view GRAPH_VIEW_NAME as (v) [e] with GRAPH_ALGO(@GRAPH_VIEW_NAME,
   VIEW_STORE_PATH, CONFIG_MAP);
   ```

   a. GRAPH_VIEW_NAME：视图名，由用户指定。

   b. 获取子图的点数据：

      i. 使用 `(v)` 获取完整的点数据；

      ii. 或使用 `[v collection_contains(['V1','V2'],labels(v))]` 过滤指定label的点数据；

      iii. 或使用 `(v where v.prop > 10)` 根据过滤条件获取点数据。

   c. 获取子图的边数据：

      i. 使用 `[e]` 获取完整的边数据；

      ii. 或使用 `[e collection_contains(['E1','E2'],labels(e))]` 过滤指定label的边数据；

      iii. 或使用 `[e where e.prop > 10]` 根据过滤条件获取边数据。

   d. GRAPH_ALGO：算法名，由用户指定。

   e. VIEW_STORE_PATH：视图持久化目录。

   f. CONFIG_MAP：JSON格式配置参数，分为通用参数和算法特定参数，其中通用参数有：

      i. directed，用于控制是否使用有向图数据，默认为 **true**

      ii. reuseGraph，用于控制是否复用 **VIEW_STORE_PATH** 的视图数据，默认为 **true**

iii. clearGraph，用于控制是否在算法运行结束后清除 **VIEW_STORE_PATH** 的视图数据，默认为 **false**

iv. edgeWeightProperty，用于控制算法使用的 **edge weight** 对应字段，通过"lable1:attr1,label2:attr2"为不同lable指定不同的属性值，默认情况下所有边的 **edge weight** 为1

v. limit，用于控制算法返回数据条数，默认为-1

算法返回后可以返回额外属性，如下示例，该语句执行PageRank算法，并返回点的p1属性：

+

```
create query temporary graph view sample_1 as (v) [e] with graph_pagerank(@sample_1,
"/tmp/view_1", '{factor: 0.85, rounds: 10}') as unapply(vertex, rank) return
uid(vertex), rank, vertex.p1;
```

VIEW_STORE_PATH 存储的是单个图的持久化视图，若切换图执行算法，请重新设置 **VIEW_STORE_PATH** 为其他目录，或配置 `config` 属性 **reuseGraph** 为 **false**。

2. 预建视图

执行算法前，可通过 **graph_warmup** 语句提前创建视图。在算法执行时，如果发现VIEW_STORE_PATH目录下不存在视图，会自动创建，但执行时间也会相应增加。

```
create query temporary graph view GRAPH_VIEW_NAME as (a) [b] with graph_warmup(@GRAPH_VIEW_NAME,
VIEW_STORE_PATH, CONFIG_MAP) as unapply(result) return result;
```

CONFIG_MAP配置参数：

a. genDirected:true，控制是否生成有向图数据，默认true 可关闭

b. genUndirected:true，控制是否生成无向图数据，默认true 可关闭

c. edgeWeightProperty:""，指定生成视图过程中边上的权重属性，默认为空不生成权重属性

d. edgeDefaultWeight:1.0，指定默认的权重值，当前指定权重属性为空值时使用

e. （内部）baseRDD:true，当该参数为true时，warmup使用低版本格式；如果该参数为false，warmup使用新版本warmup格式

6.1.3. 在子图中执行图算法

图算法支持通过TEoC语句获取一个子图（包含原图）执行图算法，且当前允许对点数据和边数据做过滤，如下示例：

- 筛选标签为X1和X2的点、f1值大于10的边参与PageRank计算：

```
create query temporary graph view algo_test_view_1 as (a where
collection_contains(['X1','X2'],labels(a)) ) [b where b.f1 > 10] with
graph_pagerank(@algo_test_view_1, "/tmp/algo_test_view_1", '{factor: 0.85, rounds: 10}')
 as unapply(vertex, rank) return uid(vertex), rank;
```

6.1.4. 基本度量(Network Measurements)

基本度量(Network Measurements)类算法提供图半径、直径、度、聚集系数等经典度量算法，可用于探索图的基本性质。

6.1.4.1. 图直径(Diameter)

1. 算法简介 点的离心率指该点能到达的最远的点之间的最短距离。图直径返回离心率最大的点的离心率。本算法结果为估计值，适用于大规模图的直径计算。图直径算法常用来衡量图中点点间最短距离的最坏情况。
2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_diameter(@sample_1,
"/tmp/view_1") as unapply(diameter) return diameter;
```

3. CONFIG参数说明：无参数配置
4. 返回字段 diameter，即图直径，数据类型为 int 类型。

6.1.4.2. 图半径(Radius)

1. 算法简介 点的离心率指该点能到达的最远的点之间的最短距离。图半径返回离心率最小的点的离心率。本算法结果为估计值，适用于大规模图的半径计算。图直径算法常用来衡量图中点点间最短距离的最优情况。
2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_radius(@sample_1,
"/tmp/view_1")
as unapply(radius) return radius;
```

3. CONFIG参数说明：无参数配置
4. 返回字段 radius，即图半径，数据类型为 int 类型

6.1.4.3. 度(Degree)

1. 算法简介 出入度中心度算法用来计算每个点的出入度，从而找到度最大或最小的点达到衡量点中心度的意义。Degree算法常用来找出图中最受欢迎的点。
2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_degree(@sample_1,
"/tmp/view_1",
'{direction: "out"}') as unapply(vertex, degree) uid(vertex), degree;
```

3. CONFIG参数说明
   a. direction: "out" | "in" | "both"，用于指定统计内容为出度、入度或出入度
4. 返回字段
   a. vertex，即点类型
   b. degree，度，数据类型为 int 类型

6.1.4.4. K-Core

1. 算法简介 K-Core 算法用来在图中找出指定核心度的子图结构，在结果子图中每个顶点至少具有 k 的度数并且所有顶点都至少与该子图中的 k 个其他节点相连。K-Core 算法用来在图中过滤出想对重要的点及关系结构，之后进行相应的分析处理。K-Core 算法由于其线性的时间复杂度和符合直观认识的可解释性，在风控金融，社交网络和生物学上都具有较多的应用场景。
2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_kcore(@sample_1, "/tmp/view_1",
'{limit: 10}') as unapply(vertex, cores) return uid(vertex), cores;
```

3. CONFIG参数说明：无参数配置
4. 返回字段
   a. vertex，即点类型
   b. cores，即 KCore 值，数据类型为`long`类型

6.1.4.5. 三角计数(Triangle Counting)

1. 算法简介 三角计数算法用来计算图中点包含的三角形的个数，能够反映图结构的关联程度，三角形越多组织关系越严密。三角计数算法是许多复杂网络分析的基础算法，适用于衡量图的结构特性场景，常用于社交网络结构分析，垃圾邮件分析和欺诈检测。
2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_triangle_count(@sample_1,
"/tmp/view_1", '{directed: false, limit: 10}') as unapply(vertex, triangles) return uid(vertex),
triangles;
```

3. CONFIG参数说明：无参数配置
4. 返回字段
   a. vertex，即点类型
   b. triangles，即计算所得三角形数量，数据类型为 long 类型

6.1.4.6. 局部聚集系数(Local Clustering Coefficient)

1. 算法简介 集聚系数算法计算节点间的集聚程度，局部集聚系数则用来计算每个节点与其周边节点的集聚程度，局部集聚系数越高的节点与其周边节点形成社区组织的倾向越高。其计算公式为：

$$C_i = \frac{2T_i}{d_i(d_i-1)}$$

其中，T_i 表示顶点 i 的三角形个数，d_i 表示顶点 i 的度。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with
graph_local_cluster_coefficient(@sample_1, "/tmp/view_1", '{directed: false, limit: 10}') as
unapply(vertex, coefficient) return uid(vertex), coefficient;
```

3. CONFIG参数说明：无参数配置
4. 返回字段
   a. vertex，即点类型
   b. coefficient，即coefficient值，数据类型为 double 类型

6.1.4.7. 平均聚集系数(Average Clustering Coefficient)

1. 算法简介 集聚系数算法计算节点间的集聚程度，平均集聚系数算法计算图中所有节点的集聚系数的平均值，平均集聚系数用来衡量图整体的集聚程度。其计算公式为：

$$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$$

其中，n 表示图中顶点的个数，C_i 表示顶点 i 的局部集聚系数。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with
graph_average_cluster_coefficient(@sample_1, "/tmp/view_1", '{directed:false}') as
unapply(coefficient) return coefficient;
```

3. CONFIG参数说明：无参数配置
4. 返回字段 coefficient，即coefficient值，数据类型为 double 类型

6.1.5. 中心性算法(Centrality)

中心性算法可用于度量网络中各个节点的重要性，常见于网页排名、社会网络分析等领域。

6.1.5.1. PageRank

1. 算法简介 PageRank 算法计算网络中点的相关性和重要性，由Google公司发明，常见用途是网页排名。其计算公式为：

$$PR_i(v) = \frac{1-d}{|V|} + d \cdot \sum_{u \in N_{in}(v)} \frac{PR_{i-1}(u)}{N_{out}(u)} + \frac{d}{|V|} \cdot \sum PR_{i-1}(u)$$

其中，d 为阻尼系数，$|V|$ 为节点总数，N_{in} 和 N_{out} 代表节点入、出邻居。
公式最后一项为对悬挂点问题的调和项，能防止在悬挂点处损失 PR 值。
第 i 轮迭代中，节点 $PageRank$ 的值 PR_i 由 PR_{i-1} 计算得出

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_pagerank(@sample_1,
"/tmp/view_1", '{factor: 0.85, rounds: 10, limit: 10}') as unapply(vertex, rank) return
uid(vertex), rank;
```

3. CONFIG参数说明
   a. factor: 0.85，用于配置 damping_factor，默认为0.85
   b. rounds: 10，用于设置算法循环次数，默认为10
   c. tolerance: 0.0000001，用于设置算法迭代精度控制，默认为0.0000001
   d. initial: 0.2，用于指定初始 rank 值，默认为0，算法会使用 1/(vertex number) 作为初始值
4. 返回字段
   a. vertex，即点类型
   b. rank，即PageRank值，数据类型为 double 类型

6.1.5.2. Personalized PageRank

1. 算法简介 Personalized PageRank 算法是 PageRank 的一种变体，计算对指定节点集合而言"个性化"的网络排名。Personalized PageRank 常用于推荐系统等领域。其计算公式为：

$$PR_i(v) = (1-d)r_v + d \cdot \sum_{u \in N_{in}(v)} \frac{PR_{i-1}(u)}{N_{out}(u)},$$

$$r_v = \begin{cases} 1, & v \in sourceNodes \\ 0, & v \notin sourceNodes \end{cases}$$

其中，d 为阻尼系数，N_{in} 和 N_{out} 代表节点入、出邻居。sourceNodes 为用户指定的个性化点集，是 $PersonalizedPageRank$ 算法中用户“感兴趣”的节点。
算法每轮以 $1-d$ 的概率随机跳跃时，只会跳到 $sourceNodes$ 中的点上。

2. 调用示例

```
decl sourceNodes:list[bytes];

sourceNodes:= match (a) where uid(a) in ["1","2","3","4","5"] return collect(id(a));

create query temporary graph view sample_1 as (v) [e] with
graph_personalized_pagerank(@sample_1, "/tmp/view_1", @sourceNodes, '{factor: 0.85, rounds: 10,
limit: 10}') as unapply(vertex, rank) return uid(vertex), rank;
```

3. CONFIG参数说明
   a. factor: 0.85，用于指定 damping_factor，默认为0.85
   b. rounds: 10，用于设置算法循环次数，默认为10
4. 返回字段
   a. vertex，即点类型
   b. rank，即PageRank值，数据类型为 double 类型

6.1.5.3. ArticleRank

1. 算法简介 ArticleRank 算法是 PageRank 的一种变体，常用于为引文网络（citation network）中的论文进行评分，使用的图模型为无边权重的有向图。经典 PageRank 算法用在引文网络中，会使得引用数量少的文章贡献出更多 PR 值，不符合实际情况。ArticleRank 对 PageRank 公式进行微调，通过在分母上引入平均出度，降低了低出度节点的影响力，从而更加符合引文网络的计算场景。其计算公式为：

$$AR_i(v) = (1-d) + d \cdot \sum_{u \in N_{in}(v)} \frac{AR_{i-1}(u)}{|N_{out}(u)| + \overline{N_{out}}}$$

其中，d 为阻尼系数，N_{in} 和 N_{out} 代表节点入、出邻居，$\overline{N_{out}}$ 是所有节点的平均出度。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_articlerank(@sample_1,
"/tmp/view_1", '{factor: 0.85, rounds: 10, limit: 10}') as unapply(vertex, rank) return
uid(vertex), rank;
```

3. CONFIG参数说明
   a. factor: 0.85，用于设置 damping_factor，默认为0.85
   b. rounds: 10，用于设置算法循环次数，默认为10
4. 返回字段
   a. vertex，即点类型
   b. rank，即PageRank值，数据类型为 double 类型

6.1.5.4. Weighted PageRank

1. 算法简介 Weighted PageRank 算法在 PageRank 的基础上进行调整。常见的 Weighted PageRank 算法有两种：

   模式1: 适用于有权图，算法将边上的权重作为系数，加入到PageRank的PR值计算过程中。其计算公式为：

$$PR_i(v) = \frac{1-d}{|V|} + d \cdot \sum_{u \in Nin(v)} \frac{W_{u,v}}{W_{out}(u)} \cdot PR_{i-1}(u) + \frac{d}{|V|} \cdot \sum PR_{i-1}(u)$$

其中，d 为阻尼系数，$|V|$ 为节点总数，$W_{u,v}$ 为边 (u,v) 上的权重，$W_{out}(u)$ 代表节点 u 出边上的权重值之和。公式最后一项为对悬挂点问题的调和项，能防止在悬挂点处损失 WPR 值。

模式2:
算法与边上的权重无关。算法考虑节点的重要性，越"重要"的节点会获得越大比例的PR值。其计算公式为：

$$PR_i(v) = (1-d) + d \cdot \sum_{u \in N_{in}(v)} PR_{i-1}(u) W^{in}_{(u,v)} W^{out}_{(u,v)}$$

$W^{in}_{(u,v)} = \frac{I_v}{\sum_{p \in N_{out}(u)} I_p}$，$I_v$ 和 I_p 表示 v 和 p 的入度，$N_{out}(u)$ 表示 u 的出邻居集合。$W^{out}_{(u,v)} = \frac{O_v}{\sum_{p \in N_{out}(u)} O_p}$，$O_v$ 和 O_p 表示 v 和 p 的入度。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_weighted_pagerank(@sample_1,
"/tmp/view_1", '{factor: 0.85, rounds: 10, mode: 1, edgeWeightProperty: "weight", limit: 10}')
as unapply(vertex, rank) return uid(vertex), rank;
```

3. CONFIG参数说明
   a. factor: 0.85，用于指定 damping_factor，默认为0.85
   b. rounds: 10，用于指定算法循环次数，默认为10
   c. mode: 1，用于设置算法执行模式，可以指定模式1或模式2，两种模式算法逻辑不同，默认为模式1
   d. edgeWeightProperty: "weight"，edgeWeightProperty 用来指定算法使用的边权重字段。当使用模式1时，如果使用已经 warmup 好的数据，需要确保 warmup 时指定了 edgeWeightProperty 字段，否则需要重新 warmup。模式2不需要指定 edgeWeightProperty
4. 返回字段
   a. vertex，即点类型
   b. rank，即PageRank值，数据类型为 double 类型

6.1.5.5. 特征向量中心性(Eigenvector Centrality)

1. 算法简介 特征向量中心性算法基于这种思想："如果节点的邻居都很重要，那么这个节点也是重要的"。算法给每个节点赋予相同的初值， 并在每一轮迭代中，把每个节点上的值通过出边传递给它的邻居。在图的邻接矩阵表示下，算法等同于求解矩阵的最大特征值和对应的特征向量。 PageRank 算法也是特征向量中心性算法的一种变体。特征向量中心性计算公式为：

$$EC_i(v) = \frac{1}{\lambda} \sum_{u \in N_{in}(v)} EC_{i-1}(u)$$

其中，λ 为一个常数，$N_{in}(v)$ 是节点 v 的相邻节点集合。经过一系列变形，该公式可变换为特征向量方程 $Ax = \lambda x$，A 为图的邻接矩阵，x 为中心行向量，也是 A 的特征向量，λ 为矩阵 A 的特征值。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with
graph_eigenvector_centrality(@sample_1, "/tmp/view_1", '{limit: 10}') as unapply(vertex, score)
return uid(vertex), score;
```

3. CONFIG参数说明
   a. rounds: 10，用于设置算法循环次数，默认为10
   b. tolerance: 0.0000001，用于设置算法迭代精度控制，默认为0.0000001
4. 返回字段
   a. vertex，即点类型
   b. score，即Eigenvector Centrality得分，数据类型为 double 类型

6.1.5.6. 介度中心性(Betweenness Centrality)

1. 算法简介 节点的介数表示一个网络中经过该结点的最短路径的数量。节点的介数越大，那么它在结点间的重要程度也越大。其计算公式为：

$$Betweenness(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

其中，s 和 t 是图上与 v 不同的任意两个节点。σ_{st} 是从点 s 到点 t 的最短路径的总数；$\sigma_{st}(v)$ 表示所有最短路径中经过 v 的路径的数量。

2. 调用示例 - 展示每个点的介度中心性

```
create query temporary graph view sample_1 as (v) [e] with graph_node_betweenness(@sample_1,
"/tmp/view_1", '{}') as unapply(vertex, betweenness) return uid(vertex),
betweenness;
```

3. 调用示例 - 展示经过每个点的最短路径条数（起点终点不认定为最短路径经过的点）

```
create query temporary graph view sample_1 as (v) [e] with graph_node_betweenness(@sample_1,
"/tmp/view_1", '{normalized:false, include_endpoints:false}') as unapply(vertex, betweenness)
return uid(vertex), betweenness;
```

4. CONFIG参数说明
   a. direction: "out" | "in"，用于指定计算过程中使用边的为出度或者入度，默认out
   b. sample: 16，用于设置算法运行起点的采样个数，默认为16
   c. normalized: true，用于设置是否对结果进行归一化处理，归一化使用`1/ vertex_number-1)*(vertex_number-2`公式，默认为true
   d. include_endpoints: true，用于设置算法计算过程中是否将起点终点认定为最短路径经过的点，默认为true
5. 返回字段
   a. vertex，即点类型
   b. betweenness，即Betweenness值，数据类型为 `double` 类型

6.1.5.7. 接近中心性(Closeness Centrality)

1. 算法简介 接近中心性算法用于发现网络中的重要节点。接近中心性计算节点到其他节点的最短路径距离之和，然后对得到的和求倒数。如果节点到图中其他节点的最短距离都很小，那么它的接近中心性就很高。其计算公式为：

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

其中，n 为节点 u 的邻居总数，$d(v,u)$ 表示节点 v 和 u 之间的最短路径长度。Wasserman 和 Faust 针对图中有多个连通分量的情况，对算法做了改进：$C(u) = \frac{n-1}{N-1} \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$，$N$ 为节点总数。

   接近中心性算法适用于连通图，非连通图的情况可以使用调和中心性。StellarDB中采用Wasserman和Faust的版本。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_closeness(@sample_1,
"/tmp/view_1", '{limit: 10}') as unapply(vertex, closeness) return uid(vertex), closeness;
```

3. CONFIG参数说明
   a. direction: "out" | "in"，用于指定计算过程中使用边的为出度或者入度，默认in

4. 返回字段
   a. vertex，即点类型
   b. closeness，即Closeness值，数据类型为 `double` 类型

6.1.5.8. 调和中心性(Harmonic Centrality)

1. 算法简介 调和中心性算法用于发现网络中的重要节点。调和中心性计算节点到其他节点的最短路径距离的倒数之和。如果节点到图中其他节点的最短距离都很小，那么它的调和中心性就很高。其计算公式为：

$$C(u) = \sum_{v \neq u} \frac{1}{d(v,u)}$$

其中，$d(v,u)$ 表示节点 v 和 u 之间的最短路径长度。

   调和中心性算法适用于非连通图，连通图的情况可以使用接近中心性。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_harmonic_centrality(@sample_1,
"/tmp/view_1", '{limit: 10}') as unapply(vertex, score) return uid(vertex), score;
```

3. CONFIG参数说明：无参数配置
4. 返回字段
   a. vertex，即点类型
   b. score，即Harmonic Centrality得分，数据类型为 `double` 类型

6.1.5.9. 影响力最大化(Greedy Influence Maximization)

1. 算法简介 `Greedy Influence Maximization` 算法用于计算网络影响力最大的点集。
2. 调用示例

示例1 – 计算网络中影响最大的4个节点，计算过程使用权重代表影响力

```
create query temporary graph view sample_1 as (a) [b] with
graph_greedy_influence_maximization(@sample_1, "/tmp/view_1", 4) as unapply( vertex, spread )
return uid(vertex), spread limit 10;
```

示例2 – 计算网络中影响最大的4个节点，计算过程通过迭代来计算影响力

```
create query temporary graph view sample_1 as (a) [b] with
graph_greedy_influence_maximization(@sample_1, "/tmp/view_1", 4, '{useIC: true,
monteCarloSimulations: 1000, propagationProbability: 0.1}') as unapply( vertex, spread ) return
uid(vertex), spread limit 10;
```

3. CONFIG参数说明
   a. useIC: false，用于指定是否使用IC方式计算影响力，默认 `false`，计算较慢不建议开启
   b. monteCarloSimulations: 1000，用于设置计算影响力时迭代轮次，默认1000，仅当 `useIC` 为 `true` 时生效
   c. propagationProbability: 0.1，用于设置计算影响力时传播的可能性，默认0.1，仅当 `useIC` 为 `true` 时生效
4. 返回字段

a. vertex，即点类型

b. spread，即影响力，数据类型为 double 类型

6.1.5.10. CELF Influence Maximization

1. 算法简介 CELF Influence Maximization 算法计算网络影响力最大的点集。
2. 调用示例

示例1 – 计算网络中影响最大的4个节点，计算过程使用权重代表影响力

```
create query temporary graph view sample_1 as (a) [b] with
graph_celf_influence_maximization(@sample_1, "/tmp/view_1", 4) as unapply( vertex, spread )
return uid(vertex), spread limit 10;
```

示例2 – 计算网络中影响最大的4个节点，计算过程通过迭代来计算影响力

```
create query temporary graph view sample_1 as (a) [b] with
graph_celf_influence_maximization(@sample_1, "/tmp/view_1", 4, '{useIC: true,
monteCarloSimulations: 1000, propagationProbability: 0.1}') as unapply( vertex, spread ) return
uid(vertex), spread limit 10;
```

3. CONFIG参数说明
   a. useIC: false，用于设置是否使用 IC 方式计算影响力，默认 false，计算较慢不建议开启
   b. monteCarloSimulations: 1000，用于设置计算影响力时迭代轮次，默认1000，仅在 useIC 为 true 时生效
   c. propagationProbability: 0.1，用于计算影响力时传播的可能性，默认0.1，仅在 useIC 为 true 时生效
4. 返回字段
   a. vertex，即点类型
   b. spread，即影响力，数据类型为 double 类型

6.1.5.11. TrustRank

1. 算法简介 TrustRank 算法可用于诈骗网页检测等场景。算法使用了类似 Personalized PageRank 的思想，从一系列由专家评估的"白名单"节点出发，将这些节点上的Trust值(或信用评分)以 PageRank 算法的方式传播给网络中的其他节点，从而实现各节点的信用评分。

   TrustRank算法可分为两个过程：

   a. 第一步(SelectSeed)会确定一组备选节点。由于对整个网络进行人工评分开销过大，需要选出较为"值得评估"的种子节点集合。 SelectSeed 算法通过 Inverse PageRank 来选出对网络影响较大的节点，用户可以对少数评分较高的节点进行人工评估，从而确定白名单列表。其计算公式为：

$$PR_i^{inverse} = \frac{1-d}{|V|} + d \cdot \sum_{u \in N_{in}(v)} \frac{PR_{i-1}^{inverse}(v)}{N_{out}(v)} + \frac{d}{|V|} \cdot PR_{i-1}^{inverse}(v)$$

   其中，d 为阻尼系数，N_{out} 为节点的出邻居。公式为反向的 $PageRank$，每轮计算会将每条边的终点 v 上的 $PR^{inverse}$ 值传到起点 u 上，从而选出对整个网络影响较大的节点。

   b. 第二步则根据白名单列表，进行 TrustRank 运算。其计算公式为：

$$Trust_i(v) = (1-d)r_v + d \cdot \sum_{u \in N_{in}(v)} \frac{Trust_{i-1}(u)}{N_{out}(u)}$$

$$r_v = \begin{cases} \frac{1}{|seedNodes|}, & v \in seedNodes \\ 0, & v \notin seedNodes \end{cases}$$

其中，d 为阻尼系数，N_{out} 为节点的出邻居。$seedNodes$ 为用户指定的种子点集，也就是经过人工评估后认为可信赖的节点。

2. 调用示例

   a. SelectSeed调用示例

```
create query temporary graph view sample_1 as (v) [e] with
graph_trustrank_select_seed(@sample_1, "/tmp/view_1", '{factor: 0.85, rounds: 10}') as
unapply(vertex, rank) return uid(vertex), rank order by rank desc limit 20;
```

   i. CONFIG参数说明

      A. factor: 0.85，用于设置 damping_factor，默认为0.85

      B. rounds: 10，用于设置算法循环次数，默认为10

   ii. 返回字段

      A. vertex，即点类型

      B. rank，即Inverse PageRank值，数据类型为 double 类型

   b. TrustRank调用示例

```
decl seedNodes:list[bytes];

seedNodes:= match (a) where uid(a) in ["1","2","3","4","5"] return collect(id(a));

create query temporary graph view sample_1 as (v) [e] with graph_trustrank(@sample_1,
"/tmp/view_1", @seedNodes, '{factor: 0.85, rounds: 10}') as unapply(vertex, rank) return
uid(vertex), rank order by rank desc limit 20;
```

   i. CONFIG参数说明

      A. factor: 0.85，用于设置 damping_factor，默认为0.85

      B. rounds: 10，用于设置算法循环次数，默认为10

   ii. 返回字段

      A. vertex，即点类型

      B. rank，即TrustRank值，数据类型为 double 类型

6.1.5.12. Hyperlink-Induced Topic Search(HITS)

1. 算法简介 Hyperlink-Induced Topic Search(HITS) 算法基于网络结构计算节点的 Authority 评分和 Hub 评分。 Authority 评分评估节点的"权威性"，而 Hub 评分评估节点的"枢纽性"。 算法的基本假设是：一个好的 "Authority" 页面会被很多好的 "Hub" 页面指向；而一个好的 "Hub" 页面会指向很多好的 "Authority" 页面。 基于此假设，算法进行多轮迭代计算，直到网络中所有页面的 "Authority" 和 "Hub" 评分趋于稳定。其计算公式为：

$$Auth_i(v) = \sum_{u \in N_{in}(v)} Hub_{i-1}(u),$$

$$Hub_i(v) = \sum_{u \in N_{in}(v)} Auth_{i-1}(u)$$

其中，节点的 $Authority$ 值等于所有指向它的节点的 Hub 值之和；节点的 Hub 值等于所有指向它的节点的 $Authority$ 值之和。

2. 调用示例

```
create query temporary graph view sample_1 as (v) [e] with graph_hits(@sample_1, "/tmp/view_1",
'{rounds: 10, limit: 10}') as unapply(vertex, auth, hub) return uid(vertex), auth, hub;
```

3. CONFIG参数说明
   a. rounds: 10，用于设置算法循环次数，默认为10
   b. tolerance: 0.0000001，用于设置算法迭代精度控制，默认为0.0000001
   c. initial: 0.2，用于指定初始 rank 值，默认为0，算法会使用 1/(vertex number) 作为初始值
4. 返回字段
   a. vertex，即点类型
   b. authority，即 Authority 值，数据类型为 double 类型
   c. hub，即 Hub 值，数据类型为 double 类型

6.1.6. 路径搜寻(Pathfinding)

路径搜寻算法基于一定的规则，寻找两个或多个点之间的最佳路径，或从某一(组)点出发，对可达的路径进行探索。

6.1.6.1. 深度优先搜索(DFS)

1. 算法简介 宽度优先遍历算法按照深度优先的顺序进行图遍历，通过指定最大深度可以提前结束遍历过程。DFS 算法主要应用在拓扑排序的场景中，在找到结果比最优解更重要的场景下 DFS 相对 BFS 更有优势。
2. 调用示例

```
decl start_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);

create query temporary graph view sample_1 as (v) [e] with graph_dfs(@sample_1, "/tmp/view_1",
@start_node, '{maxDepth: 6, limit: 10}') as unapply(vertex, depth) return uid(vertex), depth;

drop var start_node;
```

3. CONFIG参数说明
   a. maxDepth: -1，用于设置搜索深度，默认-1代表不限制深度
4. 返回字段
   a. vertex，即点类型
   b. depth，即节点深度，数据类型为 long 类型

6.1.6.2. 广度优先搜索(BFS)

1. 算法简介 宽度优先遍历算法按照深度递增的顺序进行图遍历，计算指定节点上不同距离上的点，通过指定最大深度可以提前结束遍历过程。BFS 算法主要应用在搜寻无权图中最短距离的场景中，可以理解为单源最短路径在无权图上的特例。
2. 调用示例

```
decl start_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);

create query temporary graph view sample_1 as (v) [e] with graph_bfs(@sample_1, "/tmp/view_1",
@start_node, '{maxDepth: 6, limit: 10}') as unapply(vertex, depth) return uid(vertex), depth;

drop var start_node;
```

3. CONFIG参数说明
   a. maxDepth: -1，用于设置搜索深度，默认-1代表不限制深度
4. 返回字段
   a. vertex，即点类型
   b. depth，即节点深度，数据类型为 long 类型

6.1.6.3. 最短路径(Shortest Path)

1. 算法简介 最短路径算法计算指定两点间的最短路径，可以输出路径或者距离。最短路径算法用来计算无权图上的最短路径，有权图上适用 Dijkstra 最短路径或者 SPFA 最短路径算法。最短路径算法可以理解为指定了终点的 BFS 算法。
2. 调用示例

示例1 – 返回完整路径

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with graph_shortestpath(@sample_1,
"/tmp/view_1", @start_node, @end_node, '{output: "path"}') as unapply(allpath) return allpath;

drop var start_node;
drop var end_node;
```

示例2 – 返回距离

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with graph_shortestpath(@sample_1,
"/tmp/view_1", @start_node, @end_node, '{output: "distance"}') as unapply(distance) return
distance;

drop var start_node;
drop var end_node;
```

3. CONFIG参数说明
   a. output: "path" | "single_path" | "distance"，用于指定返回全部路径，单条路径或距离
4. 返回字段
   a. allpath，即所有路径，每个点返回其 uid，数据类型为 array 类型
   b. distance，即距离，数据类型为 double 类型

6.1.6.4. Dijkstra Shortest Path

1. 算法简介 Dijkstra 最短路径算法计算正权图上两点间的最短路径，是路径搜寻领域的经典算法。主要利用贪心的思想每次选择一条边构建最短路径，最终生成起点到终点的最短距离。
2. 调用示例

   示例1 – 返回完整路径

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with
graph_dijkstra_shortestpath(@sample_1, "/tmp/view_1", @start_node, @end_node, '{output:
"path"}') as unapply(allpath) return allpath;

drop var start_node;
drop var end_node;
```

   示例2 – 返回距离

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with
graph_dijkstra_shortestpath(@sample_1, "/tmp/view_1", @start_node, @end_node, '{output:
"distance"}') as unapply(distance) return distance;

drop var start_node;
drop var end_node;
```

3. CONFIG参数说明
   a. output: "path" | "single_path" | "distance"，用于指定返回全部路径，单条路径或距离
4. 返回字段
   a. allpath，即所有路径，每个点返回其 uid，数据类型为 array 类型
   b. distance，即距离，数据类型为 double 类型

6.1.6.5. SPFA Shortest Path

1. 算法简介 SPFA 最短路径算法计算有权图上两点间的最短路径，相较 Dijkstra 算法额外支持了带负权图上最短路径的计算，同时实现上开启并发后 SPFA 的效率优于 Dijkstra 算法。主要利用了动态规划思想不断调整节点间的距离最终生成最短路径。
2. 调用方法

   示例1 – 返回完整路径

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with graph_spfa_shortestpath(@sample_1,
"/tmp/view_1", @start_node, @end_node, '{output: "path"}') as unapply(allpath) return allpath;

drop var start_node;
drop var end_node;
```

示例2 – 返回距离

```
decl start_node:bytes;
decl end_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);
end_node := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (v) [e] with
graph_dijkstra_shortestpath(@sample_1, "/tmp/view_1", @start_node, @end_node, '{output:
"distance"}') as unapply(distance) return distance;

drop var start_node;
drop var end_node;
```

3. CONFIG参数说明
   a. output: "path" | "single_path" | "distance"，用于指定返回全部路径，单条路径或距离
4. 返回字段
   a. allpath，即所有路径，每个点返回其 uid，数据类型为 array 类型
   b. distance，即距离，数据类型为 double 类型

6.1.6.6. SPFA Single Source Shortest Path

1. 算法简介 单源最短路径算法计算指定顶点到图中其他顶点间的最短距离，算法实现上使用 SPFA 算法。
2. 调用示例

```
decl start_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);

create query temporary graph view sample_1 as (v) [e] with
graph_spfa_single_source_shortestpath(@sample_1, "/tmp/view_1", @start_node) as unapply(vertex,
distance) return uid(vertex), distance;

drop var start_node;
```

3. CONFIG参数说明：无参数配置
4. 返回字段
   a. vertex，即点类型
   b. distance，即与源点的距离，数据类型为 double 类型

6.1.6.7. 随机游走(RandomWalk)

1. 算法简介 随机游走算法从指定起点开始按照指定的边方向在图上随机游走最后形成随机游走路径。随机游走能很好的模拟现实生活中的某些场景，可以作为信息检索等领域中的基础思想，比如 PageRank 算法就是基于随机游走思想上计算不同网页的重要性算法。
2. 调用示例

```
decl start_node:bytes;

start_node := match (a) where uid(a) = "1" return id(a);

create query temporary graph view sample_1 as (a) [b] with graph_randomwalk(@sample_1,
"/tmp/xxx1234", @start_node, '{steps: 6, direction: "out"}') as unapply(vertex, step) return
uid(vertex), step order by step;

drop var start_node;
```

3. CONFIG参数说明
   a. steps: 10，用于指定游走层数
   b. direction: "out" | "in" | "both" 用于设置游走方向，可选项有出边方向、入边方向或出入边方向
4. 返回字段
   a. vertex，即点类型
   b. step，即节点层数

6.1.7. 社区发现(Community Detection)

社区发现算法根据一定的规则将节点划分成社区，能够反映出丰富的网络结构信息。

6.1.7.1. 连通子图(Connected Components)

1. 算法简介 弱连通分量算法计算图中沿无向边连通的点集合，单个弱连通分量中的所有点间都有路径可达（不区分方向）。弱连通分量算法常用来在图分析的早期阶段理解图的结构。
2. 调用示例

示例1 – 返回每个点对应的社区号

```
create query temporary graph view sample_1 as (v) [e] with graph_connected_component(@sample_1,
"/tmp/view_1", '{limit: 10}') as unapply(vertex, community) return uid(vertex), community;
```

示例2 – 返回社区个数和最大社区的大小

```
create query temporary graph view sample_1 as (v) [e] with graph_connected_component(@sample_1,
"/tmp/view_1", '{limit: 10, printVertex: false}') as unapply(wccNum, maxWccSize) return wccNum,
maxWccSize;
```

3. CONFIG参数说明
   a. printVertex: true，用于指定是否打印每个节点对应的社区号，默认为 `true`
4. 返回字段
   a. vertex，即点类型
   b. community，即社区号，数据类型为 `long` 类型
   c. wccNum，即社区个数，数据类型为 `long` 类型
   d. maxWccSize，即最大社区包含的节点数，数据类型为 `long` 类型

6.1.7.2. 强连通子图(Strongly Connected Components)

1. 算法简介 强连通分量算法计算图中沿有向边连通的点集合，单个强连通分量中的所有点间沿着出边或者入边都有路径可达。强连通分量算法常见的应用是在算法分析阶段通过找出强连通分量后缩点形成有向无环图后简化图结构。
2. 调用示例

示例1 – 返回每个节点对应的社区号

```
create query temporary graph view sample_1 as (v) [e] with
graph_strongly_connected_component(@sample_1, "/tmp/view_1", '{limit: 10}') as unapply(vertex,
community) return uid(vertex), community;
```

示例2 – 返回社区个数和最大社区的大小

```
create query temporary graph view sample_1 as (v) [e] with
graph_strongly_connected_component(@sample_1, "/tmp/view_1", '{limit: 10, printVertex: false}')
as unapply(sccNum, maxSccSize) return sccNum, maxSccSize;
```

3. CONFIG参数说明
   a. printVertex: true，用于指定是否打印每个节点对应的社区号，默认为 `true`
   b. limit: 10，用于指定算法返回 `K` 个节点的社区号
   c. directed: true，用于指定是否使用有向图计算，默认使用有向图
4. 返回字段
   a. vertex，即点类型
   b. community，即社区号，数据类型为 `long` 类型
   c. wccNum，即社区个数，数据类型为 `long` 类型
   d. maxWccSize，即最大社区包含的节点数，数据类型为 `long` 类型

6.1.7.3. 标签传播(LPA)

1. 算法简介 标签传播算法是基于图的半监督学习方法，通过使用已经标记的点的标签与测其邻居节点的标签信息，并依据最终标签信息划分社区。其主要逻辑是：
   a. 每个节点初始赋予一个唯一的社区编号
   b. 开启迭代传播标签，每个节点将自身标签更新为邻居节点中出现最多次的标签
   c. 迭代直到所有节点标签都不再变化或者达到最大迭代次数
2. 调用示例

示例1 – 返回每个节点对应的社区号

```
create query temporary graph view sample_1 as (a) [b] with graph_lpa(@sample_1, "/tmp/view_1",
'{rounds: 10, limit: 10}') as unapply(vertex, community) return uid(vertex), community;
```

示例2 – 返回社区号及模块度

```
create query temporary graph view sample_1 as (a) [b] with graph_lpa(@sample_1, "/tmp/view_1",
'{rounds: 10, limit: 10, output: "stat"}') as unapply(number, modularity) return number,
modularity;
```

3. CONFIG参数说明
   a. rounds: 10，用于设置迭代轮数
   b. output: "stat"，用于设置输出模式，默认为空输出每个节点对应的社区号，指定为 `stat` 输出社区个数及对应的模块度
4. 返回字段
   a. vertex，即点类型

b. community，即社区号，数据类型为 long 类型

c. number，即社区个数，数据类型为 long 类型

d. modularity，即模块度，数据类型为 double 类型

6.1.7.4. Speaker-Listener标签传播(SLPA)

1. 算法简介 Speaker-Listener 标签传播算法（SLPA）是标签传播算法的一个变种，可以用来计算可重叠社区的划分。相对标签传播算法，现实中很多实体信息可能包含于多个社区中，SLPA 算法可以在迭代过程中维护顶点的多个标签，并通过指定阈值过滤概率较低的标签。
2. 调用示例

```
create query temporary graph view sample_1 as (a) [b] with graph_slpa(@sample_1, "/tmp/view_1",
'{rounds: 10, threshold: 0.2}') as unapply(vertex, communities) return uid(vertex), communities;
```

3. CONFIG参数说明
   a. rounds: 10，用于设置迭代轮数
   b. threshold: 0.2，用于指定过滤标签使用的阈值，出现次数小于该阈值的标签将在结果前被剔除
4. 返回字段
   a. vertex，即点类型
   b. communities，即节点属于的标签，数据类型为 array 类型

6.1.7.5. Fast Unfolding(Louvain)

1. 算法简介 FastUnfolding 算法也叫 **鲁汶算法**，该算法是基于模块度优化的社区检测算法，尤其适用于发现层次性的社区结构，其主要思想是通过贪心选择优化最大化社区的整体模块度。模块度也称模块化度量值，是目前常用的一种衡量网络社区结构强度的方法。模块度越大的社区社区的紧密程度也越高，较为合理的社区划分模块度分布在0.3-0.7之间。模块度的计算公式一般为：

$$Q = \frac{1}{2m}\sum_{ij}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j)$$

其中，A_{ij} 表示顶点 i 和 j 之间边的权重，$m = \frac{1}{2}\sum_{ij} A_{ij}$ 代表图中所有边的权重之和，$k_i = \sum_j A_{ij}$ 表示顶点 i 上所有边的权重，c_i 表示顶点 i 所在社区，$\delta(c_i, c_j)$ 表示顶点 i 所在社区与 j 所在社区相同的概率，如果 $c_i = c_j$ 则 $\delta(c_i, c_j) = 1$，否则 $\delta(c_i, c_j) = 0$。

迭代过程中，模块度差值计算公式为：

$$\Delta Q = [\frac{k_{i,in}}{2m} - \frac{\sum_{tot} k_i}{2m^2}]$$

其中，$k_{i,in}$ 表示社区 c 中节点与节点 i 的边权重之和，$\sum_{tot}$ 表示与社区 c 中节点相连边的权重。

2. 调用示例

示例1 – 返回每个节点对应的社区号

```
create query temporary graph view sample_1 as (v) [e] with graph_fastunfolding(@sample_1,
"/tmp/view_1", '{limit: 10}') as unapply(vertex, community) return uid(vertex), community;
```

示例2 – 返回社区数及模块度

```
create query temporary graph view sample_1 as (a) [b] with graph_fastunfolding(@sample_1,
"/tmp/view_1", '{limit: 10, output: "stat"}') as unapply(number, modularity) return number,
modularity;
```

3. CONFIG参数说明

a. rounds：10，用于设置迭代轮数

b. tolerance：0.000001，用于指定迭代过程中模块度变化阈值，出现模块度变化小于该阈值时算法结束

c. output："stat"，用于设定输出模式，默认为空输出每个节点对应的社区号，指定为 `stat` 输出社区个数及对应的模块度

4. 返回字段

a. vertex，即点类型

b. community，即社区号，数据类型为 `long` 类型

c. number，即社区个数，数据类型为 `long` 类型

d. modularity，即模块度，数据类型为 `double` 类型

6.1.8. 相似度(Similarity)

相似度算法从一系列可选的度量指标出发，评估给定两个节点之间的相似度，也可用于找出相似度最高的节点对。

6.1.8.1. Jaccard相似度

1. 算法简介 用于比较有限样本集之间的相似性和差异性定义。公式为：

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A|+|B|-|A\cap B|}$$

其中，A 和 B 为两个集合。

2. 调用示例

```
decl x:list[bytes];
decl y:list[bytes];

x := match (a) where uid(a) in ["1","2","3"] return collect(id(a));
y := match (b) where uid(b) in ["1","2","4"] return collect(id(b));

return graph_jaccard_similarity(@x, @y);

drop var x;
drop var y;
```

3. CONFIG参数说明：无参数配置

4. 返回字段：相似度值，数据类型为 `double` 类型

6.1.8.2. Pearson相似度

1. 算法简介 用于比较有限样本集之间的相似性和差异性定义。公式为：

$$Pearson(A,B)=\frac{cov(A,B)}{\rho_A\rho_B}$$

其中，A 和 B 为通过集合构造的向量。$cov(A,B)$ 为 A、B 的协方差，ρ_A 为 A 的标准差，ρ_B 为 B 的标准差。

2. 调用示例

a. 示例1 - 比较两个点集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) in ["1","2","3"] return collect(todouble(uid(a)));
y := match (b) where uid(b) in ["1","2","4"] return collect(todouble(uid(b)));

return graph_pearson_similarity(@x, @y);

drop var x;
drop var y;
```

b. 示例2 - 比较两个点的属性集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) = "1" return [todouble(a.age),todouble(a.salary)];
y := match (b) where uid(b) = "2" return [todouble(b.age),todouble(b.salary)];

return graph_pearson_similarity(@x, @y);

drop var x;
drop var y;
```

3. CONFIG参数说明：无参数配置
4. 返回字段：相似度值，数据类型为 `double` 类型

6.1.8.3. Cosine相似度

1. 算法简介：用于比较有限样本集之间的相似性和差异性定义。公式为：

$$cos(A,B) = \frac{A \cdot B}{||A|| \cdot |B||} = \sum_{i=1}^{n} \frac{a_i \times b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \times \sqrt{\sum_{i=1}^{n} b_i^2}}$$

其中，A 和 B 为通过集合构造的向量。

2. 调用示例
   a. 示例1 - 比较两个点集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) in ["1","2","3"] return collect(todouble(uid(a)));
y := match (b) where uid(b) in ["1","2","4"] return collect(todouble(uid(b)));

return graph_cosine_similarity(@x, @y);

drop var x;
drop var y;
```

   b. 示例2 - 比较两个点的属性集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) = "1" return [todouble(a.age),todouble(a.salary)];
y := match (b) where uid(b) = "2" return [todouble(b.age),todouble(b.salary)];

return graph_cosine_similarity(@x, @y);

drop var x;
drop var y;
```

3. CONFIG参数说明：无参数配置
4. 返回字段：似度值，数据类型为 `double` 类型

6.1.8.4. Overlap相似度

1. 算法简介：用于比较有限样本集之间的相似性和差异性定义。公式为：

$$O(A,B) = \frac{|A \cap B|}{min(|A|,|B|)}$$

其中，A 和 B 为两个集合。

2. 调用示例

   a. 示例1 - 比较两个点集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) in ["1","2","3"] return collect(todouble(uid(a)));
y := match (b) where uid(b) in ["1","2","4"] return collect(todouble(uid(b)));

return graph_overlap_similarity(@x, @y);

drop var x;
drop var y;
```

   b. 示例2 - 比较两个点的属性集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) = "1" return [todouble(a.age),todouble(a.salary)];
y := match (b) where uid(b) = "2" return [todouble(b.age),todouble(b.salary)];

return graph_overlap_similarity(@x, @y);

drop var x;
drop var y;
```

3. CONFIG参数说明：无参数配置
4. 返回字段：相似度值，数据类型为 `double` 类型

6.1.8.5. 欧式距离相似度(Euclidean Distance)

1. 算法简介：用于计算样本间的欧式距离。公式为：

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

其中，A 和 B 为通过集合构造的向量。其中 a_i 表示第一个点的第 i 维坐标，b_i 表示第二个点的第 i 维坐标。

2. 调用示例

   a. 示例1 - 比较两个点集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) in ["1","2","3"] return collect(todouble(uid(a)));
y := match (b) where uid(b) in ["1","2","4"] return collect(todouble(uid(b)));

return graph_euclidean_distance(@x, @y);

drop var x;
drop var y;
```

   b. 示例2 - 比较两个点的属性集合之间的相似度

```
decl x:list[double];
decl y:list[double];

x := match (a) where uid(a) = "1" return [todouble(a.age),todouble(a.salary)];
y := match (b) where uid(b) = "2" return [todouble(b.age),todouble(b.salary)];

return graph_euclidean_distance(@x, @y);

drop var x;
drop var y;
```

3. CONFIG参数说明：无参数配置
4. 返回字段：相似度值，数据类型为 `double` 类型

6.1.8.6. 节点相似度

1. 算法简介：用于比较指定节点之间的相似性和差异性定义。其中除 `Dice` 系数外其他系数的公式已在上文给出，`Dice` 系数的公式补充如下：

$$Dice(A,B) = \frac{2|A \cap B|}{|A| + |B|}$$

其中，A 和 B 分别为节点 x 和 y 的邻居集合，$A = N(x)$，$B = N(y)$。

2. 调用示例

示例1 – 返回与指定节点Jaccard相似度最高的K个节点及其相似度

```
decl node1:bytes;

node1 := match (a) where uid(a)="1000" return id(a);

create query temporary graph view sample_1 as (a) [b] with graph_node_similarity(@sample_1,
"/tmp/view_1", @node1, '{metric:"jaccard",topk: 10}') as unapply( vertex1, vertex2, similarity )
return vertex1, vertex2, similarity;

drop var node1;
```

示例2 – 返回与指定节点Overlap相似度最低的K个节点及其相似度

```
decl node1:bytes;

node1 := match (a) where uid(a)="1000" return id(a);

create query temporary graph view sample_1 as (a) [b] with graph_node_similarity(@sample_1,
"/tmp/view_1", @node1, '{metric:"overlap",bottomk: 10}') as unapply(vertex1, vertex2,
similarity) return vertex1, vertex2, similarity;

drop var node1;
```

示例3 – 返回与指定节点Dice相似度最低的K个节点及其相似度

```
decl node1:bytes;

node1 := match (a) where uid(a)="1000" return id(a);

create query temporary graph view sample_1 as (a) [b] with graph_node_similarity(@sample_1,
"/tmp/view_1", @node1, '{metric:"dice",bottomk: 10}') as unapply( vertex1, vertex2, similarity)
return vertex1, vertex2, similarity;

drop var node1;
```

示例4 – 返回全图范围内Dice相似度最低的K个节点及其相似度

```
create query temporary graph view sample_1 as (a) [b] with graph_node_similarity(@sample_1,
"/tmp/view_1", '{metric:"dice",bottomk: 10}') as unapply( vertex1, vertex2, similarity) return
vertex1, vertex2, similarity;
```

3. CONFIG参数说明
   a. metric: "jaccard" | "overlap" | "dice"，用于指定使用的相似度计算方式
   b. topk: 10，用于指定返回相似度最高的 K 个节点
   c. bottomk: 10，用于指定返回相似度最低的 K 个节点，若同时指定 topk 与 bottomk 则 topk 生效
   d. @node1，如果要在全图范围计算相似度，在算法参数 @node1 处不填值即可
4. 返回字段
   a. vertex1，即节点1的 uid
   b. vertex2，即节点2的 uid
   c. similarity，即相似度，数据类型为 double 类型

6.1.9. 链路预测(Link Prediction)

链路预测算法基于既有的网络结构，预测节点之间产生新链接的可能性。

6.1.9.1. 共同邻居(Common Neighbors)

1. 算法简介 算法返回两个指定节点的共同邻居数量，若共同邻居数量多，则两个节点间未来产生链接的可能性也更高。其计算公式为：

$$CN(x,y) = |N(x) \cap N(y)|$$

其中，$N(x)$，$N(y)$ 分别为节点 x 和 y 的邻居集合。

2. 调用示例

```
decl x:bytes;
decl y:bytes;

x := match (a) where uid(a) = "1" return id(a);
y := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (a) [b] with graph_common_neighbors(@sample_1,
"/tmp/view_1", @x, @y, '{relationship_direction="BOTH"}') as unapply(score) return score ;

drop var x;
drop var y;
```

3. CONFIG参数说明

   relationship_direction: "INCOMING" | "OUTGOING" | "BOTH"，用于设置算法使用出边、入边或者双向边统计邻居数量

4. 返回字段

   score，即共同邻居数量，数据类型为 int 类型

6.1.9.2. 总邻居数(Total Neighbors)

1. 算法简介 算法返回两个指定节点的总邻居数量，若总邻居数量多，则两个节点间未来产生链接的可能性也更高。其计算公式为：

$$TN(x,y) = |N(x) \cup N(y)|$$

其中，$N(x)$，$N(y)$ 分别为节点 x 和 y 的邻居集合。

2. 调用示例

```
decl x:bytes;
decl y:bytes;

x := match (a) where uid(a) = "1" return id(a);
y := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (a) [b] with graph_total_neighbors(@sample_1,
"/tmp/view_1", @x, @y, '{relationship_direction="BOTH"}') as unapply( score ) return score;

drop var x;
drop var y;
```

3. CONFIG参数说明

   relationship_direction: "INCOMING" | "OUTGOING" | "BOTH"，用于设置算法使用出边、入边或者双向边统计邻居数量

4. 返回字段

   score，即共同邻居数量，数据类型为 `int` 类型

6.1.9.3. Adamic Adar

1. 算法简介：算法计算两个节点每个共同邻居的邻居数量，取对数的倒数后，进行求和。其计算公式为：

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log|N(u)|}$$

其中，$N(x)$，$N(y)$ 分别为节点 x，y 的邻居集合。

2. 调用示例

```
decl x:bytes;
decl y:bytes;

x := match (a) where uid(a) = "1" return id(a);
y := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (a) [b] with graph_adamic_adar(@sample_1,
"/tmp/view_1", @x, @y, '{relationship_direction="BOTH"}') as unapply(score) return score ;

drop var x;
drop var y;
```

3. CONFIG参数说明

   relationship_direction: "INCOMING" | "OUTGOING" | "BOTH"，用于设置算法使用出边、入边或者双向边统计邻居数量

4. 返回字段

   score，即共同邻居数量，数据类型为 `int` 类型

6.1.9.4. Resource Allocation

1. 算法简介：算法计算两个节点每个共同邻居的邻居数量，取倒数后进行求和。其计算公式为：

$$RA(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{|N(u)|}$$

其中，$N(x)$，$N(y)$ 分别为节点 x，y 的邻居集合。

2. 调用示例

```
decl x:bytes;
decl y:bytes;

x := match (a) where uid(a) = "1" return id(a);
y := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (a) [b] with graph_resource_allocation(@sample_1,
"/tmp/view_1", @x, @y, '{relationship_direction="BOTH"}') as unapply(score) return score ;

drop var x;
drop var y;
```

3. CONFIG参数说明

   relationship_direction: "INCOMING" | "OUTGOING" | "BOTH"，用于设置算法使用出边、入边或者双向边统计邻居数量

4. 返回字段

   score，即共同邻居数量，数据类型为 int 类型

6.1.9.5. Preferential Attachment

1. 算法简介：算法返回两个指定节点的邻居数量的乘积。其计算公式为：

$$PA(x,y) = |N(x)| * |N(y)|$$

其中，$N(x)$，$N(y)$ 分别为节点 x，y 的邻居集合。

2. 调用示例

```
decl x:bytes;
decl y:bytes;

x := match (a) where uid(a) = "1" return id(a);
y := match (b) where uid(b) = "100" return id(b);

create query temporary graph view sample_1 as (a) [b] with
graph_preferential_attachment(@sample_1, "/tmp/view_1", @x, @y,
'{relationship_direction="BOTH"}') as unapply(score) return score;

drop var x;
drop var y;
```

3. CONFIG参数说明

   relationship_direction: "INCOMING" | "OUTGOING" | "BOTH"，用于设置算法使用出边、入边或者双向边统计邻居数量

4. 返回字段

   score，即共同邻居数量，数据类型为 int 类型

6.2. StellarDB3.0算法兼容性说明

StellarDB3.0版本图算法在StellarDB4.0版本中可以运行，但部分算法在4.0版本中不再支持，部分算法在超大规模图（数百亿边）中拥有更好性能。

6.2.1. PageRank

1. 算法简介：PageRank 算法计算网络中点的相关性和重要性，由Google公司发明，常见用途是网页排名。在StellarDB 4.0版本中对该算法进行了参数修改，使得该算法更加适合超大图。
2. 调用方法

```
graph_pagerank(vertex_caller, vertex_callee, half(Boolean) = false, factor(Double) = 0.85,
iters = 10);
```

3. 参数说明
   a. vertex_caller，用于指定起点列名
   b. vertex_callee，用于指定终点列名
   c. half，用于指定是否图数据只有一半（用于无向图只提供一半边数据的情况），**必须** 为 false
   d. factor，用于设置 damping_factor，默认为0.85
   e. iters，用于设置算法循环次数，默认为10
4. 返回列名
   a. vertex，即节点的内部 id
   b. rank，即 PageRank 值
5. 示例

```
match [f] with graph_pagerank(startuid(f), enduid(f)) as unapply(vertex, rank) return vertex,
rank limit 10;
```

6.2.2. GuaranteeCircle

1. 算法简介：计算全图中的所有环。目前该算法暂不支持StellarDB 4.0新语法。
2. 参数说明：无参数配置
3. 返回字段
   a. vertex，即节点的内部 id
   b. circle_id，即处于环上的节点 id
4. 示例

```
match [f] with graph_circle(startuid(f), enduid(f)) as unapply(vertex, circle_id) return vertex,
circle_id limit 10;
```

6.2.3. All Pair Shortest Path

1. 算法简介：多源最短路径用于计算网络中任意两节点的最短距离。目前该算法暂不支持StellarDB 4.0新语法。
2. 调用方法

```
graph_allpair_shortestpath(src, dst, weight, half(Boolean) = false, printType(String) =
'distance' | 'path' | 'both', printMultiplePaths(Boolean) = false, step(Int) = 6,
pathNumber(Int)
= 5, dataClean(Boolean) = false, <order(Boolean) = false>);
```

3. 参数说明
   a. src，用于指定起点列名
   b. dst，用于指定终点列名
   c. weight，用于指定权重列名
   d. half，用于指定是否图数据只有一半（用于无向图只提供一半边数据的情况），必须为 false
   e. printType，用于指定输出最短路径类型，共有距离值（distance）、路径（path），距离值组合路径（both）三种选择
   f. printMultiplePaths：如果最短路径有多条，则输出多条路径
   g. step：用于多源最短路径的距离限制
   h. pathNumber：如果最短路径有多条，则最多输出 pathNumber 条路径；若 pathNumber 为-1，则全部输出
   i. dataClean：如果输入数据存在重复边或 null 值等脏数据情况，则将该参数置为 true
   j. order：打印路径或者距离时，是否根据路径的 id 或者距离值排序. 默认为 false
4. 返回列名
   a. (src, dst, distance)，即起点、终点、距离
   b. (src, dst, path)，即起点、终点、路径
   c. (src, dst, distance, path)，即起点、终点、距离、路径
5. 示例

```
match [f] with graph_allpair_shortestpath(startuid(f), enduid(f), 1, false, 'distance', false,
6,
1, false) as unapply(src, dst, distance) return src, dst, distance limit 10;
```

6.2.4. Triangle Counting

1. 算法简介：Triangle Counting 用于计算图中三角形数量，常用于社交网络结构分析，垃圾邮件分析和欺诈检测。此处函数返回值有变动。
2. 调用方法

```
graph_triangle_count(src, dst);
```

3. 返回列名

   triangles，即三角形数量，数据类型为 long 类型

4. 示例

```
match [f] with graph_triangle_count(startuid(f), enduid(f)) as unapply(triangles)
return triangles;
```

6.2.5. Global Cluster Coefficient（deprecated）

StellarDB 5.0.0 对该函数不再进行支持

6.2.6. Eccentricity（deprecated）

StellarDB 5.0.0 对该函数不再进行支持。

6.3. 图算法相关扩展：

6.3.1. 图算法结果写回图 (3.0版本)

```
match (a)-[f]->(b) with <GRAPH_ALGO> as unapply(vertex, result) set
findNodeWithId(vertex).property_name = result;
```

示例：

```
match (a)-[f]->(b) with graph_pagerank(id(a), id(b)) as unapply(v, r) set findNodeWithId(v).rank =
r;
```

通过执行该语句，将 PageRank 算法，通过用 findNodeWithId() 函数找到对应节点，并将节点的 rank 值更新为 r。

- findNodeWithId()方法意义是根据内部id找到节点，所以要求参数vertex来源是内部id值，目前供此场景下专用。
- 目前property_name只能是创图时schema里指定的列名，不支持动态生成新的列名。如果预期要根据图算法写回图中，请提前在schema里指定要写入的列。

6.3.2. 图算法结果写回图 (4.0版本及以上)

```
create query temporary graph view GRAPH_VIEW_NAME as (v) [e] with GRAPH_ALGO(@GRAPH_VIEW_NAME,
VIEW_STORE_PATH, CONFIG_MAP, PROPS) as unapply(vertex, result) bulk set
findNodeWithId(vertex).property_name = result;
```

示例：

```
create temporary graph view sample_1 as (v) [e] with graph_pagerank(@sample_1, "/tmp/view_1",
'{factor: 0.85, rounds: 5}') as unapply(v, r) bulk set findNodeWithId(v).rank = r;
```

通过执行此语句，将 PageRank 算法，通过用 findNodeWithId() 函数找到对应节点，并将节点的 rank 值更新为 r。

目前property_name只能是创图时schema里指定的列名，不支持动态生成新的列名。如果预期要根据图算法写回图中，请提前在schema里指定要写入的列。

6.3.3. 图算法结果导出至关系型表

图算法结果支持导出至Quark的文本型数据表：

1. 表结构将由导出内容决定；
2. 模型列名为_a_c0，_a_c1等，需要通过alias来自定义列名；

示例 - 导出PageRank算法结果至test_db.page_rank_result

```
create query temporary graph view algo_test_view_1 as (a) [b] with graph_pagerank(@algo_test_view_1,
"/tmp/algo_test_view_1", '{factor: 0.85, rounds: 10}') as unapply(vertex, rank) store
tostring(uid(vertex)) as uid, todouble(rank) as rank_value into txt table test_db.page_rank_result;
```

7. StellarDB自定义函数、存储过程和图算法

7.1. 自定义函数

StellarDB支持用户添加自定义函数，添加后可在cypher语句中使用。

7.1.1. 自定义函数实现

自定义函数通过java/scala语言开发，可继承实现两种基类，编译成jar包，通过指定命令加载到StellarDB。 需要实现的基类为如下两种，可自行选择继承合适的基类：

1. 继承UDF基类
2. 继承GenericUDF基类。

7.1.1.1. 继承UDF基类

该类实现简单，功能较为单一。支持Quark的基本类型、数组和Map。适合实现简单的逻辑。

继承 **org.apache.hadoop.hive.ql.exec.UDF** 类 继承UDF类必须实现evaluate方法且返回值类型不能为 void，支持定义多个evaluate方法不同参数列表用于处理不同类型数据。

```
@Description(
    name="my_plus",
    value="my_plus() - if string, do concat; if integer, do plus",
    extended = "Example : \n    >select my_plus('a', 'b');\n    >ab\n    >select my_plus(3, 5);\n
>8"
)
/**
 * 实现UDF函数，若字符串执行拼接，int类型执行加法运算。
 */
public class AddUDF extends UDF {
    /**
     * 编写一个函数，要求如下：
     * 1. 函数名必须为 evaluate
     * 2. 参数和返回值类型可以为：Java基本类型、Java包装类、org.apache.hadoop.io.Writable等类型、List、Map
     * 3. 函数一定要有返回值，不能为 void
     */
    public String evaluate(String... parameters) {
        if (parameters == null || parameters.length == 0) {
            return null;
        }
        StringBuilder sb = new StringBuilder();
        for (String param : parameters) {
            sb.append(param);
        }
        return sb.toString();
    }
    /**
     * 支持函数重载
     */
    public int evaluate(IntWritable... parameters) {
        if (parameters == null || parameters.length == 0) {
            return 0;
        }
        long sum = 0;
        for (IntWritable currentNum : parameters) {
            sum = Math.addExact(sum, currentNum.get());
        }
        return (int) sum;
    }
}
```

7.1.1.2. 继承GenericUDF基类

该类功能更加强大，实现稍微复杂一些。支持任意长度、任意类型的参数，可以根据参数个数和类型实现不同的逻辑，资源消耗更低；

一般在以下几种场景下考虑使用GenericUDF： 传参情况复杂、该函数被高频地使用、该函数功能未来预期的重构、扩展场景较多。

GenericUDF的输入输出参数类型由ObjectInspector封装，ObjectInspector类用于存储解复杂对象的类型信息。GenericUDF需要使用的是Hadoop数据格式，不是编写UDF的Java的数据类型。下面是其对应关系：

Java类型	Hadoop类型
Byte	ByteWritable
Short	ShortWritable
Integer	IntWritable
Long	LongWritable
String	Text
Character	HiveCharWritable
Boolean	BooleanWritable
Float	FloatWritable
Double	DoubleWritable
BigDecimal	HiveDecimalWritable
Date	DateWritable
List	ArrayListWritable
Map<K.V>	HashMapWritable

继承 **org.apache.hadoop.hive.ql.udf.generic.GenericUDF**。需要实现以下三个方法：

```
public class MyCountUDF extends GenericUDF {
        private PrimitiveObjectInspector.PrimitiveCategory[] inputType;
        private transient ObjectInspectorConverters.Converter intConverter;
        private transient ObjectInspectorConverters.Converter longConverter;
        // 初始化
        @Override
        public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {
        }
        // DeferredObject封装实际参数的对应Writable类
        @Override
        public Object evaluate(DeferredObject[] deferredObjects) throws HiveException {
        }
        // 函数信息
        @Override
        public String getDisplayString(String[] strings) {
        }
}
```

initialize()方法只在 GenericUDF 初始化时被调用一次，执行一些初始化操作，包括：参数个数检查；参数类型检查与转换；确定返回值类型。initialize() 需要 return 一个 ObjectInspector 实例，用于表示自定义UDF返回值类型。**initialize() 的返回值决定了 evaluate() 的返回值类型**。创建ObjectInspector时，不要用new的方式创建，应该用工厂模式去创建。

示例如下：

```
public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {
        int length = arguments.length;
        inputType = new PrimitiveObjectInspector.PrimitiveCategory[length];
        for (int i = 0; i < length; i++) {
            ObjectInspector currentOI = arguments[i];
            ObjectInspector.Category type = currentOI.getCategory();
            //检查参数类型
            if (type != ObjectInspector.Category.PRIMITIVE) {
                throw new UDFArgumentException("The function my_count need PRIMITIVE Category, but
get " + type);
            }
            PrimitiveObjectInspector.PrimitiveCategory primitiveType =
                ((PrimitiveObjectInspector) currentOI).getPrimitiveCategory();
            inputType[i] = primitiveType;
            switch (primitiveType) {
                case INT:
                    if (intConverter == null) {
                        ObjectInspector intOI =
PrimitiveObjectInspectorFactory.getPrimitiveWritableObjectInspector(primitiveType);
                        intConverter = ObjectInspectorConverters.getConverter(currentOI, intOI);
                    }
                    break;
                case LONG:
                    if (longConverter == null) {
                        ObjectInspector longOI =
PrimitiveObjectInspectorFactory.getPrimitiveWritableObjectInspector(primitiveType);
                        longConverter = ObjectInspectorConverters.getConverter(currentOI, longOI);
                    }
                    break;
                default:
                    throw new UDFArgumentException("The function my_count need INT OR BIGINT, but
get " + primitiveType);
            }
        }
        //返回结果类型，需要是wriableObjectInspector，这里表示evaluate返回值类型为LongWritable
        return PrimitiveObjectInspectorFactory.writableLongObjectInspector;
    }
```

evaluate()方法是GenericUDF的核心方法，自定义UDF的实现逻辑。需要注意返回值类型与 initialize() 中定义一致。示例如下：

```
public Object evaluate(DeferredObject[] deferredObjects) throws HiveException {
    LongWritable out = new LongWritable();
    // deferredObjects是
    for (int i = 0; i < deferredObjects.length; i++) {
        PrimitiveObjectInspector.PrimitiveCategory type = this.inputType[i];
        Object param = deferredObjects[i].get();
        switch (type) {
            case INT:
                Object intObject = intConverter.convert(param);
                out.set(Math.addExact(out.get(), ((IntWritable) intObject).get()));
                break;
            case LONG:
                Object longObject = longConverter.convert(param);
                out.set(Math.addExact(out.get(), ((LongWritable) longObject).get()));
                break;
            default:
                throw new IllegalStateException("Unexpected type in MyCountUDF evaluate : " + type);
        }
    }
    return out;
}
```

getDisplayString() 返回的是 explain 时展示的信息。这里不能return null，否则可能在运行时抛出空指针异常。

```
public String getDisplayString(String[] strings) {
    return "my_count(" + Joiner.on(", ").join(strings) + ")";
}
```

7.1.2. 自定义函数加载

1. 将开发好自定义函数的项目打包成jar包，注意：jar 包中的自定义UDF 类名，不能和现有UDF 类，在包名+类名上，完全相同。

2. 使用jar包有两种方式

    1. 将jar放在服务器HDFS某个有权限的目录 `hdfs dfs -put Transwarp-UDF-1.0-transwarp.jar /tmp/test`。再登录beeline使用如下语句添加函数：

```
config query.lang sql;
-- 创建永久函数
CREATE PERMANENT FUNCTION my_count AS 'io.transwarp.inceptor.MyCountUDF' USING JAR
'hdfs:///tmp/test/Transwarp-UDF-1.0-transwarp.jar';
-- 创建临时函数
add jar hdfs:///tmp/test/Transwarp-UDF-1.0-transwarp.jar;
create temporary function my_count as 'io.transwarp.inceptor.MyCountUDF';
```

    2. 将打好的jar包放到quark server镜像中/usr/lib/inceptor/lib目录下。再登录beeline使用如下语句添加函数：

```
-- 创建永久函数
CREATE PERMANENT FUNCTION my_count AS 'io.transwarp.inceptor.MyCountUDF';
-- 创建临时函数
create temporary function my_count as 'io.transwarp.inceptor.MyCountUDF';
```

3. 删除函数

```
config query.lang sql;
-- 删除永久函数
drop permanent function if exists my_count;
-- 删除临时函数
drop temporary function if exists my_count;
-- 清理jar包
deleteres jar hdfs:///tmp/test/Transwarp-UDF-1.0-transwarp.jar;
```

7.2. 自定义存储过程

StellarDB支持用户添加自定义存储过程，封装一组图库操作。

7.2.1. 自定义存储过程实现

自定义存储过程通过java/scala语言开发，需要实现CallableProcedure接口，并编译为jar包，将jar包放置到图库的加载目录通过命令注册到StellarDB中。

CallableProcedure接口定义如下：

```
trait CallableProcedure extends Serializable {
    //存储过程签名，涉及名称、输入、输出、描述等信息
    def signature(): ProcedureType
    //具体执行逻辑
    @throws[CypherException]
    def apply(ctx: CallProcedureContext, input: Array[RDD[_]]): RDD[_]
    //初始化
    def init(ctx: CallProcedureContext): Unit
    //清理资源
    def clean(ctx: CallProcedureContext): Unit
}
```

以下例子定义了一个名为procedures.getnodes的存储过程：

```
class NodeProcedure extends CallableProcedure {
  override def signature(): ProcedureType = {
    ProcedureType.Builder(
      QualifiedName.apply("procedures.getnodes"),  //procedures为命名空间，
bylabel为内部名称。整体构成存储过程的唯一标识
       ProcedureCategory.SIMPLE) //类型：涉及SIMPLE|ALGO|CYPHER
      .inputArgs("label_name", VarStringType) //定义输入参数
      .inputArgs("limit", IntType)  //定义输入参数
      .outputArgs("uid", VarStringType) //定义输出参数
      .description("get_nodes_with_label_procedure") //存储过程描述，方便查阅可以用的存储过程
      .build()
  }
    // 创建rdd，在server端执行
    override def apply(ctx: CallProcedureContext, input: Array[RDD[_]]): RDD[_] = {
    // 获取参数
    val labelName = ctx.inputOI
      .apply(0)
      .asInstanceOf[WritableConstantStringObjectInspector]
      .getWritableConstantValue
      .toString
    val limit = ctx.inputOI
      .apply(1)
      .asInstanceOf[WritableConstantIntObjectInspector]
      .getWritableConstantValue
      .get()
    // 封装RDD作为返回结果，RDD包含处理逻辑
    new RDD[Any](OperatorEnv.sc, Nil) {
      override def compute(split: Partition, context: TaskContext): Iterator[Any] = {
        // CallProcedureContext中获取上下文
        val graph = ctx.getGraph()
        val response = graph.prepareSearch(ctx.getGraphConf())
          .setQuery(
            new VertexQuery().setFilter(
              GraphLabelFilter.build().add(labelName)
            )
          )
          .setLimit(limit)
          .get()
          .asInstanceOf[SearchResponse[Vertex]]
        new Iterator[Any] {
          var curCount = 0
          override def hasNext: Boolean = {
            response.hasNext && curCount < limit
          }
          override def next(): Any = {
            curCount += 1
            val vertex = response.next()
            Array(new Text(vertex.getUserId))
          }
        }
      }
      override protected def getPartitions: Array[Partition] = Array(
        new Partition {
          override def index: Int = 0
        }
      )
    }
  }
  override def init(ctx: CallProcedureContext): Unit = {
    //do nothing
  }
  override def clean(ctx: CallProcedureContext): Unit = {
    //do nothing
  }



// 在apply之后分区执行，在executor端调用
override def partition(ctx: CallProcedureContext, split: Int, iterator: Iterator[_],
sparkTaskContext: TaskContext): Iterator[_] = {
  iterator.map(it => {
    val text = it.asInstanceOf[Array[Text]].head
    new Text(s"uid is ${text.toString}")
  })
}
```

```
    // 在processPartition之后进行处理，在server端调用
    override def collect(ctx: CallProcedureContext, input: RDD[_]): RDD[_] = {
      input.mapPartitionsWithContext((context, partition) => {
        val buffer: ArrayBuffer[String] = ArrayBuffer()
        partition.foreach(p => buffer.append(p.asInstanceOf[Text].toString))
        Iterator(Array(new Text(buffer.mkString(","))))
      })
    }
    // 为true是调用前需要use graph
    override def isQuery: Boolean = true
  }
```

signature() 定义存储过程的元数据信息。通过ProcedureType.Builder的构造器，定义名称、类型、输入和输出、功能描述等信息；

apply() 在server端执行，从CallProcedureContext中获取上下文信息以及常量输入，构造指定label的点的count请求信息进行查询，封装为RDD；

partition() 会将apply中的rdd进行分区处理，在executor端分布式执行，可将一些计算繁重的任务放在这里处理；

collect() 将partition后的数据进行处理，在server端执行，可将统计汇总的任务放在这里处理；

init()和clean()可以初始化、资源清理CallProcedureContext信息。

isQuery()用于判断存储过程是否为查询，如果是则需要调用前use graph

7.2.2. 自定义存储过程加载

1. 将开发好自定义存储过程的项目打包成jar包。
2. 将jar包放置HDFS某个有权限的目录 hdfs dfs -put ./target/stellardb-apoc-1.0.jar /tmp/poc/
3. 登录beeline执行：

```
config query.lang cypher;
use graph twitter;
add jar hdfs:///tmp/poc/stellardb-apoc-1.0.jar;
call procedures.load("procedures.getnodes");
call procedures.getnodes("v", 10);
```

7.3. 自定义图算法

StellarDB支持用户添加自定义图算法，封装一组图库操作。

7.3.1. 自定义图算法实现

自定义图算法通过java/scala语言开发，需要实现GraphAlgoProcedure接口，并编译为jar包，将jar包放置到图库的加载目录通过命令注册到StellarDB中。

GraphAlgoProcedure接口定义如下：

```
  abstract class GraphAlgoProcedure extends io.transwarp.inceptor.execution.crux.CruxGraphAlgorithm
with io.transwarp.crux.procedure.CallableProcedure {
    // 定义图算法名字
    def getAlgorithmName(): scala.Predef.String
    // 定义图算法描述
    def getAlgorithmDescription(): scala.Predef.String
    // 定义图算法输入类型
    def getInputArgs(): scala.Array[scala.Tuple2[scala.Predef.String,
io.transwarp.crux.types.datatype.DataType]]
    /**
      * 定义图算法输出类型，一般图算法输出有两种
      * 1) 和点数据关联，比如每个点返回一个double值。例如Array((node, NodeType), (value,
DoubleType))。目前限制返回结果为一个数组
      * 2) 一个统计值，比如全图返回一个long值。例如Array((value, LongType))
      * @return Array[(name, type)]
      */
    def getOutputArgs(): scala.Array[scala.Tuple2[scala.Predef.String,
io.transwarp.crux.types.datatype.DataType]]
    // 算法结果是否为点相关。如果true，算法应返回N行结果，其中第一列必须是
NodeType类型；如果不是，算法的返回列不应包含NodeType类型
    def isVertexRelatedOutput(): scala.Boolean
    // 是否用有向图进行计算
    def useDirectedGraph(): scala.Boolean
    /**
      * 算法逻辑实现
      *
      * @param args 输入的参数map
      * @param graph 一个紧凑内存结构的图格式
      * @return 算法返回结果，需要是基础类型: int/long/double/string或者Array[int/long/double/string]
      */
    def runAlgorithm(args: java.util.HashMap[scala.Predef.String, java.lang.Object], graph:
io.transwarp.eagle.graph.V5EagleGraph[io.transwarp.eagle.graph.V5EagleVertex]): scala.Any
  }
```

其中实现算法逻辑的runAlgorithm方法，使用V5EagleGraph[V5EagleVertex]类型传入图数据。

其中V5EagleVertex是点数据类型，id为int类型，每个V5EagleVertex实例存储了边信息包括邻居id和权值，可通过如下接口访问点数据：

```
class V5EagleVertex(vid: scala.Int, var inDegree: scala.Int = ???, var outDegree: scala.Int = ???,
var inEdges: io.transwarp.eagle.graph.EdgeBlock = ???, var outEdges:
io.transwarp.eagle.graph.EdgeBlock = ???, var inOffset: scala.Long = ???, var outOffset: scala.Long
= ???, var weight: scala.Double = ???) extends io.transwarp.eagle.graph.EagleVertex {

  override def getVid: Int = ???

  override def getOutDegree: Int = ???

  override def getInDegree: Int = ???

  override def getOutDegreeWithoutSelfLoop(isWeighted: Boolean): Int = ???

  override def addOutEdge(oe: scala.Int): scala.Unit = ???

  override def addInEdge(ie: scala.Int): scala.Unit = ???

  override def getOutEdges(isWeighted: scala.Boolean):
scala.collection.mutable.ArrayBuffer[scala.Int] = ???

  override def getOutEdgeIterator(isWeighted: scala.Boolean):
scala.Iterator[io.transwarp.eagle.graph.NonBoxedInt] = ???

  override def getOutEdgeWeightIterator(isWeighted: scala.Boolean):
scala.Iterator[io.transwarp.eagle.graph.NonBoxedVidWeight] = ???

  override def getInEdgeWeightIterator(isWeighted: scala.Boolean):
scala.Iterator[io.transwarp.eagle.graph.NonBoxedVidWeight] = ???

  override def getInEdges(isWeighted: scala.Boolean):
scala.collection.mutable.ArrayBuffer[scala.Int] = ???

  override def getInEdgeIterator(isWeighted: scala.Boolean):
scala.Iterator[io.transwarp.eagle.graph.NonBoxedInt] = ???

  /**
   * 对出边做计算，下一轮可计算的计算会写入outEdges
   *
   * @param offset 结果集outEdges中的偏移量
   * @param outEdges 结果集，如果为null，则不产生下一轮计算节点
   * @param algorithm 算法
   * @param isWeighted 是否为权重图
   * @param reuseEdgeStruct 用于解码的复用数据结构
   */
  override def computeOutNghSparse(offset: scala.Int, outEdges: scala.Array[scala.Int], algorithm:
io.transwarp.eagle.algorithm.Algorithm, isWeighted: scala.Boolean, isReverse: scala.Boolean,
reuseEdgeStruct: io.transwarp.eagle.utils.EdgeEncodeStruct): scala.Unit = ???
  /**
   * 对入边做计算，下一轮可计算的节点会在nextVertices中标记
   *
   * @param vSet 待计算的点集
   * @param algorithm 算法
   * @param nextVertices 结果集，如果为null，则不产生下一轮计算节点
   * @param parallel 是否并行计算
   * @param isWeighted 是否为权重图
   * @param reuseEdgeStruct 用于解码的复用数据结构
   * @param isForward 用于判定是否要过滤起点
   */
  override def computeInNghDense(vSet: io.transwarp.eagle.graph.VertexSubset, algorithm:
io.transwarp.eagle.algorithm.Algorithm, nextVertices: scala.Array[scala.Boolean], parallel:
scala.Boolean = ???, isWeighted: scala.Boolean, isReverse: scala.Boolean, reuseEdgeStruct:
io.transwarp.eagle.utils.EdgeEncodeStruct, isForward: scala.Boolean): scala.Unit = ???
  /**
   * DenseForward模式下计算出边
   *
   * @param nextVertices 结果集，如果为null，则不产生下一轮计算节点
   * @param algorithm 算法
   * @param isWeighted 是否为权重图
   * @param reuseEdgeStruct 用于解码的复用数据结构
   * */
  override def computeOutNghDense(nextVertices: scala.Array[scala.Boolean], algorithm:
io.transwarp.eagle.algorithm.Algorithm, isWeighted: scala.Boolean, isReverse: scala.Boolean,
reuseEdgeStruct: io.transwarp.eagle.utils.EdgeEncodeStruct): scala.Unit = ???

  override def hasSelfLoops(isWeighted: scala.Boolean): scala.Boolean = ???
}
```

V5EagleGraph提供了如下接口访问图中的数据：

```
class V5EagleGraph[T <: io.transwarp.eagle.graph.EagleVertex](_vn: scala.Int, _en: scala.Long, var
_vertices: scala.Array[T], val mapping: io.transwarp.eagle.graph.V5VertexMapping, _isWeighted:
scala.Boolean, _loops: scala.Array[scala.Int] = ???, conCurr: scala.Int = ???) extends
io.transwarp.eagle.graph.EagleGraph[T] {
  // 点数量
  override def getVertexNumber: scala.Int = ???
  // 边数量
  override def getEdgeNumber: scala.Long = ???
  // 根据点id查找
  def getVertex(idx: scala.Int): T = ???
  // 获取所有点
  def getVertices: scala.Array[T] = ???
  // 是否存在自环
  override def hasSelfLoops: scala.Boolean = ???
  // 是否为无向图
  override def isSymmetric: scala.Boolean = ???
  // 转置图
  override def transpose(): scala.Unit = ???
  // 获取全图边权重和
  override def getGraphWeight: scala.Long = ???
  // 获取图中的环
  override def getLoops: scala.Array[scala.Int] = ???
  // 是否有权重
  override def isWeighted: scala.Boolean = ???

}
```

以下例子定义了一个名为my_pagerank的图算法：

```
class UserDefinedPageRankProcedure extends GraphAlgoProcedure {

  // 定义图算法名字。通过call调用的完整前缀是procedures.algo.my_pagerank()
  override def getAlgorithmName(): String = "my_pagerank"

  override def getAlgorithmDescription(): String = "pagerank defined by user"

  override def getInputArgs(): Array[(String, DataType)] =
    Array(
      ("rounds", IntType),
      ("factor", DoubleType)
    )

  override def getOutputArgs(): Array[(String, DataType)] =
    Array(
      ("node", NodeType),
      ("rank", DoubleType)
    )

  override def isVertexRelatedOutput(): Boolean = true

  override def useDirectedGraph(): Boolean = true

  override def runAlgorithm(args: java.util.HashMap[String, Object], graph:
V5EagleGraph[V5EagleVertex]): Any = {
    // 获取参数
    val rounds = args.get("rounds").asInstanceOf[Int]
    val factor = args.get("factor").asInstanceOf[Double]

    val curRank = Array.fill(graph.getVertexNumber)(0.0)
    val nextRank = Array.fill(graph.getVertexNumber)(0.0)

    var iter = 0

    while (iter < rounds) {
      val danglingSum = computeDanglingValue(graph, curRank)

      val func1 = (vertex: V5EagleVertex) => {
        var j = 0
        vertex.getOutEdges(isWeighted = false)
          .foreach(dst => {
            nextRank(dst) += curRank(vertex.getVid) / vertex.getOutDegree
          })
      }
      // 传递rank值
      vertexMap(graph, func1)

      val residue = (1 - factor) / graph.getVertexNumber
      val dangling = factor / graph.getVertexNumber * danglingSum

      val func2 = (vertex: V5EagleVertex) => {
        nextRank(vertex.getVid) = residue + factor * nextRank(vertex.getVid) + dangling
      }
```

```
      // 更新rank值
      vertexMap(graph, func2)

      var curEpsilon = 0D
      val func3 = (vertex: V5EagleVertex) => {
        val k = vertex.getVid
        curEpsilon += Math.abs(curRank(k) - nextRank(k))
        curRank(k) = nextRank(k)
        nextRank(k) = 0
      }
      // 交换rank值
      vertexMap(graph, func3)

      if (curEpsilon < 0.00001) {
        iter = rounds
      }
      iter += 1
    }

    curRank
  }

  private def vertexMap(graph: V5EagleGraph[V5EagleVertex], func: V5EagleVertex => Unit): Unit = {
    var i = 0
    while (i < graph.getVertexNumber) {
      func(graph.getVertex(i))
      i += 1
    }
  }

  private def computeDanglingValue(graph: V5EagleGraph[V5EagleVertex],
                                   curRank: Array[Double]): Double = {
    var result = 0.0D

    var i = 0
    while (i < graph.getVertexNumber) {
      val vertex = graph.getVertex(i)
      if (vertex.getOutDegree == 0) {
        result += curRank(i)
      }
      i += 1
    }

    result
  }
}
```

7.3.2. 自定义图算法加载

1. 将开发好自定义图算法的项目打包成jar包。
2. 将jar包放置HDFS某个有权限的目录 `hdfs dfs -put ./target/stellardb-apoc-1.0.jar /tmp/poc/`
3. 登录beeline执行：

```
config query.lang cypher;
use graph twitter;
add jar hdfs:///tmp/poc/stellardb-apoc-1.0.jar;
call procedures.load("procedures.algo.my_pagerank");
call procedures.algo.my_pagerank(10, 0.85) yield node, rank;
```

8. StellarDB运行与维护

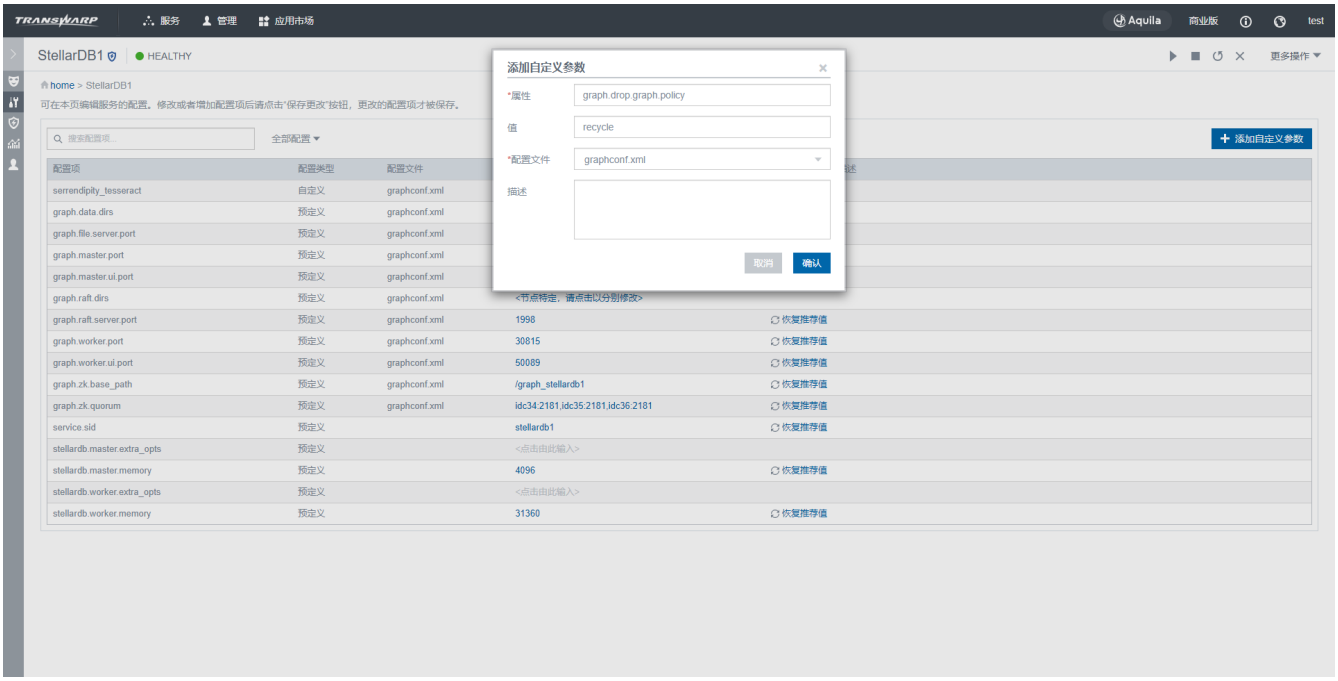
8.1. 回收站功能

StellarDB支持通过配置集群，使执行删除图 `drop graph` 时不立即清除图数据，而是将其隐藏，类似于放入回收站中，避免运维中操作失误引起数据丢失。 数据库管理员可以在之后对回收站中的 `graphs` 执行恢复或删除两种操作。

- 支持在删除某图后立即创建同名图，与回收站中的图互不干扰。
- 回收站中可暂存同名图。可以通过查看其被删除时间和 `graphId` 来区分同名图。

8.1.1. 禁用/启用回收站功能

回收站功能默认开启，禁用功能需要在Transwarp Manager界面的StellarDB服务中添加自定义参数 `graph.drop.graph.policy` 。配置项默认值为 `recycle` ，表示启用回收站功能；若设置为 `direct` ，则表示禁用回收站功能。



8.1.2. 查看回收站内容

启用回收站功能之后，删除的图将会被移动到回收站。可以通过以下语句查看回收站中的内容。

```
manipulate graph <graphName | _all> show recycle_bin;
```

示例1 – 列举回收站中所有图名为g1的图信息

```
manipulate graph g1 show recycle_bin;
```

示例2 – 列举回收站中所有的图信息

```
manipulate graph _all show recycle_bin;
```

返回值示例：

```
[
    {
        "graphName":"g1",
        "deleteTime":"2021-01-20 16:42:42",
        "graphId":630001
    },
    {
        "graphName":"g2",
        "deleteTime":"2021-01-20 16:42:59",
        "graphId":630002
    }
]
```

- `graphName` 为回收站中的图名。
- `deleteTime` 为图被移入回收站的时间。
- `graphId` 为图的标识ID，需要在删除或恢复回收站中的图时使用。

8.1.3. 从回收站彻底删除图

在beeline客户端中执行如下命令从回收站彻底删除图。

```
manipulate graph <id=<graphId> | _all> drop_recycle_bin;
```

`graphId` 为“查看回收站内容”的返回值字段。

示例1 – 删除回收站中指定的图

```
manipulate graph id=630001 drop_recycle_bin;
```

示例2 – 清空回收站所有图

```
manipulate graph _all drop_recycle_bin;
```

返回值示例：

```
{
    "message":"Drop success.",
    "isSuccess":true
}
```

- `isSuccess` 为操作是否成功。
- `message` 在操作失败时会给出失败提示与修复建议。

回收站中的图删除后无法恢复，删除前请确认。

8.1.4. 从回收站恢复图

在beeline中执行如下命令从回收站恢复图。

```
manipulate graph id=<graphId> recover_recycle_bin;
```

`graphId` 为“查看回收站内容”的返回值字段。

示例：

```
manipulate graph id=630001 recover_recycle_bin;
```

返回值示例：

```
{
     "message":"Recover success.",
     "isSuccess":true
}
```

- `isSuccess` 为操作是否成功。
- `message` 在操作失败时会给出失败提示与修复建议。

执行了上述语句恢复图之后，还需要在Quark手动创建图的元信息（与图备份恢复中的额外操作相同），才可使用本图：

```
// 切换到SQL语言
config query.lang sql;
// 创建图的元信息
create database <graphName>;
use <graphName>;
create table graph stored as stellardb;
// 切换回Cypher语言
config query.lang cypher;
```

当数据库存在同名的可用图时，不能从回收站恢复图。

8.2. 图备份与图恢复

图数据库支持通过TEoC进行图全量备份、增量备份与图恢复。备份功能需要StellarDB用户对HDFS的写入权限。如开启Guardian，需给予StellarDB权限。若Guardian无StellarDB用户，需新创建用户并更新 `stellardb.keytab`。单机版StellarDB无HDFS，需要将配置项 `graph.is.standalone.mode` 置为 `true`。

8.2.1. 进行图全量备份

在beeline中执行 `manipulate graph … backup full` 进行图全量备份，StellarDB 5.0.0版本中不再支持 `load graph … into` 语句进行备份。StellarDB会尝试在给定HDFS目录进行图当前状态快照的备份。

图全量备份的语法规则如下所示：

```
manipulate graph <graph_name> backup full "<hdfs_path>";
```

调用示例：

```
manipulate graph test_graph backup full "/backup/graph/test_graph/full_1/";
```

图全量备份注意事项：

- `hdfs_path` 若指定路径已存在，则指定路径的目录 **必须为空**，若指定目录不存在，Quark server自动创建该路径，专用于此次备份；
- StellarDB Worker与Quark Executor可以运行于相同物理机或不同物理机；
- 若存在位于同一台机且IP地址相同的Worker与Executor，图备份功能将采用本地拷贝。节约网络资源、加速备份。但此时要求Quark Executor容器被配置为可直接访问StellarDB Worker的数据目录与raft目录（配置项 `graph.data.dirs` 与 `graph.raft.dirs` ）

8.2.2. 进行图增量备份

在beeline客户端中执行如下命令进行图增量备份。StellarDB会尝试根据过去保存的图备份数据，进行从上一次备份到当前状态的 **差值** 的备份。

增量备份语法规则如下所示：

```
manipulate graph <graph_name> backup inc "<hdfs_path>";
```

调用示例如下所示：

```
manipulate graph test_graph backup inc "/backup/graph/test_graph/inc_2/";
```

图增量备份注意事项：

- `hdfs_path` 指定路径的目录应当为空且已经创建，专用于此次备份；
- 图的首次备份必须是全量备份。增量备份之前必须要有过去的备份数据。

8.2.3. 进行图恢复

1. 在beeline客户端中执行 `manipulate` 命令进行图恢复。

   图恢复语法规则如下所示：

   ```
   manipulate graph <graph_name> recover "<hdfs_path1>","<hdfs_path2>","<hdfs_path3>",...;
   ```

   调用示例：

   ```
   manipulate graph new_graph recover
   "/backup/graph/test_graph/full_1/","/backup/graph/test_graph/inc_2/";
   ```

   图恢复注意事项：

   - 图恢复时指定的图名不能已经存在于StellarDB集群，应该是新的图名；
   - 恢复时，可以指定与备份时不同的图名，来改变恢复得到的图的名字；
   - 恢复时，shard数目不可改变；
   - 若给出多个HDFS路径参数，则执行“增量恢复” — 系统将尝试使用所有路径中的备份数据，将图恢复到最后一个路径参数对应的备份时的状态； 若只给出一个HDFS路径参数，则会检查路径下的文件 — 若符合单次全量备份的格式，则恢复图到备份数据状态；否则，将此目录的每个子目录视为一个备份目录，采用“多个HDFS路径”的逻辑尝试检查、恢复图。 所以，如果全量备份后面的增量备份次数太多，可以通过将所有备份目录放入同一个目录，并指定这个共同的父目录进行恢复，来避免命令参数列表过长的问题。

2. 图恢复成功后，图对用户不可见（即执行 `show stellargraphs;` 不显示图名），需要执行图连接操作。在beeline客户端中执行下述命令进行图连接。

   ```
   config query.lang sql;
   create database <restored_graph_name>;
   use <restored_graph_name>;
   create table graph stored as stellardb;
   ```

   图恢复注意事项：

   - `restored_graph_name` 为恢复时指定的图名；
   - 图恢复功能 **不支持** HA。许多种集群故障，包括任何一块磁盘失效，都会导致恢复失败。

8.2.4. 查看图备份信息

在beeline客户端中执行如下命令查看图备份信息。通过该命令，可以查看图的每一次备份的信息，包括备份序号，备份类型，备份时间，StellarDB存储路径等。其中，备份序号主要用于在删除备份数据时指定备份数据；备份类型分为：全量备份（FULL）和增量备份（INCREMENTAL）。

查看图备份信息语法规则如下所示：

```
manipulate graph <graph_name> show backups;
```

调用示例如下：

```
manipulate graph test_graph show backups;
```

8.2.5. 删除全部备份数据

在StellarDB客户端中执行如下命令删除某图全部备份数据。执行该指令将会清除指定图的所有备份信息，并尝试删除备份对应的HDFS数据。

删除全部备份数据语法规则如下所示：

```
manipulate graph <graph_name> drop backups all;
```

调用示例如下所示：

```
manipulate graph test_graph drop backups all;
```

8.2.6. 删除指定备份数据

在beeline客户端中执行如下命令删除某图部分备份数据。系统将依次处理给出的备份序号，删除每个序号对应的备份以及序号之后连续的增量备份信息，并尝试删除备份对应的HDFS数据。

删除指定备份数据的语法规则如下所示：

```
manipulate graph <graph_name> drop backups <backup_index1>,<backup_index2>,...;
```

调用示例如下：

```
manipulate graph test_graph drop backups 1,5;
```

8.2.7. 删除失败的恢复目录

当恢复图失败时，会提示需要手动清除失败的恢复目录，可通过执行下述指令清除。

```
manipulate graph <graph_name> scrap_offline_graph;
```

8.3. Shard数据副本迁移

由于数据规模的增长，StellarDB Worker可能会在硬盘上存储了过多的shard数据。如果硬盘被写满，会导致此shard进入异常状态，不可再接受写操作。在硬盘空余容量用完之前（可以参考KG Explorer的监控信息或Aquila的告警信息），就应当规划集群扩容与shard数据副本的迁移。

StellarDB提供数据副本迁移功能，可以将当前某Worker上的shard副本转移到另一台Worker上。目前版本只能指定原Worker与目标Worker，不能精确指定目标为具体某一块硬盘。

8.3.1. 执行副本迁移

在beeline客户端中执行如下命令以执行副本迁移。

副本迁移语法规则如下：

```
manipulate graph <graphName> move_shard <shardId> from <fromHostName> to <toHostName>;
```

其中：

- graphName，为需迁移副本的图名。
- shardId，为需迁移副本的shard ID。
- fromHostName，为旧副本位于的Worker主机名。
- toHostName，为数据副本应迁往的目标Worker主机名。

调用示例如下：

```
manipulate graph g1 move_shard 3 from tdh01 to tdh02;
```

返回值示例：

```
{
      "isSuccess": true,
      "message": "Success!"
 }
```

其中：

- isSuccess，表示操作是否执行成功。
- message，表示操作结果错误信息。当操作失败时，本字段会给出相关提示。

创建新副本注意事项：

1. 迁移操作中，新副本会从老的健康副本通过网络进行shard数据文件的传输，受数据量与网速影响，执行时间可能较长；而且由于涉及到shard各副本的状态切换等过程，即使图中的数据量很少，单次迁移也可能等待需要30秒以上的时间。
2. 本指令会同步等待迁移操作完成，最多等待300秒。若迁移指令等待超过300秒，则会报出超时错误，但这不一定意味着迁移失败，而是可能还在进行中；如果数据量很大，且此shard的leader节点对迁移目标节点还有大量网络发送，则可能迁移操作仍在进行中。可以等待数据传输结束，然后再等待约1分钟后，在KG Explorer上查看shard状态，若shard状态健康，且能查看到副本位置已经变动成功，则说明shard迁移操作成功。

8.4. 故障shard副本修复

由于磁盘文件损坏、突然断电等异常，StellarDB的shard副本可能会遭到损坏而无法正常响应读写请求。StellarDB采用shard多副本来避免数据完全损坏丢失。确认数据受损情况之后，可以通过TEoC语句，命令StellarDB进行受损shard的删除与shard新副本的重建，使shard的副本变为正常状态，恢复图的健康状态。

StellarDB默认配置为三副本。StellarDB多副本机制，支持在副本数少于一半时（比如，三副本损失一个，五副本损失两个），依然能够支持可读可写。

当出现shard状态异常时，除了排查日志，还可以通过依次重启故障Worker与ActiveMaster来判断故障类型。重启完成后，根据StellarDB Web页面的shard状态提示中副本数目与状态，可以将故障大致分为“目录丢失”与“文件损坏”两种。如果故障Worker的副本没有显示在副本列表中，则为“目录丢失”；若显示在列表中（且状态为 **STRAGGLER**），则为“文件损坏”。

对于“文件损坏”，应该对目标shard依序执行 **“删除副本 remove_rep”**、**“创建新副本 add_observer”** 操作；对于“目录丢失”，应该对目标shard依序执行 **“删除副本元信息 remove_rep_meta”**、**“创建新副本 add_observer”** 的操作。

本章节的操作不建议用户在没有支持人员指导的情况下使用。如果发现有shard副本出现故障，请先咨询StellarDB支持人员确认问题。

8.4.1. shard副本数损失少于一半的情况

8.4.1.1. 创建新副本

在beeline客户端中执行如下命令以创建新副本。

创建新副本语法规则如下：

```
manipulate graph <graphName> add_observer <shardId> at <hostname>;
```

其中：

- graphName，为需创建新副本的图名。
- shardId，为需创建新副本的shard ID。
- hostname，为新副本位于的Worker的主机名，需要与此shard当前副本所在的Worker不重复。

```
manipulate graph g1 add_observer 0 at tdh01;
```

返回值示例：

```
{
      "isSuccess": true,
      "message": "Success!"
 }
```

其中：

- isSuccess，表示操作是否执行成功。
- message，表示操作结果错误信息。当操作失败时，本字段会给出相关提示。

副本创建成功后，新副本会从老的健康副本异步进行shard数据文件的传输，所以需要等待一段时间之后，shard副本才会变为健康状态。

8.4.1.2. 删除副本/删除副本元信息

在beeline客户端中执行如下命令删除shard副本或副本元信息。

语法规则如下：

1. 删除副本

```
manipulate graph <graphName> remove_rep <shardId> at <hostname>;
```

2. 删除副本元信息

```
manipulate graph <graphName> remove_rep_meta <shardId> at <hostname>;
```

其中：

graphName，为出现故障的图名。

shardId，为出现故障的shard ID。

hostname，为故障副本所在的Worker主机名。

调用示例：

```
manipulate graph g1 remove_rep 0 at tdh01;
manipulate graph g1 remove_rep_meta 0 at tdh01;
```

返回值示例：

```
{
      "isSuccess": true,
      "message": "Success!"
 }
```

其中：

- isSuccess，表示操作是否执行成功。
- message，表示操作结果错误信息。当操作失败时，本字段会给出相关提示。

8.4.2. shard副本数损失大于一半的情况

8.4.2.1. 从shard正常节点复制到目标节点

在beeline客户端中执行如下命令复制shard副本。

语法规则如下：

```
manipulate graph <graphName> copy_shard <shardId> from <sourceHostName> to <targetHostName>;
```

其中：

- graphName，为需要修复的图名。
- shardId，为需要修复的shardId。
- sourceHostName，为健康副本位于的Worker的主机名。
- targetHostName，为新副本位于的Worker的主机名。

返回示例

```
{
        "jobId":[
                        "CopyShardFiles_graphName_0_2020-12-15T03:14:00.831Z_51"
        ]
 }
```

- jobId，为返回的任务Id，以供查询和记录。

8.4.2.2. 查询复制任务状态

```
manipulate graph <graphName> copy_shard_status <jobId> from <targetHostName>;
```

其中：

- graphName，为需要修复的图名。
- jobId，为复制任务的id。
- targetHostName，为新副本位于的Worker的主机名。

返回示例

```
{
      "message": "FINISH",
      "isSuccess": true
 }
```

- message，为返回的任务当前状态。
- isSuccess，为查询执行是否成功。

8.4.2.3. 手动删除复制任务状态缓存

```
manipulate graph <graphName> copy_shard_status clean from <targetHostName>;
```

其中：

- graphName，为需要修复的图名。
- targetHostName，为新副本位于的Worker的主机名。

返回示例

```
{
      "message": "clean over at targetHostName",
      "isSuccess": true
 }
```

- message，为清理状态。
- isSuccess，代表执行结果。

8.5. 硬盘写满异常处理方式

在遭遇各种I/O异常时，StellarDB的shard副本将会进入“I/O异常暂停(IOE-Paused)”状态，拒绝写入操作以避免触发更多异常。相比StellarDB 3.0版本中shard遭遇I/O异常就会下线的行为，在StellarDB 4.0的此状态下，shard副本将不支持读写，但是保持在线。如果使用日志排查出了I/O异常的原因，并加以恢复，那么可以通过TEoC指令解除异常状态，实现在不重启服务的情况下让图恢复健康。

8.5.1. 解除shard的IOE-Paused状态

首先，排查StellarDB日志并手动解决导致I/O异常的问题，然后在beeline中执行如下命令以解除shard副本的 IOE-Paused 状态。

语法规则如下：

```
manipulate graph <graphName> resolve_ioe <shardId> <rebuildIndex=<true | false>>;
```

其中：

- graphName，为需恢复shard副本状态的图名。
- shardId，为需恢复状态的shard ID。
- rebuildIndex，参数为可选，表示状态恢复后是否自动对此shard启动一次索引重建；如果忽略此参数，则不启动索引重建。

调用示例如下：

```
manipulate graph g1 resolve_ioe 0 rebuildIndex=true; // 尝试使shard 0的所有副本解除IOE-Paused
状态，并启动shard 0的索引重建
manipulate graph g1 resolve_ioe 1; // 尝试使shard 1的所有副本解除IOE-Paused
```

返回值：

- 如果不启动索引重建，则返回 success。
- 如果启动索引重建，则返回 Successfully resolved IOE-Paused status. Starting index rebuilding: ….。

解除IOE-Paused状态注意事项：

1. 当shard遭遇I/O异常时，如果是正在插入数据，那么可能会导致数据与索引不一致的情况出现。在这种情况下，解除 `IOE-Paused` 状态后应该进行一次shard索引重建。本TEoC操作可以同时启动索引重建任务，默认对点与边的索引都进行重建。如果能够确认图遭遇I/O异常时并非在做数据插入，那么可以取消索引重建的步骤，或者之后手动执行仅包含点或边的索引重建。
2. 应当确保导致I/O异常的系统故障被排除之后，再执行上面的TEoC命令来尝试恢复。否则，shard可能会再次遭遇I/O异常，再次状态转为 `IOE-Paused` .
3. 在少数副本变为 `IOE-Paused` 状态时，shard仍然可以支持前面小节中的“shard副本迁移”与“故障shard副本修复”操作。如果I/O异常难以恢复，则可以放弃此副本，采用这两种操作将副本转移到其他状态良好的节点。

8.6. 图存储配额限制

用户可以为图设置存储配额，限制图的占用硬盘空间（所有副本累计的存储空间总和）。当图占用的空间达到限制值时，除非重新设置图的存储配额，否则尝试向此图进行插入/更新/bulkload时会报错、拒绝操作。本功能依赖于StellarDB周期性的图存储用量统计；为了避免统计影响性能，StellarDB的存储用量统计任务调度频率较低，存储用量数据可能最多有5分钟的延迟。

设置StellarDB服务的配置项 `graph.client.storage.limit.check.enabled` 可控制本功能是否启用。默认值为 `true`，即启用检查功能；若改为 `false` 则可禁用本功能。

StellarDB支持两种配额设置方式：“StellarDB native配额设置”与“Guardian端配额设置”。两者起到的效果相同，都能在图占用空间过大时起到限制作用；但两种方法需要使用各自不同的方式进行设置或取消。

8.6.1. StellarDB native存储配额设置

通过下列TEoC语句可以设置指定图的native存储配额。

语法规则如下：

```
manipulate graph <graphName> set_space_quota <limitValue> [unit];
```

其中：

- graphName，为设置native存储配额图名。
- limitValue，为限制值，应当为非负数；当单位为MB/MiB/GB/GiB时可以为小数，其他单位支持整数。
- unit，为限制值的单位，参数可选（默认为B），支持的单位包括B/KB/KiB/MB/MiB/GB/GiB。

调用示例如下：

```
manipulate graph g1 set_space_quota 50 MiB; // 将g1的native存储空间配额设置为50 MiB
manipulate graph g2 set_space_quota UNLIMITED; // 取消g2的native存储空间配额限制
```

通过下列TEoC语句可以查看指定图当前的native存储配额。

```
manipulate graph <graphName> get_space_quota;
```

需要注意的是，该语句的执行结果显示为整数，单位为字节；未设置过限制/已经消除配额限制的图，结果为"UNLIMITED"。

8.6.2. Guardian存储配额设置

当StellarDB服务启用了Guardian插件时，可以使用Guardian页面进行图的存储配额的设置。 只需在Guardian的Quark服务配置页面，进入“存储配额”选项卡，然后执行“添加配额”，为与StellarDB目标图同名的Quark数据库设置存储配额即可。

8.6.3. 查询指定图的存储占用量

StellarDB通过周期性统计更新存储端各图的硬盘占用量信息，并将此值与通过上面的方法设置的配额限制进行对比，决定是否拒绝写操作。由于统计硬盘占用信息也会占用一定量的计算资源，所以StellarDB每隔几分钟才会重新统计；存储用量的信息并非实时，可能最多会有5分钟的延迟。

通过下列TEoC语句可以查看指定图的存储用量信息。

```
manipulate graph <graphName> get_storage_usage;
```

返回示例：

```
{
  "FullSize(B)":77574,
  "RaftSize(B)":71321,
  "IndexSize(B)":1231,
  "SegmentSize(B)":5022,
  "FullSize(KiB)":75
}
```

返回结果中：

- FullSize，为图个部分总共占用存储空间的量，为下面几部分之和。默认单位为B（字节），如果FullSize足够大，会自动增加更大单位的计算值显示（此处自动转换为了KiB）。
- SegmentSize，为数据存储文件占用空间。
- IndexSize，为数据索引文件占用空间。
- RaftSize，为数据写操作日志占用空间。

9. StellarDB数据安全

9.1. 图数据静态加密

StellarDB支持对存储数据进行静态加密。即使数据磁盘失窃，或系统遭到入侵而发生数据文件泄漏，StellarDB也可通过数据静态加密功能保障数据安全。当前版本中，StellarDB支持两种数据静态加密方法，即通过数据文件特殊编码方式和加密算法加密。

9.1.1. StellarDB数据文件编码

StellarDB数据存储使用一种独有的数据编码格式，在压缩数据的同时，使数据文件无法被StellarDB类库以外的用户解析，从而保证数据安全。

9.1.2. 对称加密功能

StellarDB也支持对底层数据使用标准对称加密算法进行自动加密，目前支持 **国标SM4分组加密算法**。需要注意的是，启用加密算法对数据进行加密会导致数据库读写性能下降。

使用加密算法对数据进行加密，通常需要在 **创建图schema** 时，在图schema中通过添加参数 `"graph.encryption.type": "SM4_CTR"` 来对图使用对称加密功能。若不设置该参数或设置该参数为 `"graph.encryption.type": "NO_ENCRYPTION"`，数据将不进行加密。

使用JSON字符串建图示例：

```
{
  "graph.name": "encrypted_graph",
  "graph.encryption.type": "SM4_CTR",
  ...
}
```

9.2. 数据脱敏

StellarDB可以通过配置并且导入规则表来实现对数据的脱敏。

9.2.1. 规则表结构

表 2. 规则表结构

图属性名	属性对应的值	用户名	需要配置掩码的图属性	掩码方式
department	营业一部	Tom	phone_number	mark(custom,[4,8])

- 上表表示，用户Tom访问该图，如果图属性department对应的值是“营业1部”，则对应的phone_number可以全部展示，如果department值是其他值，则以掩码显示；
- 如果表中第一列的属性在图中不存在，对于用户Tom，phone_number字段正常显示；
- 如果用户不在规则表中，该用户可以正常访问图中phone_number字段。

9.2.2. 配置掩码流程

1. 创建规则表

```
config query.lang sql;
create table xxx(a1 string,a2 string, a3 string,a4 string,a5 string);
```

2. 插入数据

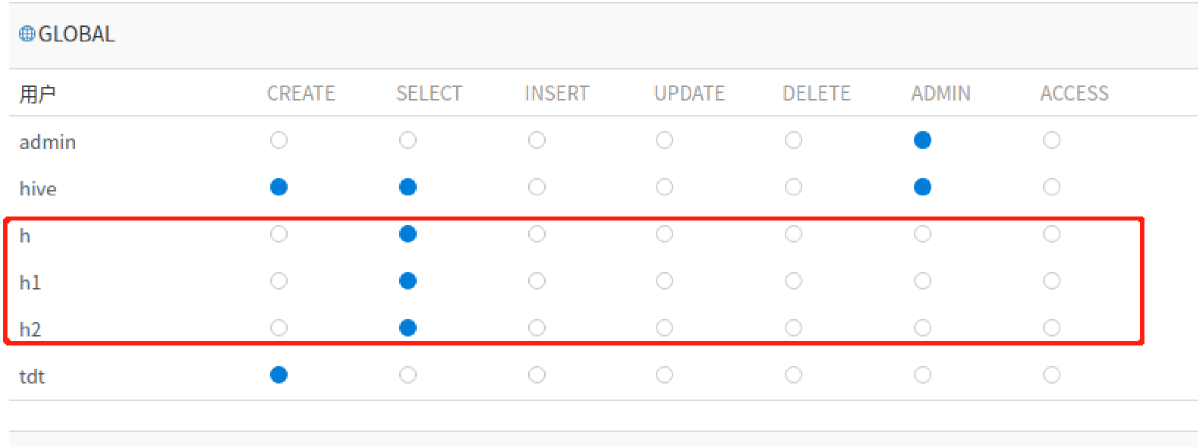
```
insert into table xxx values(…);
```

3. 导入规则

```
config query.lang cypher;
LOAD DESENSITIZATION RULES FROM TABLE [db_name].[tbl_name] INTO GRAPH [graph_name];
```

9.2.3. 参考案例

1. 在Guardian界面追加使用图谱的用户，给予 `GLOBAL` 的读权限即可（也可以按需配置指定图的 `select` 权限）



2. 通过有 `admin` 权限的用户构建图。（此处示例图为desens_wiki）

```
0: jdbc:hive2://vqa36:10008/default> use graph desens_wiki;
+--------+
| _a_c0  |
+--------+
+--------+
No rows selected (0.016 seconds)
0: jdbc:hive2://vqa36:10008/default> match (a) return a;
+------------------------------------------------------------------------------------------------------------------------------+
|                                                              a                                                               |
+------------------------------------------------------------------------------------------------------------------------------+
| {"entityKey":[50,0,0],"properties":{"__tags":[],"url1":"bb","__uid":"2","department":"abc","attr":2,"url":"jd"},"labels":["PAGE"]}  |
| {"entityKey":[51,0,0],"properties":{"__tags":[],"url1":"ccc","__uid":"3","department":"abc","attr":3,"url":"tb"},"labels":["PAGE"]}  |
| {"entityKey":[49,0,0],"properties":{"__tags":[],"url1":"a","__uid":"1","department":"abc","attr":1,"url":"baidu"},"labels":["PAGE"]}  |
| {"entityKey":[52,0,0],"properties":{"__tags":[],"url1":"dddd","__uid":"4","department":"transwarp","attr":4,"url":"qq"},"labels":["PAGE"]}  |
+------------------------------------------------------------------------------------------------------------------------------+
4 rows selected (8.131 seconds)
0: jdbc:hive2://vqa36:10008/default>
```

3. 通过有 `admin` 权限的用户构建脱敏规则表。

   a. 所有的脱敏规则表一共有下面5列（均为 `string` 类型），需要严格按照顺序定义，列名无限制：

      i. 图属性名(基于脱敏的属性)

      ii. 基于脱敏属性的值(属性对应的值)

      iii. 用户名

      iv. 需要配置掩码的图属性(需要脱敏的属性)

      v. 掩码方式

   b. 掩码方式当前提供两类：

      i. mark(all)，对于需要脱敏的属性，将所有的值置为*。

      ii. mark(custom,[start_num,end_num])，对于需要脱敏的属性，将以start_num到end_num中间的值置为*。

   c. 这里示例的表为desen_table，所属数据库为desen：

```
0: jdbc:hive2://vqa36:10008/default> select * from desens.desens_table;
+----------------+----------------------+-----------+--------------+------------------------+
| group_by_attr  | group_by_attr_value  | username  | desens_prop  |      desens_rule       |
+----------------+----------------------+-----------+--------------+------------------------+
| department     | transwarp            | h         | url          | mark(all)              |
| department     | abc                  | h1        | url1         | mark(custom,[0,1])     |
+----------------+----------------------+-----------+--------------+------------------------+
2 rows selected (4.769 seconds)
```

      i. 图中h和h1用户，和guardian界面创建的用户对应。

      ii. 基于department属性，将h和h1进行分组，h属于department为transwarp的组，h1属于department为abc的组。

      iii. 基于deparment属性，有两个掩码方式：

A. 针对属性为url的掩码方式mark(all)，表示对于属于不同department的用户，他们查看到的url属性全为*

B. 针对属性为url1的掩码方式mark(custom.[0,1])，表示对于属于不同department的用户，他们查看到的url1属性的第0位为*

> 针对脱敏规则表有两个限制:
>
> 限制1：一个用户针对 **图属性名** 只能有一个 **属性对应的值**。对应表中则是用户h对department属性只能有一个值transwarp。如果还有用户h对department属性还有另外一个值，那么在执行下面步骤LOAD时会报错。
>
> 限制2：针对 **图属性名**，每个 **需要配置掩码的图属性** 的 **掩码方式** 是唯一的。对应表中则是针对department属性，需要脱敏的属性url只有一个规则mark(all)，需要脱敏的属性url1只有一个规则mark(custom,[0,1])。同样的，不满足时，在执行下一步时会报错。

1. 不同用户查询效果。

   a. 用户h查询看到的效果

```
0: jdbc:hive2://vqa36:10008/default> config query.lang cypher;
+--------------------------------------------------------+
|                        messages                        |
+--------------------------------------------------------+
| set cypher language and immediate mode successfully.  |
+--------------------------------------------------------+
1 row selected (0.029 seconds)
0: jdbc:hive2://vqa36:10008/default> use graph desens_wiki;
+---------+
| _a_c0  |
+---------+
+---------+
No rows selected (0.055 seconds)
0: jdbc:hive2://vqa36:10008/default> match (a) return a;
+------------------------------------------------------------------------------------------------------------------------------------------------+
|                                                                       a                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------------------+
| {"entityKey":[51,0,0],"properties":{"__tags":[],"url1":"*cc","__uid":"3","department":"abc","attr":3,"url":"**"},"labels":["PAGE"]}            |
| {"entityKey":[49,0,0],"properties":{"__tags":[],"url1":"*","__uid":"1","department":"abc","attr":1,"url":"*****"},"labels":["PAGE"]}           |
| {"entityKey":[52,0,0],"properties":{"__tags":[],"url1":"dddd","__uid":"4","department":"transwarp","attr":4,"url":"qq"},"labels":["PAGE"]}    |
| {"entityKey":[50,0,0],"properties":{"__tags":[],"url1":"*b","__uid":"2","department":"abc","attr":2,"url":"**"},"labels":["PAGE"]}             |
+------------------------------------------------------------------------------------------------------------------------------------------------+
4 rows selected (0.712 seconds)
```

   b. 用户h1查询看到的效果

```
0: jdbc:hive2://vqa36:10008/default> config query.lang cypher;
+--------------------------------------------------------+
|                        messages                        |
+--------------------------------------------------------+
| set cypher language and immediate mode successfully.  |
+--------------------------------------------------------+
1 row selected (0.294 seconds)
0: jdbc:hive2://vqa36:10008/default> use graph desens_wiki;
+---------+
| _a_c0  |
+---------+
+---------+
No rows selected (0.017 seconds)
0: jdbc:hive2://vqa36:10008/default>
0: jdbc:hive2://vqa36:10008/default> match (a) return a;
+------------------------------------------------------------------------------------------------------------------------------------------------+
|                                                                       a                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------------------+
| {"entityKey":[49,0,0],"properties":{"__tags":[],"url1":"a","__uid":"1","department":"abc","attr":1,"url":"baidu"},"labels":["PAGE"]}           |
| {"entityKey":[52,0,0],"properties":{"__tags":[],"url1":"*ddd","__uid":"4","department":"transwarp","attr":4,"url":"**"},"labels":["PAGE"]}    |
| {"entityKey":[50,0,0],"properties":{"__tags":[],"url1":"bb","__uid":"2","department":"abc","attr":2,"url":"jd"},"labels":["PAGE"]}             |
| {"entityKey":[51,0,0],"properties":{"__tags":[],"url1":"ccc","__uid":"3","department":"abc","attr":3,"url":"tb"},"labels":["PAGE"]}            |
+------------------------------------------------------------------------------------------------------------------------------------------------+
4 rows selected (0.478 seconds)
```

   c. 对于用户h2，因为没有在脱敏规则表中，可以查看到所有的值

```
0: jdbc:hive2://vqa36:10008/default> config query.lang cypher;
+-----------------------------------------------------+
|                      messages                       |
+-----------------------------------------------------+
| set cypher language and immediate mode successfully. |
+-----------------------------------------------------+
1 row selected (0.4 seconds)
0: jdbc:hive2://vqa36:10008/default> use graph desens_wiki;
+--------+
| _a_c0  |
+--------+
+--------+
No rows selected (0.082 seconds)
0: jdbc:hive2://vqa36:10008/default> match (a) return a;
+------------------------------------------------------------------------------------------------------------------------------------------+
|                                                                    a                                                                     |
+------------------------------------------------------------------------------------------------------------------------------------------+
| {"entityKey":[50,0,0],"properties":{"__tags":[],"url1":"bb","__uid":"2","department":"abc","attr":2,"url":"jd"},"labels":["PAGE"]}      |
| {"entityKey":[49,0,0],"properties":{"__tags":[],"url1":"a","__uid":"1","department":"abc","attr":1,"url":"baidu"},"labels":["PAGE"]}    |
| {"entityKey":[52,0,0],"properties":{"__tags":[],"url1":"dddd","__uid":"4","department":"transwarp","attr":4,"url":"qq"},"labels":["PAGE"]} |
| {"entityKey":[51,0,0],"properties":{"__tags":[],"url1":"ccc","__uid":"3","department":"abc","attr":3,"url":"tb"},"labels":["PAGE"]}     |
+------------------------------------------------------------------------------------------------------------------------------------------+
4 rows selected (0.398 seconds)
0: jdbc:hive2://vqa36:10008/default>
```

2. 如果此时需要将用户h2加入到脱敏体系中，可以将用户根据department属性进行分组。并且切换到TEoC语言，使用load语句重新加载规则。

```
0: jdbc:hive2://vqa36:10008/default> select * from desens.desens_table;
+---------------+---------------------+-----------+--------------+---------------------+
| group_by_attr | group_by_attr_value | username  | desens_prop  |     desens_rule     |
+---------------+---------------------+-----------+--------------+---------------------+
| department    | transwarp           | h         | url          | mark(all)           |
| department    | abc                 | h1        | url1         | mark(custom,[0,1])  |
| department    | transwarp           | h2        | url          | mark(all)           |
+---------------+---------------------+-----------+--------------+---------------------+
3 rows selected (0.626 seconds)
```

```
0: jdbc:hive2://vqa36:10008/default> LOAD DESENSITIZATION RULES FROM TABLE desens.desens_table INTO GRAPH desens_wiki;
+--------------------------------------+
|                _a_c0                 |
+--------------------------------------+
| Load 3 rows in desensitization rules |
+--------------------------------------+
1 row selected (3.293 seconds)
```

3. 验证修改后的效果：对于用户h2，因为没有在脱敏规则表中，可以查看到所有的值。

```
0: jdbc:hive2://vqa36:10008/default> match (a) return a;
+------------------------------------------------------------------------------------------------------------------------------------------+
|                                                                    a                                                                     |
+------------------------------------------------------------------------------------------------------------------------------------------+
| {"entityKey":[51,0,0],"properties":{"__tags":[],"url1":"*cc","__uid":"3","department":"abc","attr":3,"url":"**"},"labels":["PAGE"]}     |
| {"entityKey":[50,0,0],"properties":{"__tags":[],"url1":"*b","__uid":"2","department":"abc","attr":2,"url":"**"},"labels":["PAGE"]}      |
| {"entityKey":[49,0,0],"properties":{"__tags":[],"url1":"*","__uid":"1","department":"abc","attr":1,"url":"*****"},"labels":["PAGE"]}    |
| {"entityKey":[52,0,0],"properties":{"__tags":[],"url1":"dddd","__uid":"4","department":"transwarp","attr":4,"url":"qq"},"labels":["PAGE"]} |
+------------------------------------------------------------------------------------------------------------------------------------------+
4 rows selected (0.285 seconds)
```

4. 查看当前加载了脱敏规则表的图。

   语法规则如下：

```
SHOW DESENSITIZATION GRAPHS;
```

   该语句调用示例如图：

```
0: jdbc:hive2://vqa36:10101/default> SHOW DESENSITIZATION GRAPHS;
+--------------+
|    _a_c0     |
+--------------+
| desens_wiki  |
+--------------+
1 row selected (0.012 seconds)
```

5. 为指定图删除加载脱敏规则表。

```
DROP DESENSITIZATION GRAPH [GRAPH_NAME];
```

该语句调用示例如图：

```
0: jdbc:hive2://vqa36:10101/default> DROP DESENSITIZATION GRAPH desens_wiki;
+--------+
| _a_c0  |
+--------+
+--------+
No rows selected (0.013 seconds)
0: jdbc:hive2://vqa36:10101/default> SHOW DESENSITIZATION GRAPHS;
+--------+
| _a_c0  |
+--------+
+--------+
No rows selected (0.025 seconds)
```

在这里仅是将图desens_wiki从脱敏图中删除，它会变成普通图，仍然可以正常查询，只不过此时不会有任何脱敏的效果。

10. KG Explorer使用文档（新）

10.1. KG Explorer简介

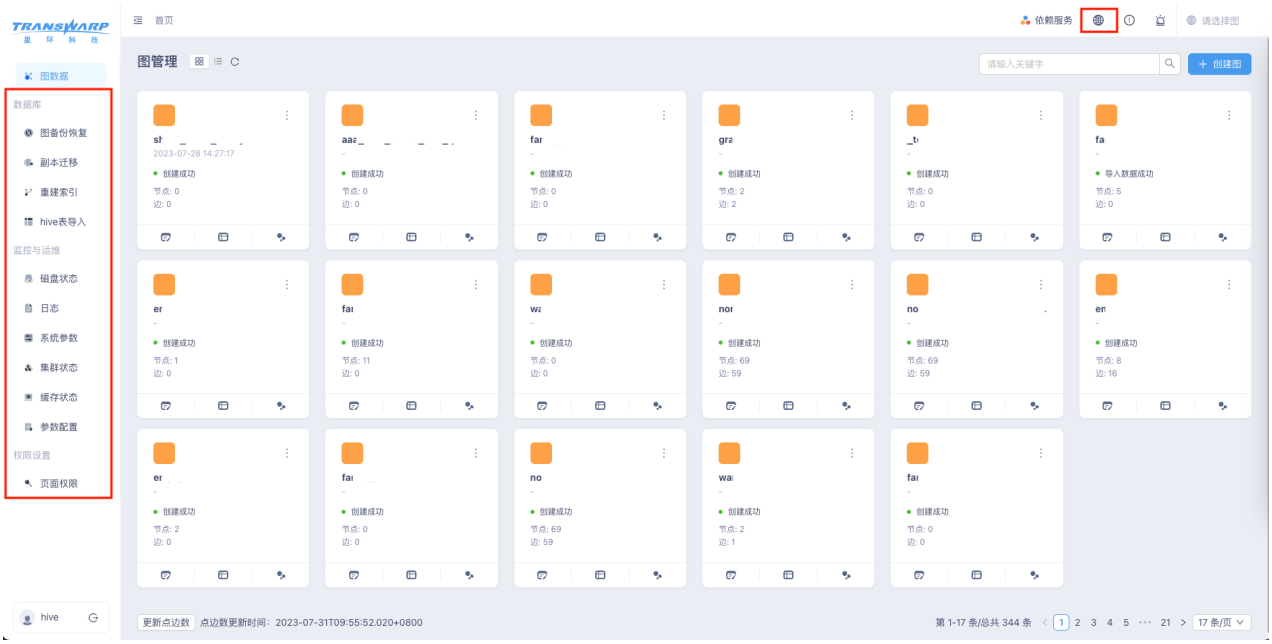
KG Explorer是StellarDB的多功能数据可视化化展示、交互界面，帮助用户查询、探索、分析图数据。相较于前置版本，StellarDB 5.0.0中，对KG Explorer引入了深度的优化和功能更新。这些优化和更新包括但是不限于新的UI设计、动态图时间轴可视化、更加丰富的样式和图例配置、易于运维的副本管理功能等。这些功能将在手册后文中进行更加详细的介绍。

10.2. KG Explorer功能介绍

KG Explorer提供的主要功能包括：

- 2D图数据可视化交互页面
- 3D图数据可视化交互页面
- 可视化创建图谱
- 数据导入工具
- 动态图时间轴可视化
- 页面权限管控工具
- 图数据库的监控与运维管理工具

上述功能对应菜单栏入口如下图所示：



10.3. KG Explorer常用参数说明

用户可以在Transwarp Manager页面修改或添加KG Explorer的配置，配置修改后需要在Transwarp Manager页面 **配置服务** 并 **重启** KG Explorer。 **配置服务** 后最终生成的配置信息可以在KG Explorer服务配置文件目录下的 `application.properties` 文件中查看。

除了在Transwarp Manager页面设置的参数以外，`application.properties` 文件中还存在一些其他配置，例如依赖服务的相关信息， 这些参数是KG Explorer自动获取相关信息生成的，一般情况下无需手动修改。若在某些特殊情况下这些参数需要改动，用户也可以在Transwarp Manager页面添加相同的配置来对其进行覆盖。

由于参数过多，下面列出了常用的重要配置参数，若配置KG Explorer时仍有问题，请联系StellarDB KG Explorer技术人员。

参数名	默认值	是否必须	参数说明	备注
server.port	38282	是	服务端口号	
server.opts	无	否	JVM启动参数	
server.ssl.enabled	false	否	是否开启https	
stellar.dependency.inceptor.hostPort	依赖的Quark服务的地址	是	依赖的Quark服务的地址：{Quark Server的host}:{Quark配置中的的hive.server2.thrift.port}	默认会自动获取，不需要手动添加该参数。当KG Explorer需要连接Quark-QuarkGateway时修改该参数为Quark-QuarkGateway服务的地址

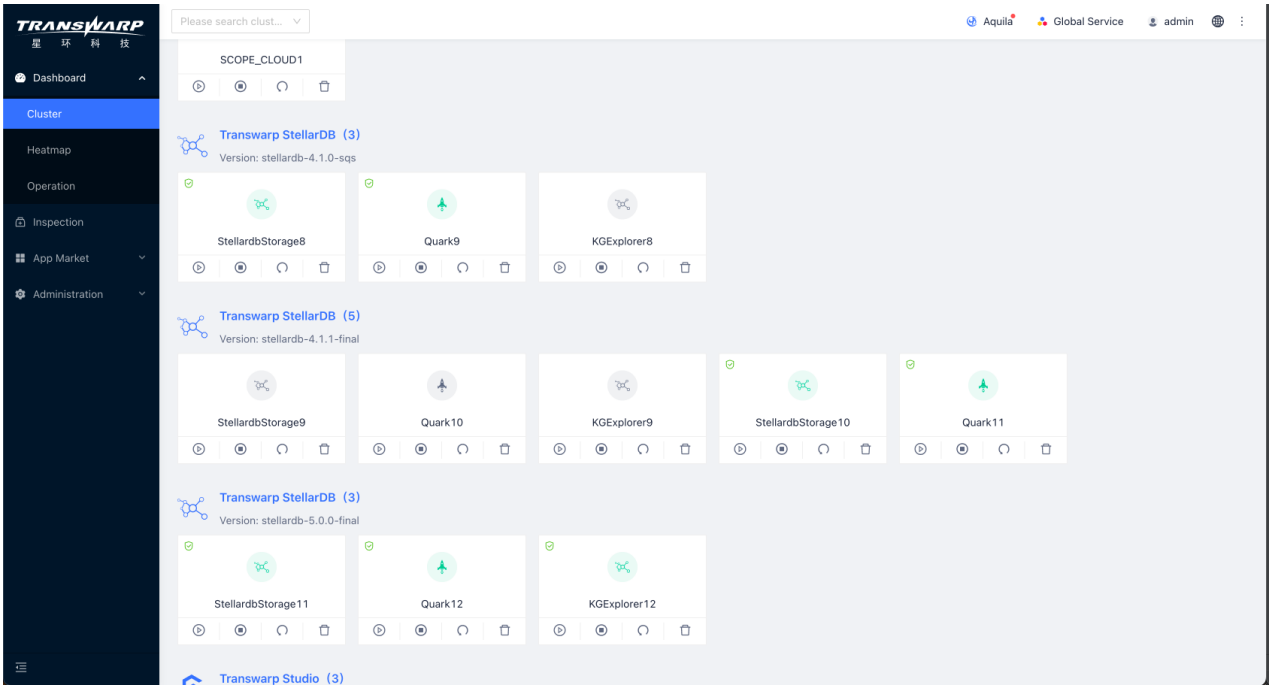
参数名	默认值	是否必须	参数说明	备注
stellar.dependency.inceptor.defaultUserName	hive	否	默认访问依赖Quark的用户名	若依赖Quark的认证方式为 LDAP，即 hive.server2.authentication=LDAP，则必须配置
stellar.dependency.inceptor.defaultUserPassword		否	默认访问依赖Quark的用户密码	若依赖Quark的认证方式为 LDAP，即 hive.server2.authentication=LDAP，则必须配置
security.oauth2.enable	false	否	是否开启跳转Guardian Federation跳转登录	开启KG Explorer跳转登录详细步骤见下方
stellar.pagePermission.enable	false	否	是否开启KG Explorer页面权限功能	需要开启KG Explorer的Guardian插件
stellar.restApi.auth	true	否	对外查询接口是否需要进行用户校验	需要开启KG Explorer的Guardian插件
stellar.meta.stellarWebGraphName	（根据服务id自动生成）	是	KG Explorer在StellarDB上存储信息的图名	会自动生成，一般无需修改；无特殊情况不要删除该图
stellar.meta.stellarWebDataBase	（根据服务id自动生成）	是	KG Explorer在Quark上存储信息的数据库名	会自动生成，一般无需修改；无特殊情况不要删除该数据库
stellar.meta.stellarHdfsBase	（根据服务id自动生成）	是	KG Explorer在HDFS上存储文件的路径	会自动生成，一般无需修改；无特殊情况不要删除目录以及里面的文件
stellar.meta.dataDirs	（自动生成）	是	KG Explorer存储文件的数据目录	会自动生成，一般无需修改从；一般会自动生成多个数据目录，当前只用到了第一个目录；无特殊情况不要删除目录以及里面的文件
stellar.jdbc.sessionPool.maxCount	100	是	session数目上限	
stellar.jdbc.sessionPool.maxCountForUser	20	是	单个用户的session数目上限	
stellar.jdbc.sessionPool.createNewSessionEachTime	false	是	是否每次操作图数据库都开启一个新的StellarJDBC session	
stellar.jdbc.sessionInvalidTime	30min	否	StellarJDBC session的空闲失效时间	可以设置的比依赖的Quark服务上实际的session的空闲失效时间短一点，让KG Explorer提前进行关闭

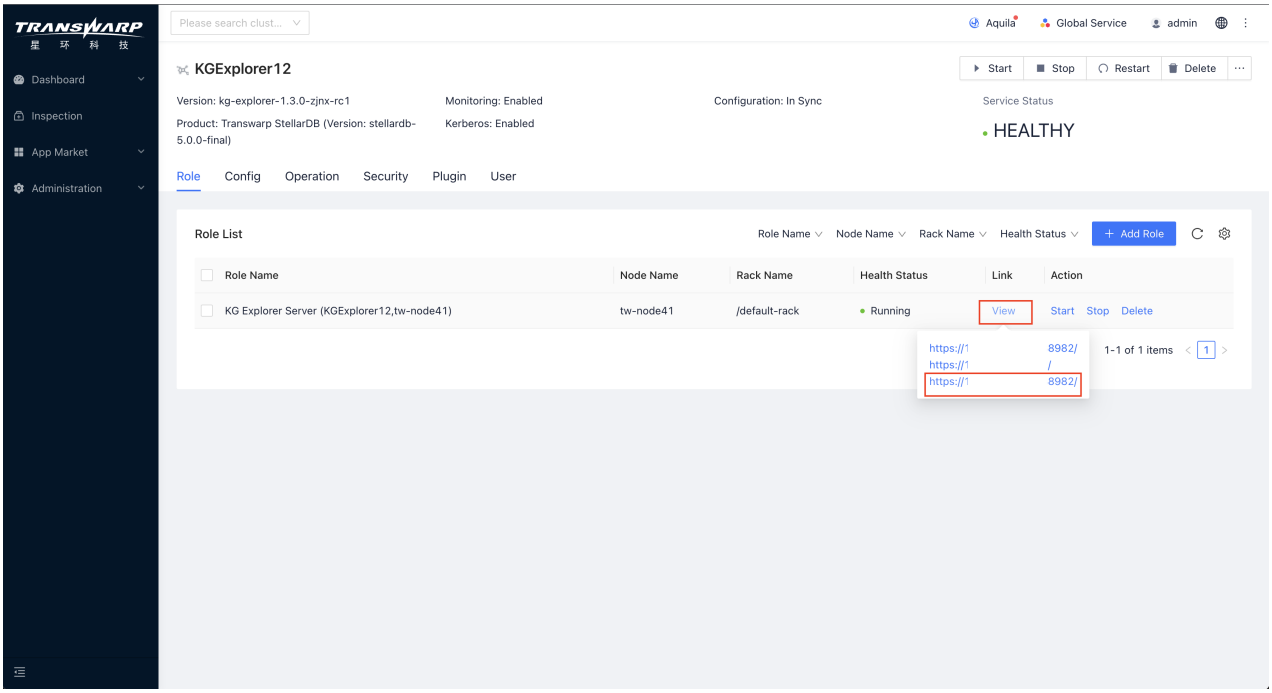
参数名	默认值	是否必须	参数说明	备注
server.servlet.session.cookie.max-age	1800（单位为秒）	否	浏览器cookie失效时间	若开启用户登录之后，cookie失效会跳转到登录页面重新登录，用户可以视情况调整失效时间
server.servlet.session.timeout	1800（单位为秒）	否	浏览器session失效时间	若开启用户登录之后，session失效也会跳转到登录页面重新登录，用户可以视情况调整失效时间

KG Explorer正常启动需要获取依赖Quark以及StellarDB等依赖服务的相关参数，若修改了依赖的配置，请 **配置服务** 并且 **重启** KG Explorer。

10.4. 访问KG Explorer

KG Explorer的登录方式为：在Transwarp Manager界面选择KG Explorer服务→角色→查看选择地址，参考下图：





10.5. KG Explorer用户认证与安全

1. 启用 `https` 协议

   KG Explorer默认不使用 `https`，可以通过配置参数 `server.ssl.enabled` 进行开启。当该参数值为 `true` 时表示开启，`false` 时表示关闭。

2. 开启用户认证

   KG Explorer默认不开启用户认证，可以通过如下方式开启：

   a. 开启KG Explorer以及依赖Quark服务的Guardian插件；

   b. 开启KG Explorer以及依赖Quark服务的安全（此步骤会为服务注册 `Oauth2 Client`；

   c. 确保依赖Quark服务支持 `oauth2`，即确保Quark服务参数 `hive.server2.authentication.oauth2.enabled=true`；

   d. KG Explorer设置参数 `enable.oauth2=true` 开启用户登录功能，将值设为 `false` 可关闭该功能。

3. 注意事项

   a. 若依赖Quark的认证方式为 `LDAP`，即 `hive.server2.authentication=LDAP`，则需要配置KG Explorer默认访问Quark的用户名和密码。可通过查看 `stellar.dependency.inceptor.defaultUserName` 查看对应默认访问Quark的用户名；通过 `stellar.dependency.inceptor.defaultUserPassword` 查看对应默认访问Quark的用户密码。

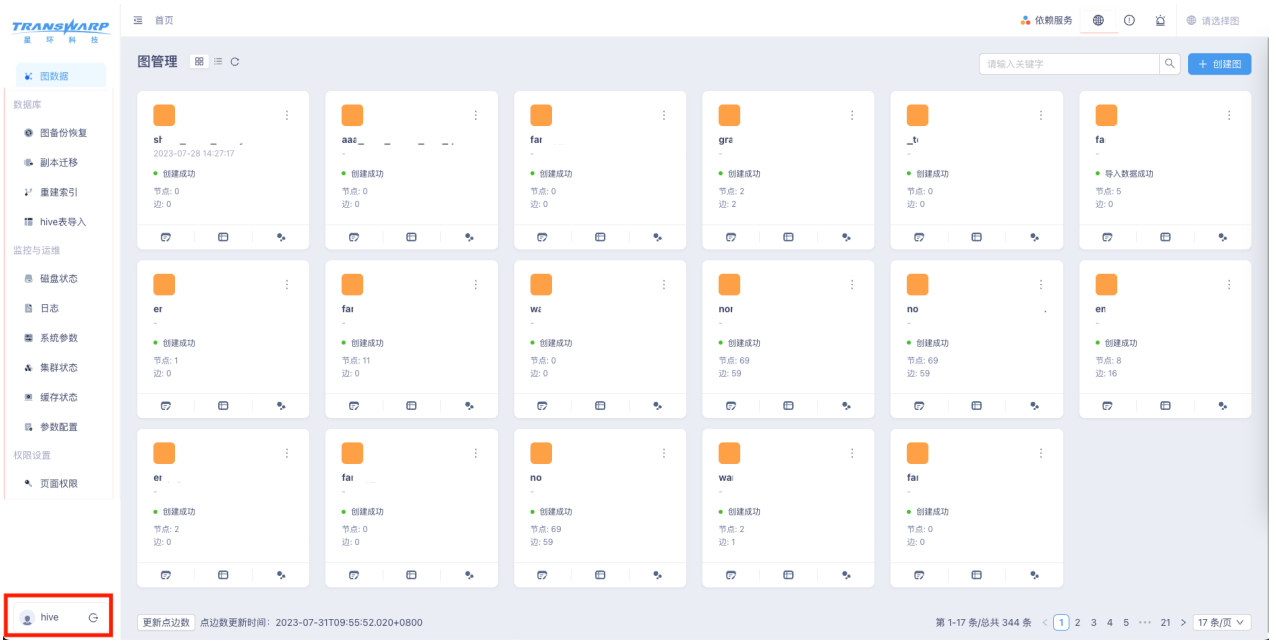
   b. 修改服务配置参数后需要配置服务然后重启才能生效。

   c. 若依赖Quark相关配置进行了修改，KG Explorer需要重新配置服务并重启以获取到依赖Quark更新后的信息。

   d. TDC环境上暂不支持开启KG Explorer用户登录认证。

若KG Explorer开启了用户认证，会自动跳转Guardian Federation登录页面。用户按如图方式登录：

若KG Explorer开启了用户认证，登陆后想切换登录用户，可以鼠标移动到左下角的用户名，用户名右边有“登出”按钮，页面会自动跳转到登录页面，即可切换用户进行登录



10.6. KG Explorer主要功能介绍

10.6.1. 可视化图谱创建

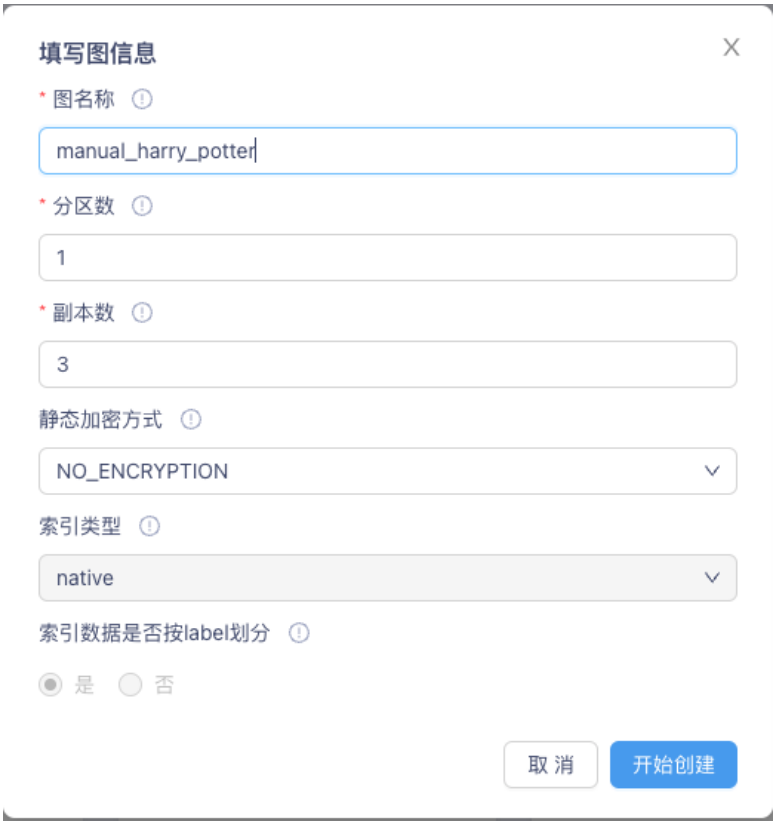
1. 功能介绍

   在页面通过鼠标点击和补充相应信息的形式在线创建图谱（schema）。

2. 操作说明

   在主页面右上角点击创建图按钮开始图谱schema的构建。

按照引导填写图基本信息后点击确定进入构建页面。

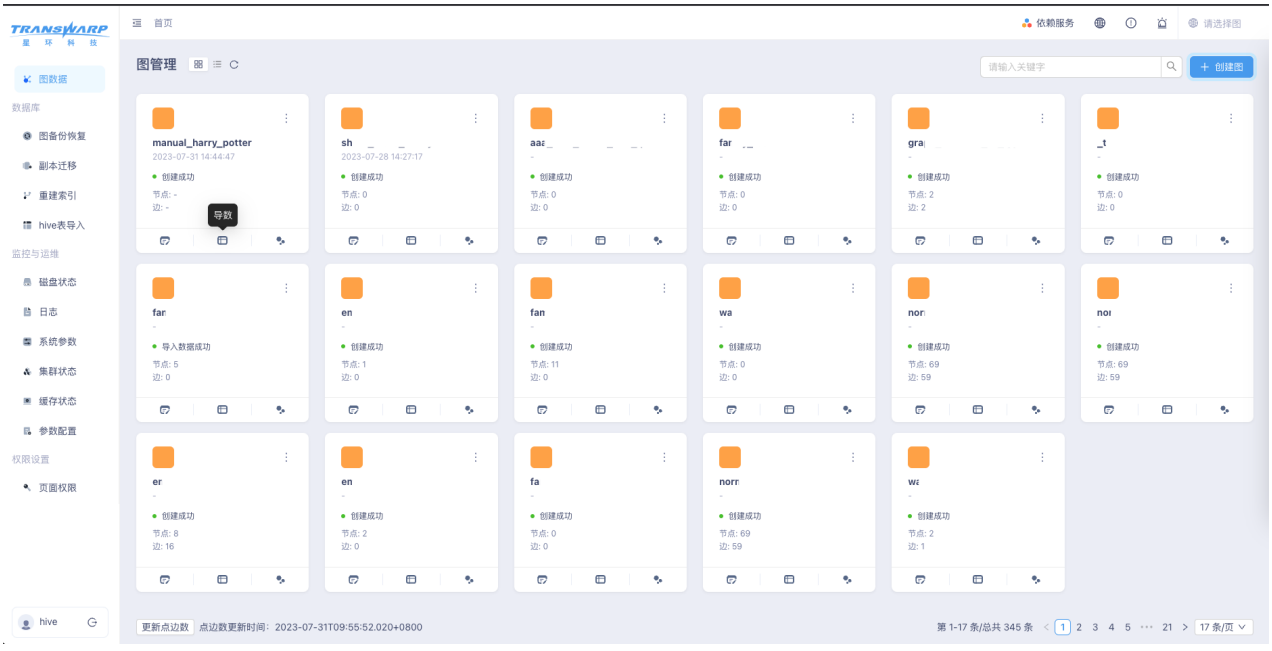


根据引导创建完成图schema后点击发布，即可在图数据库创建对应图谱。创建图相关知识详见《创建图》 章节。

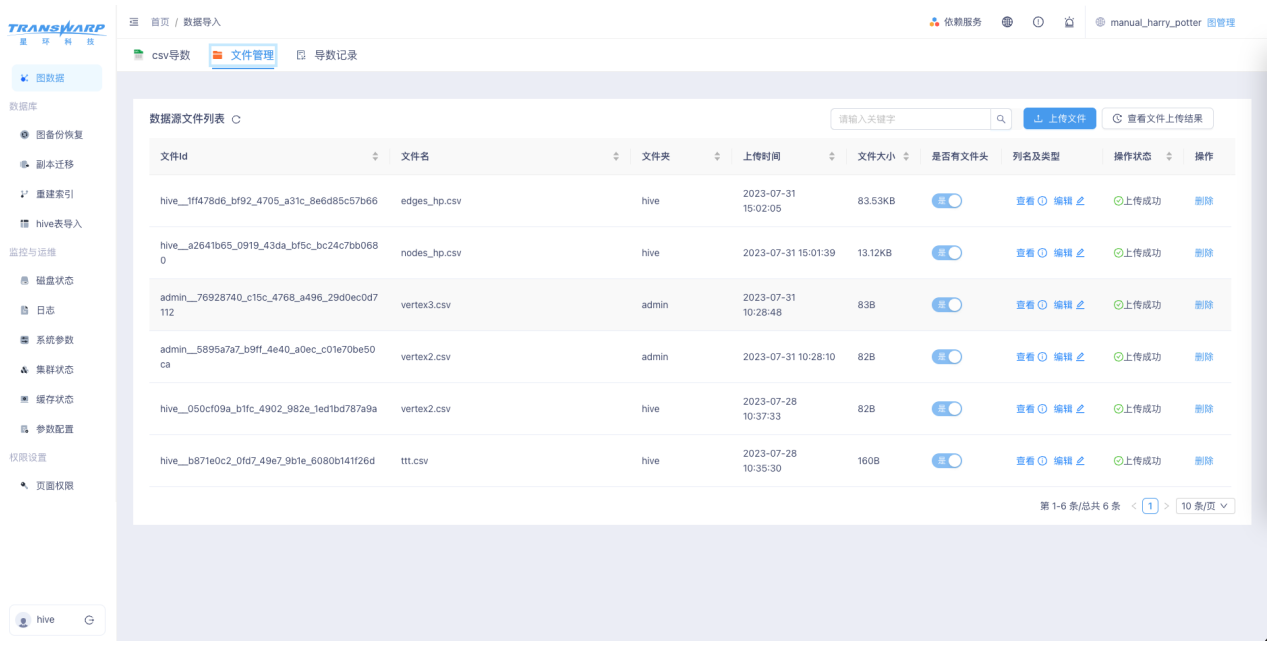
10.6.2. CSV数据导入

在StellarDB 5.0.0版本，我们在KG Explorer可视化组件中重新规划了图数据的空间管理，CSV数据的导入在当前版本中，定义为针对某个特定的图进行数据导入。

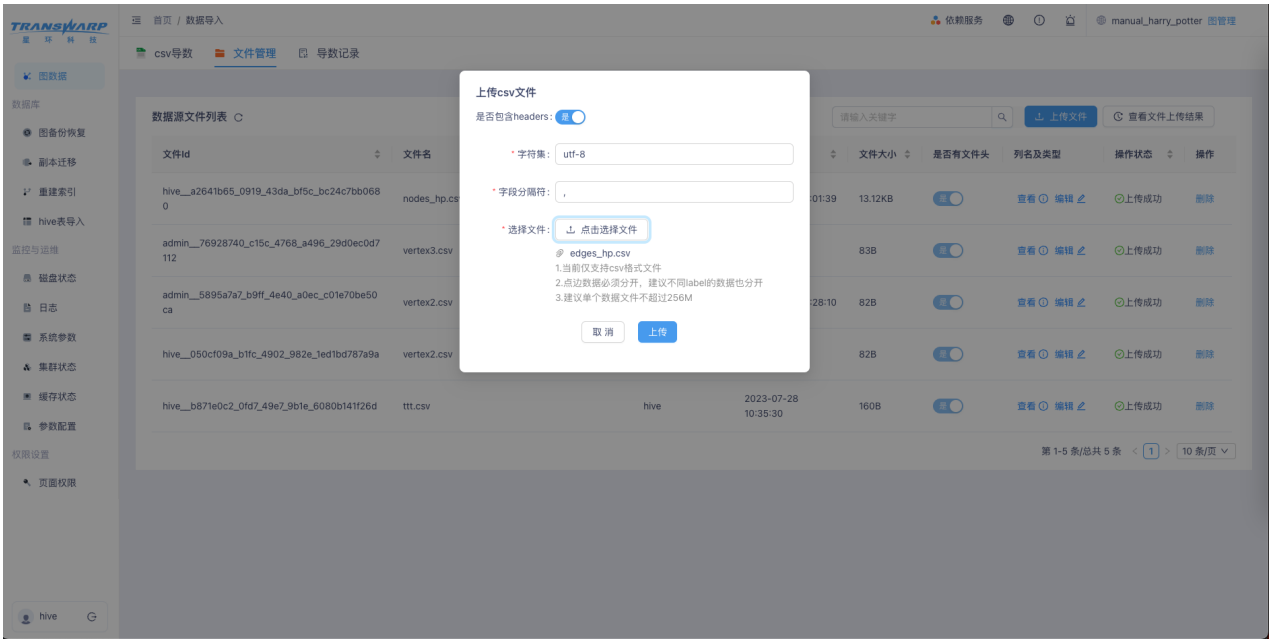
在KG Explorer图管理主页面，我们可以通过搜索、浏览图数据库中的图寻找需要进行数据导入的图。这里以”manual_harry_potter“为例，在图名下方，我们可以通过点击“导数”按钮进入CSV数据导入配置页面。



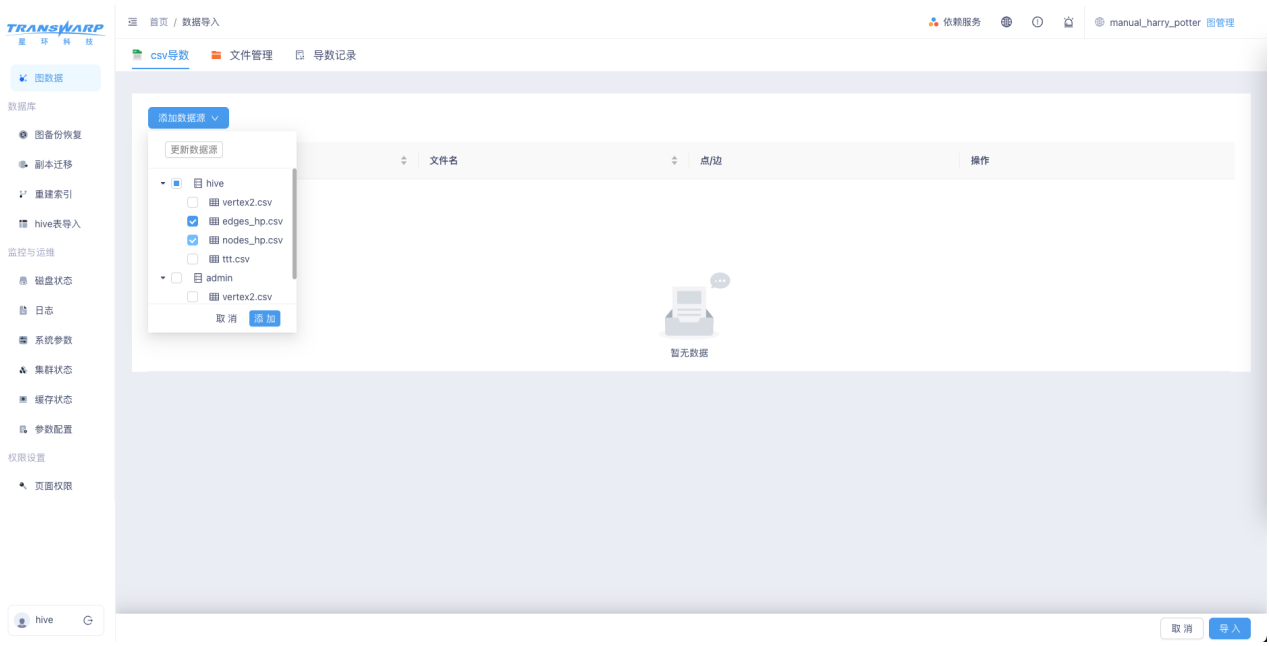
在 **数据导入** 操作页面，点击页面上方 **数据源管理** 标签可进入数据源管理页面。可在该页面管理数据源文件，包括上传文件、删除文件、修改文件列名及类型等。

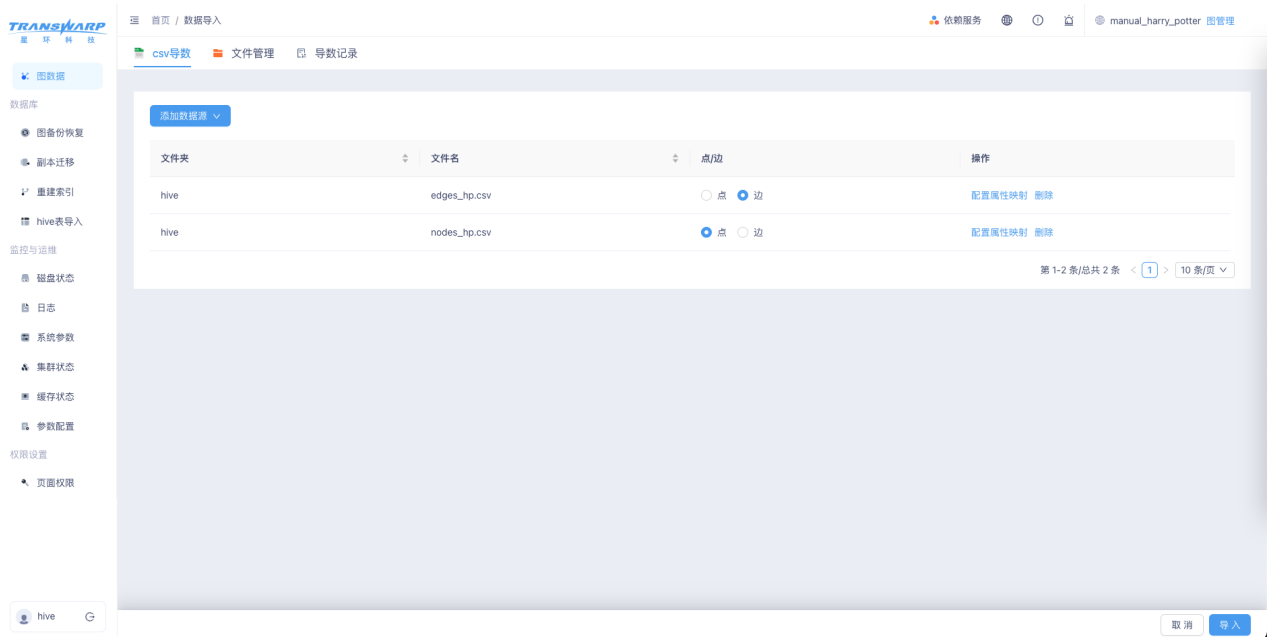


上传文件。

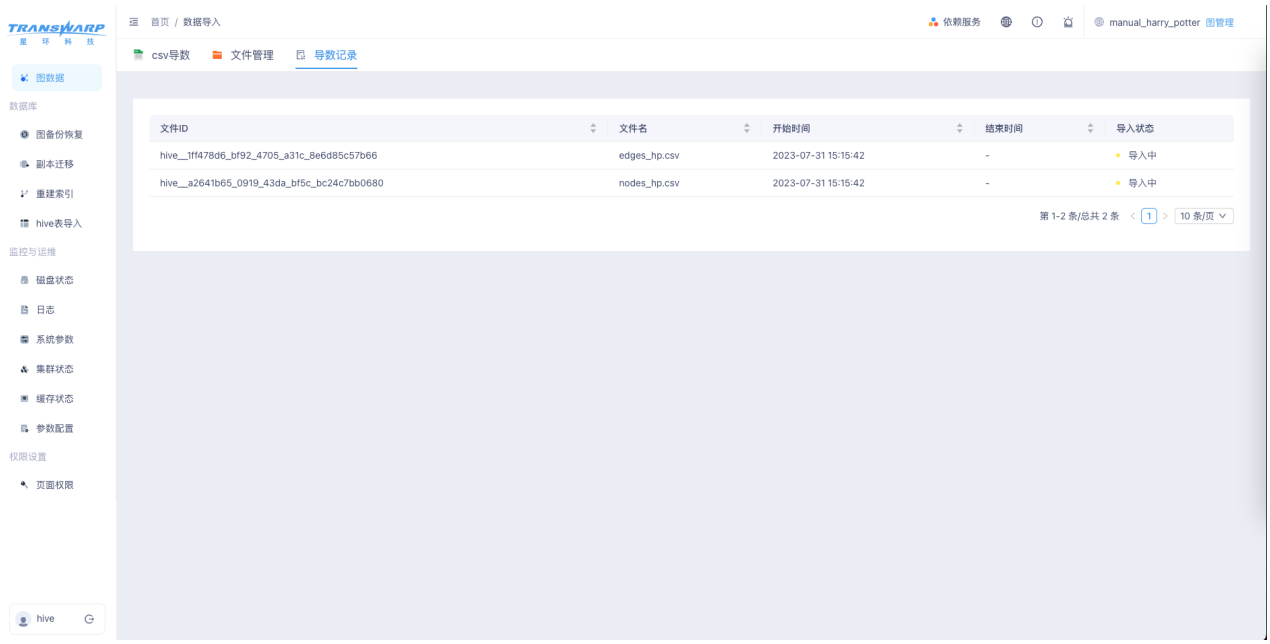


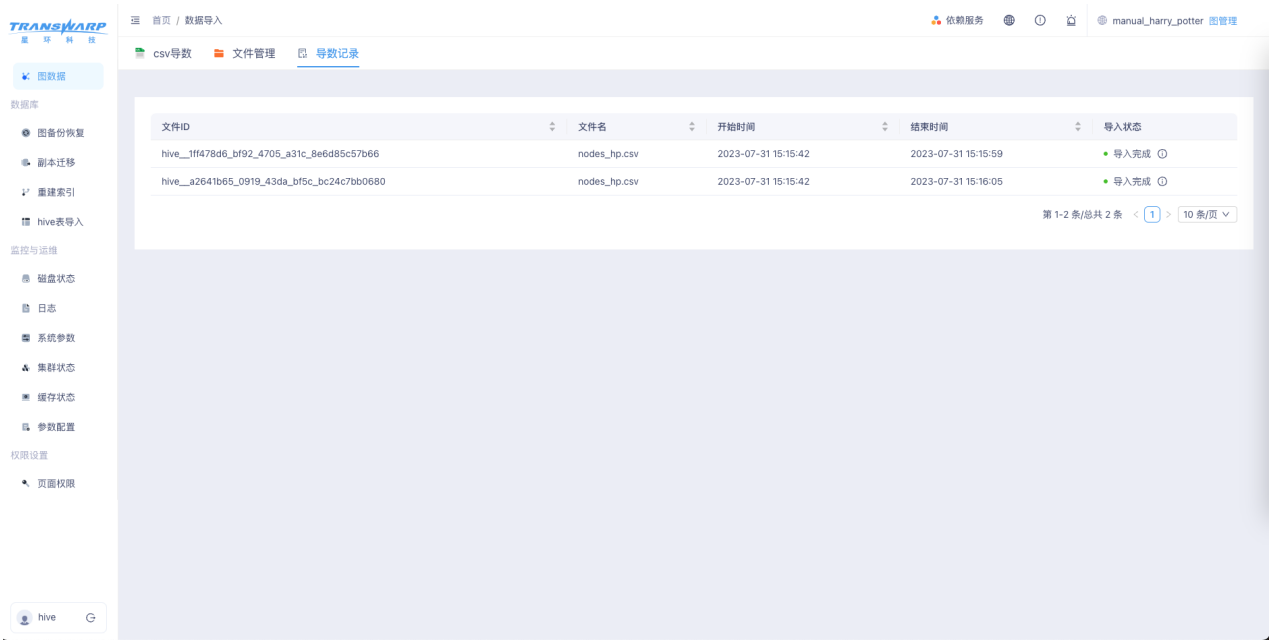
在文件导入操作页面添加数据源文件，填写对应信息进行数据映射，完成后点击“确定”即可导入数据。 此时页面状态会变成数据导入中，等待导入操作完成，可显示数据导入结果状态。


首页 / 数据导入
依赖服务
manual_harry_potter 图管理
csv导数
文件管理
导数记录
添加数据源
更新数据源
hive
vertex2.csv
edges_hp.csv
nodes_hp.csv
ttt.csv
admin
vertex2.csv
取 消
添 加
文件名
点/边
操作
暂无数据
图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
hive
取 消
导 入


首页 / 数据导入
依赖服务
manual_harry_potter 图管理
csv导数
文件管理
导数记录
添加数据源
文件夹
文件名
点/边
操作
hive
edges_hp.csv
点
边
配置属性映射 删除
hive
nodes_hp.csv
点
边
配置属性映射 删除
第 1-2 条/总共 2 条
10 条/页
图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
hive
取 消
导 入


点数据: /hive/nodes_hp.csv
* 点uid对应列:
id
* 点label映射方式:
从图schema中选择固定label值
* label映射:
v
* 属性映射:
id
图属性所属label
图属性名称
是否需要导入数据
全选是
映射类型
映射列名/常量
0
v
name
是
列名
id
第 1-1 条/总共 1 条
10 条/页
取 消
确 定

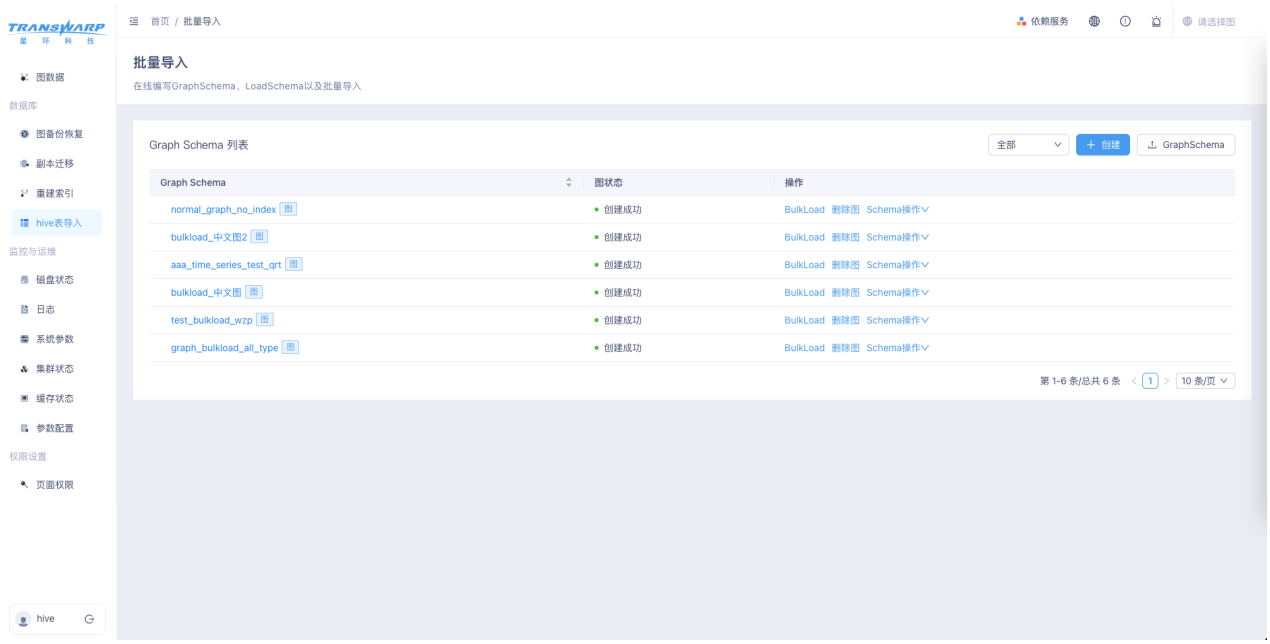




10.6.3. Hive表数据导入

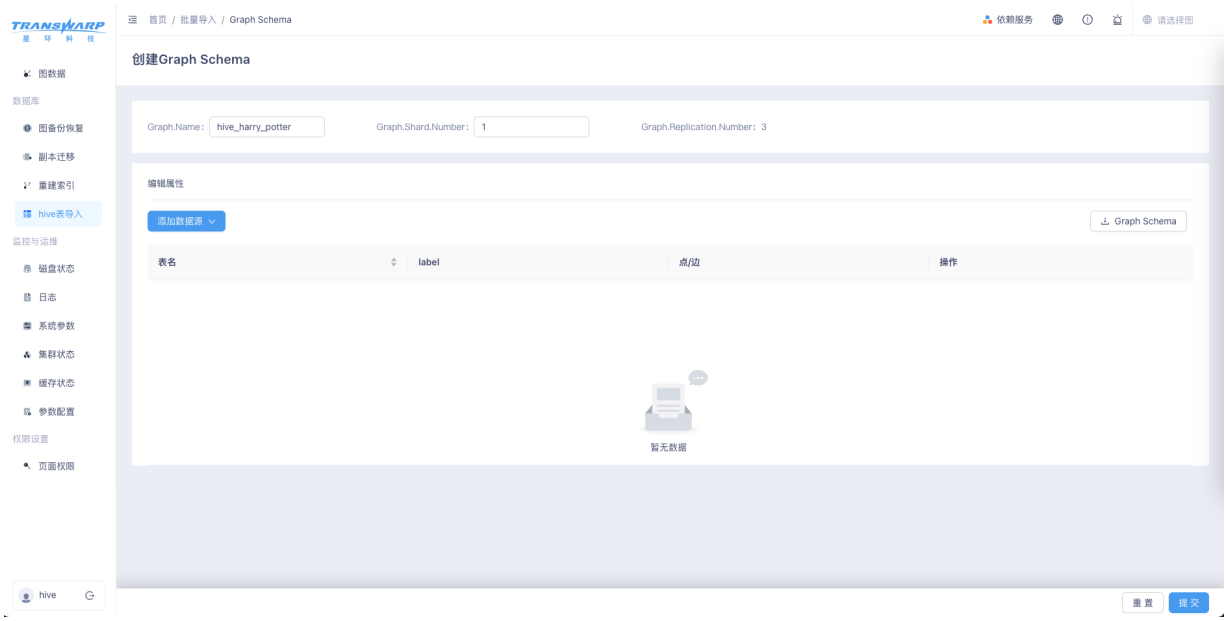
StellarDB 5.0.0 版本还继续单独提供从Hive表导出数据到StellarDB图数据库中。

在KG Explorer图管理页面左侧菜单栏点击 **Hive数据导入** 进入批量导入页面，页面中会显示创建过的图schema列表。在图schema列表，提供了查看、删除、上传、下载、导入数据等功能（导入数据详细内容见下方）。

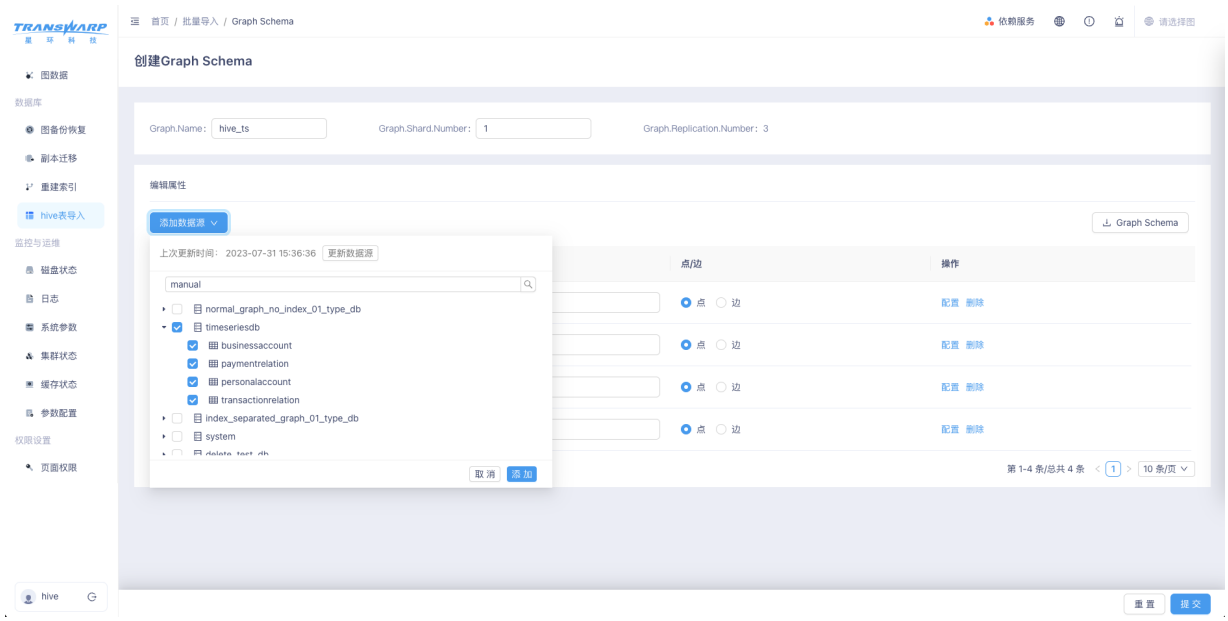


点击 **创建** 按钮即可开始创建图。创建详细步骤如下：

1. 填写基本信息：图名、图数据分片数量。



2. 数据源选择：点击 **添加数据源**，选择需要添加的点表和边表，通过勾选的方式进行添加或者取消添加。



3. Graph Schema生成：可以选择上传Graph Schema或生成Graph Schema的方式创建图，这里通过生成Graph Schema的方式，对每个加入数据源的表做如下操作：

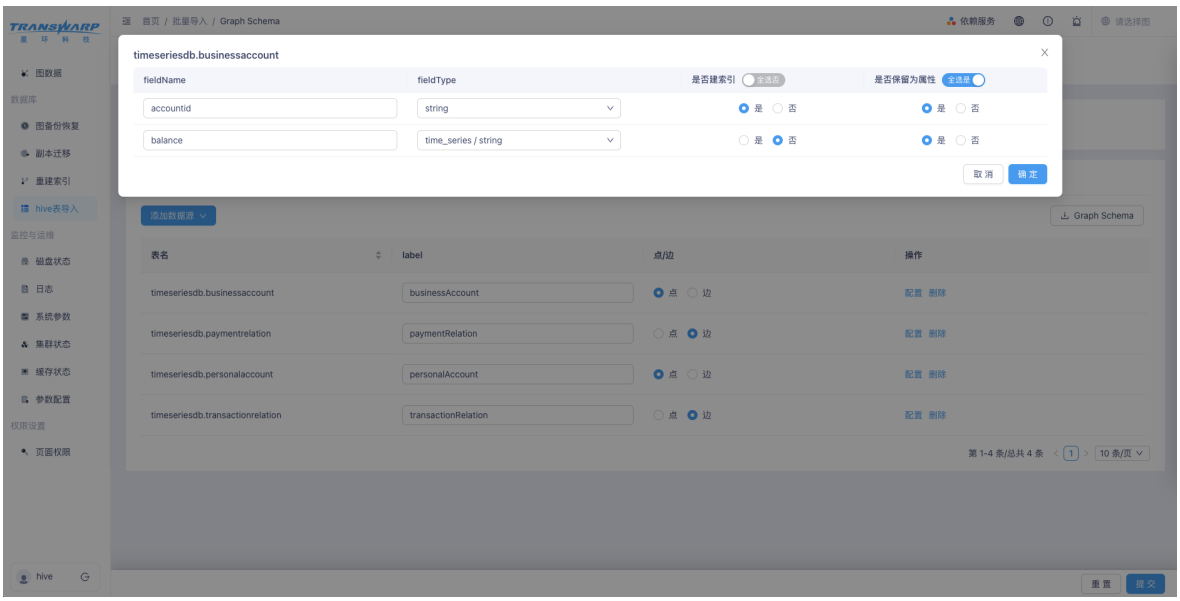
   a. 填写label类型，例如:某个点数据源的label为"person"，边的label为"friend"；

   b. 点选数据类型（点或边）；

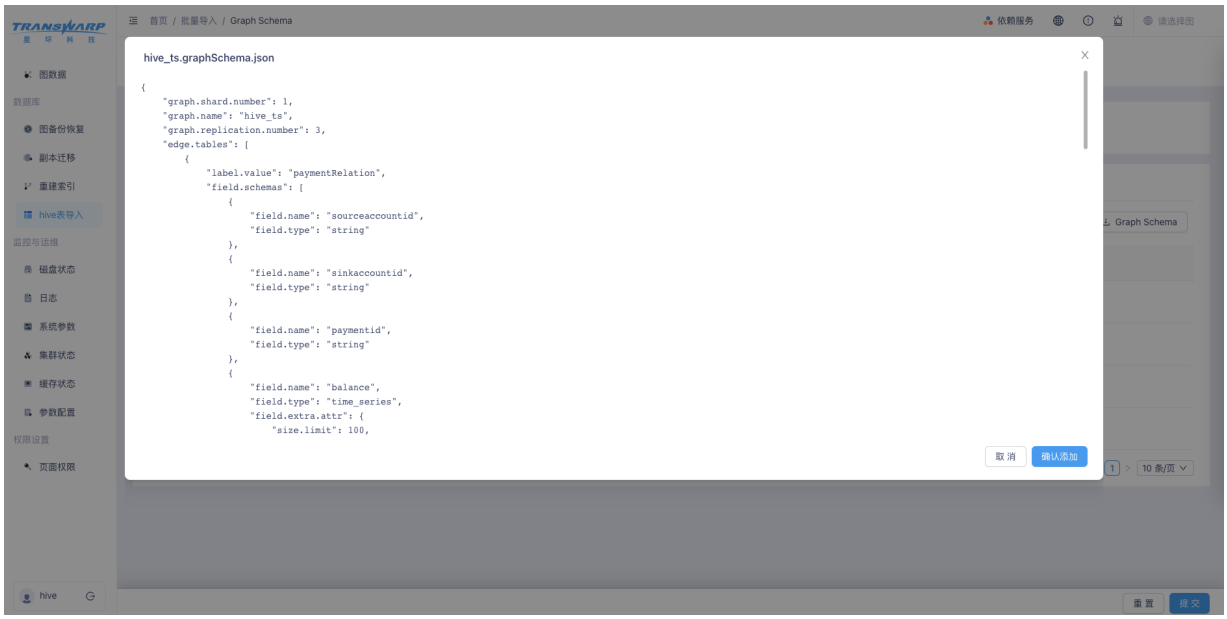
   c. 点击 **属性配置** 进行属性编辑，为每个属性选择是否保留为属性以及是否建索引；

   d. 配置完成并确认后点击提交。

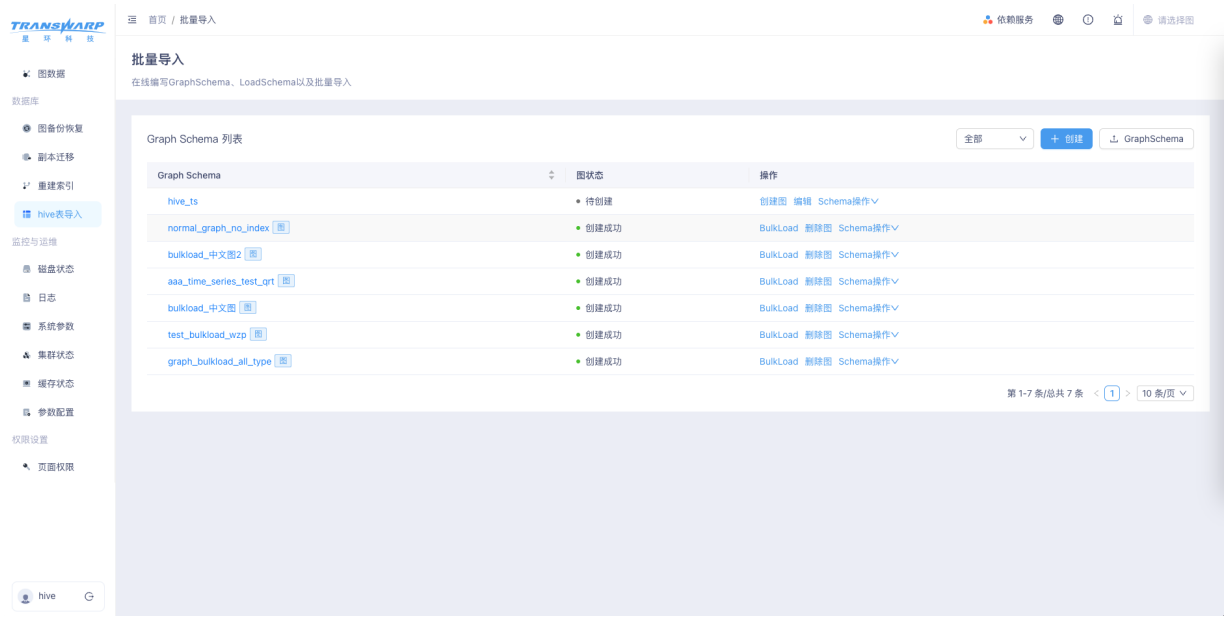
   上传的Graph Schema文件名需要满足如下格式： {图名}.graph_schema.json

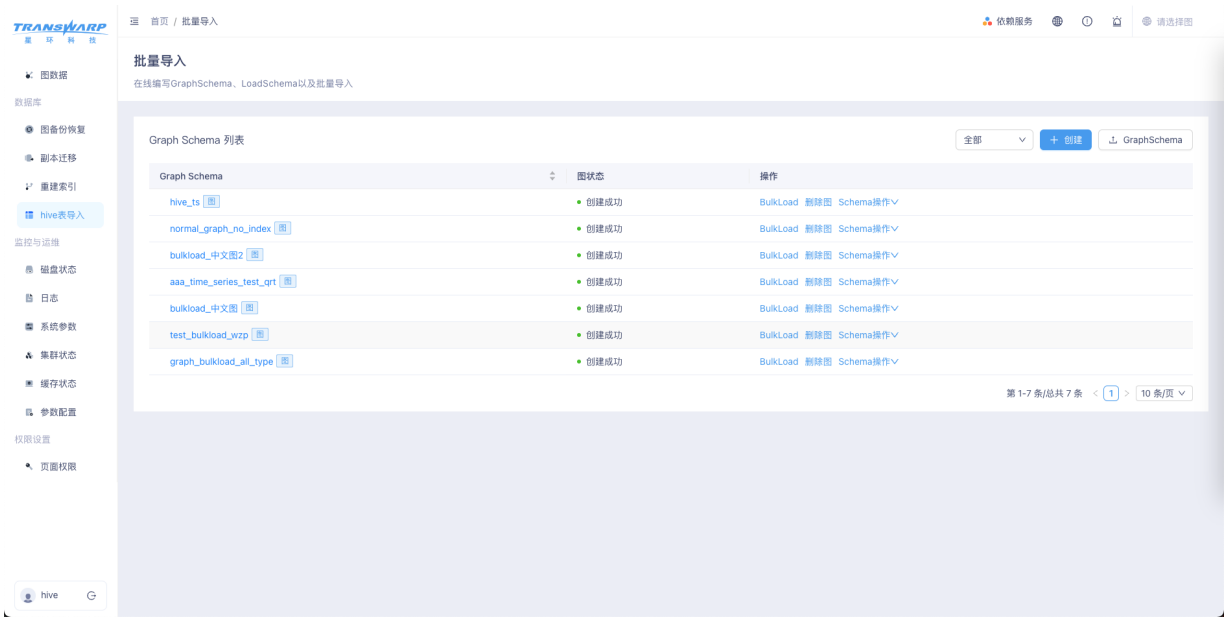


4. 点击 **提交** 按钮后，会弹出图schema的json字符串，可以在此步骤进行图schema的核对。

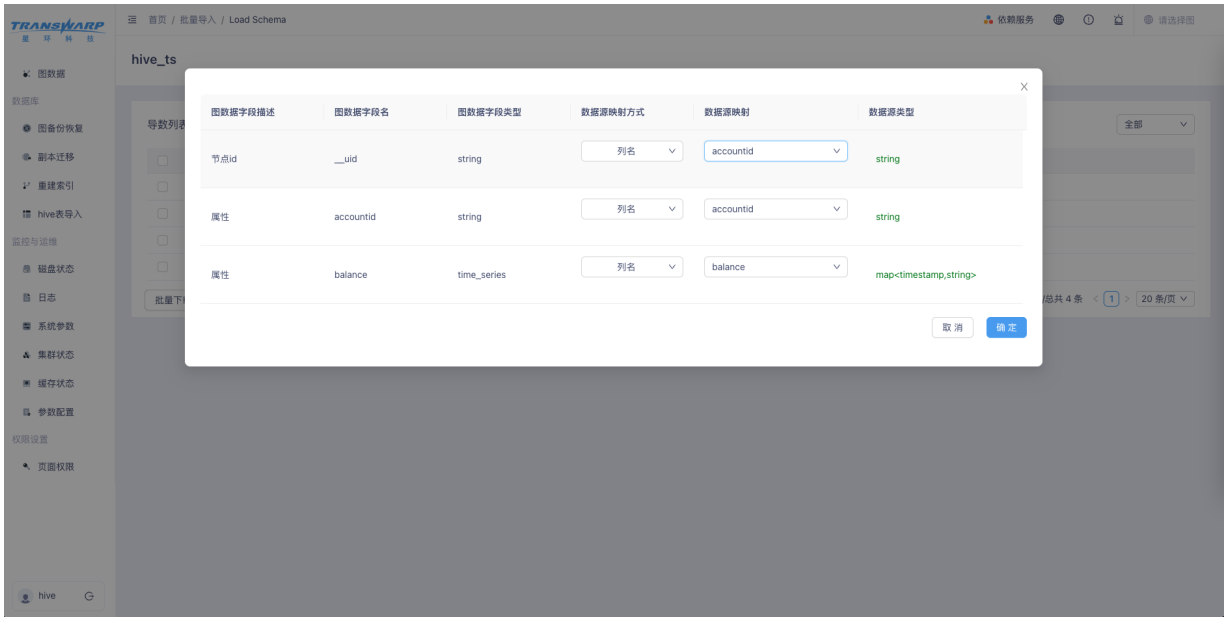


5. 创建图并导入数据。点击 **创建图** 按钮创建图谱。创建图谱成功后，会提示创建成功消息，**创建图** 按钮变成 **bulkload** 按钮。

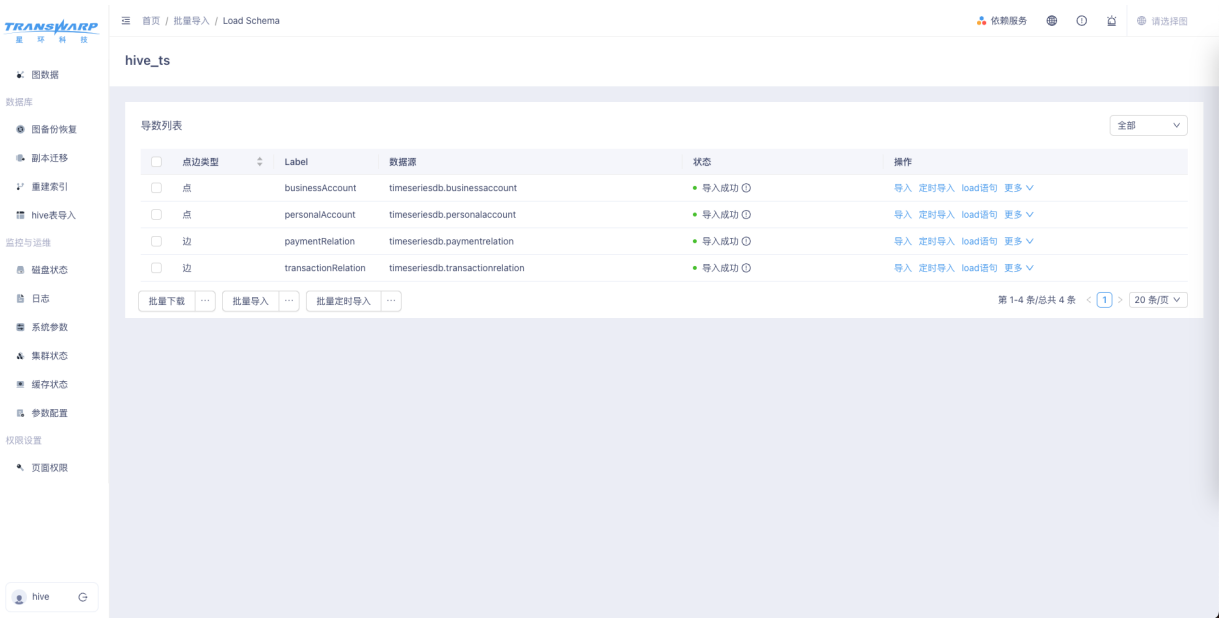




创建完成后点击 **bulkload** 按钮进入 **load schema** 创建页面。 可以点击 **数据映射配置** 按钮，配置映射信息，生成 **load schema**。



配置完成后可以选择 **导入** 或 **定时导入**，**定时导入** 可以设置导入开始时间。 开始批量导入任务之后，**导入** 按钮会变成 **导入中** 状态，等待导数操作完成，显示 **导入成功** 并且可查看导数结果。



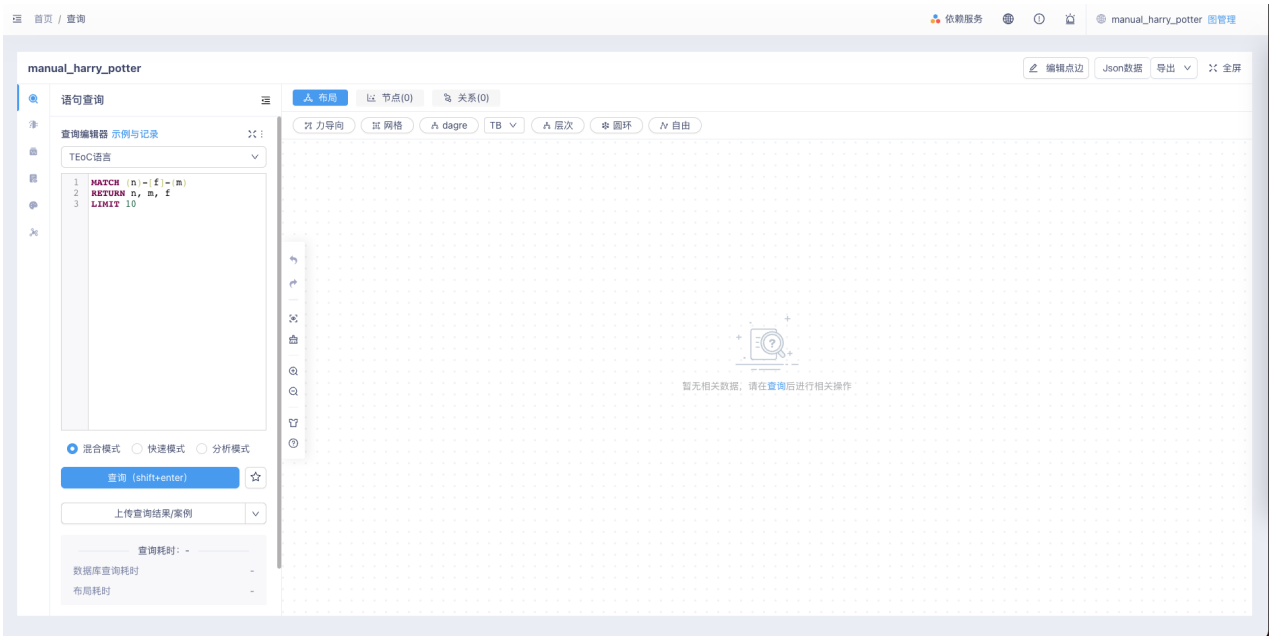
上传的load schema文件名需要满足如下格式： {图名}.{数据源database名}.{数据源table名}.{点边类型，vertex或edge}.{label}.load_schema.json

10.6.4. 2D查询分析展示

KG Explorer提供TEoC查询接口，2D展示查询结果，并提供了针对查询结果的各项页面操作。

10.6.4.1. 查询

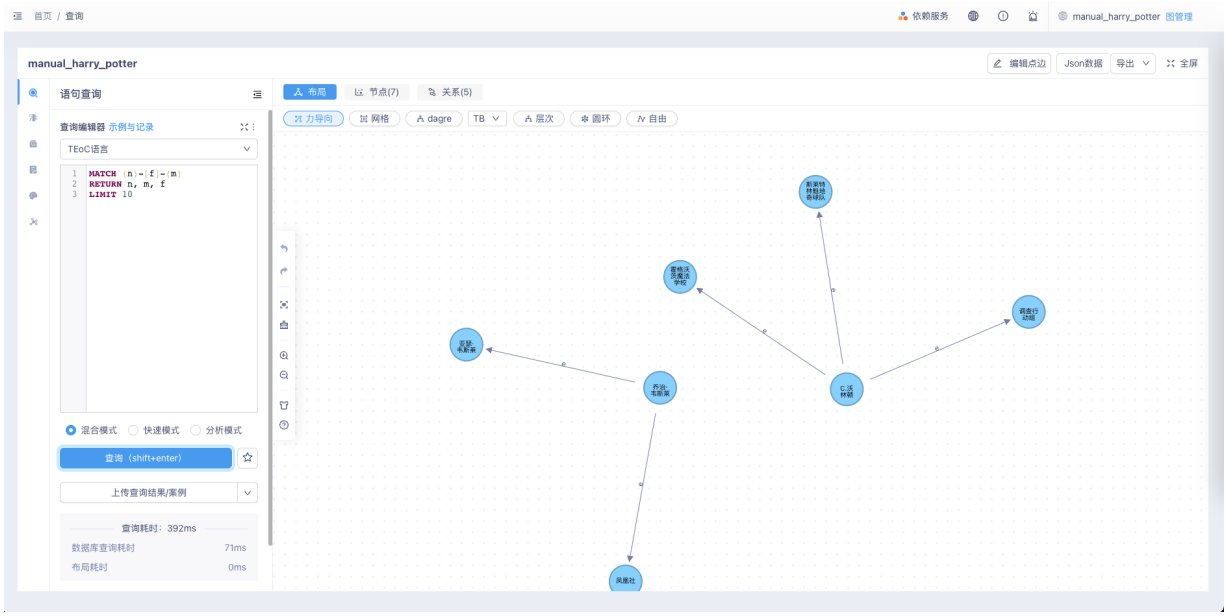
在图骨链页面，找到需要进行数据交互查询的图，在图名下方选择单栏点击“图探索”，即可进入2D查询页面。执行查询的过程为：选择查询模式 → 输入查询语句 → 点击 **查询** 按钮获得结果。KG Explorer支持多语句查询：语句编辑器输入多条语句，按分号 ; 分割语句即可。

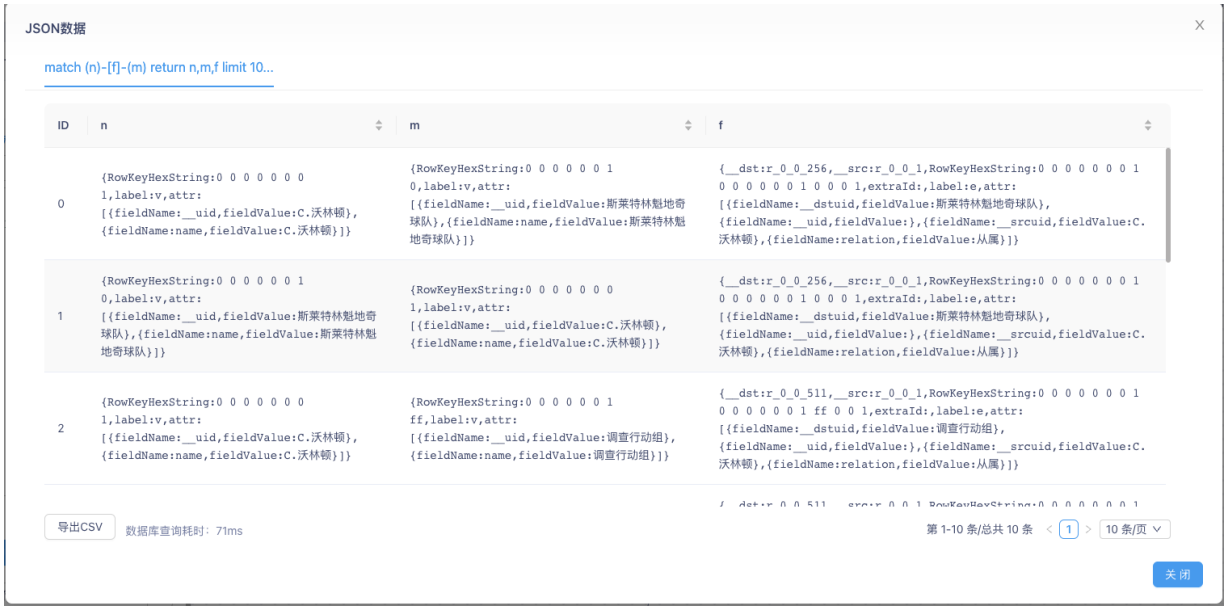


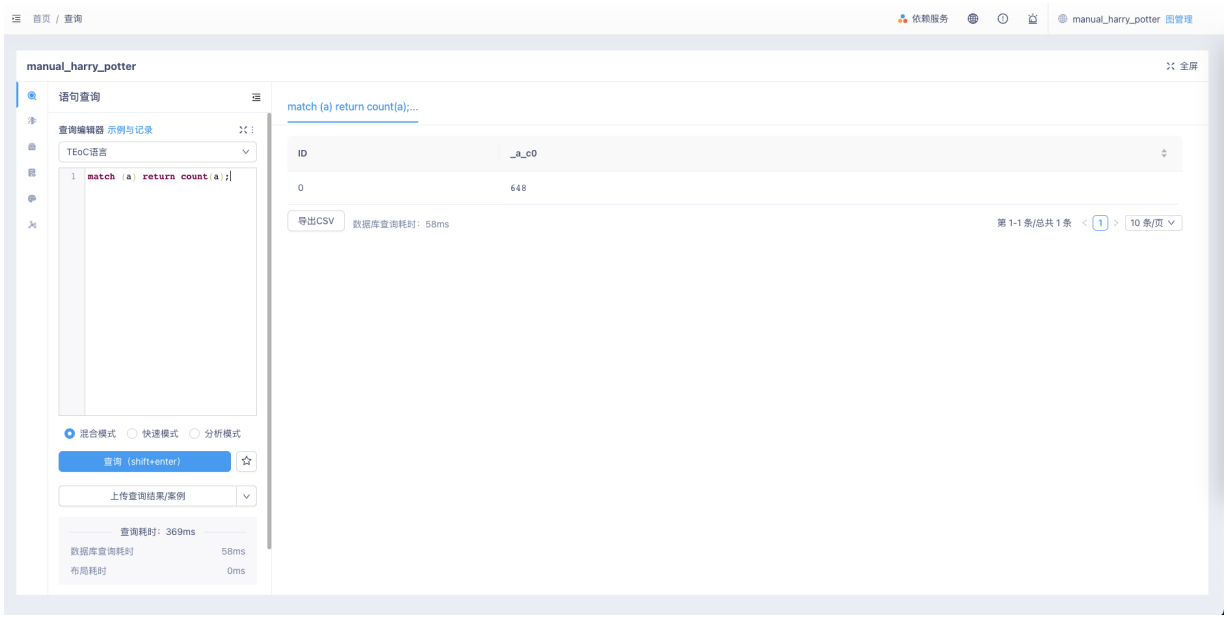
10.6.4.2. 查询结果

- 查询结果查看

若查询结果中包含点边数据，页面画布会展示点边。此时原始数据结果可以点击右侧 **结果相关** → **JSON数据** 查看。若查询结果不包含点边数据，页面上会以表格的形式显示原始查询结果。

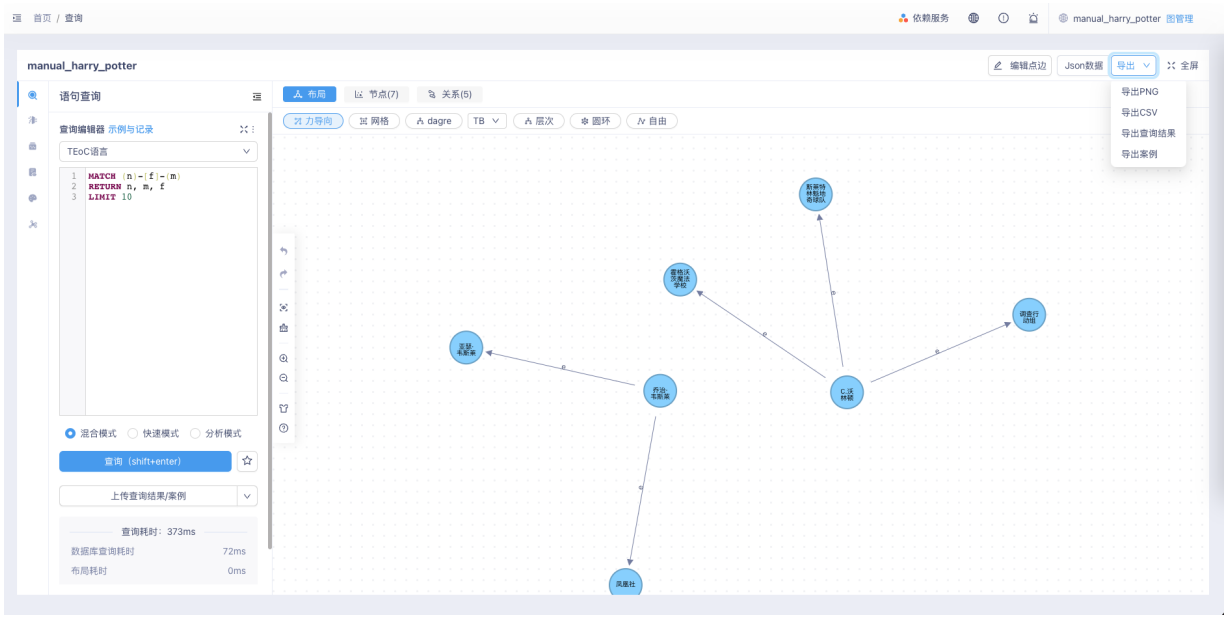






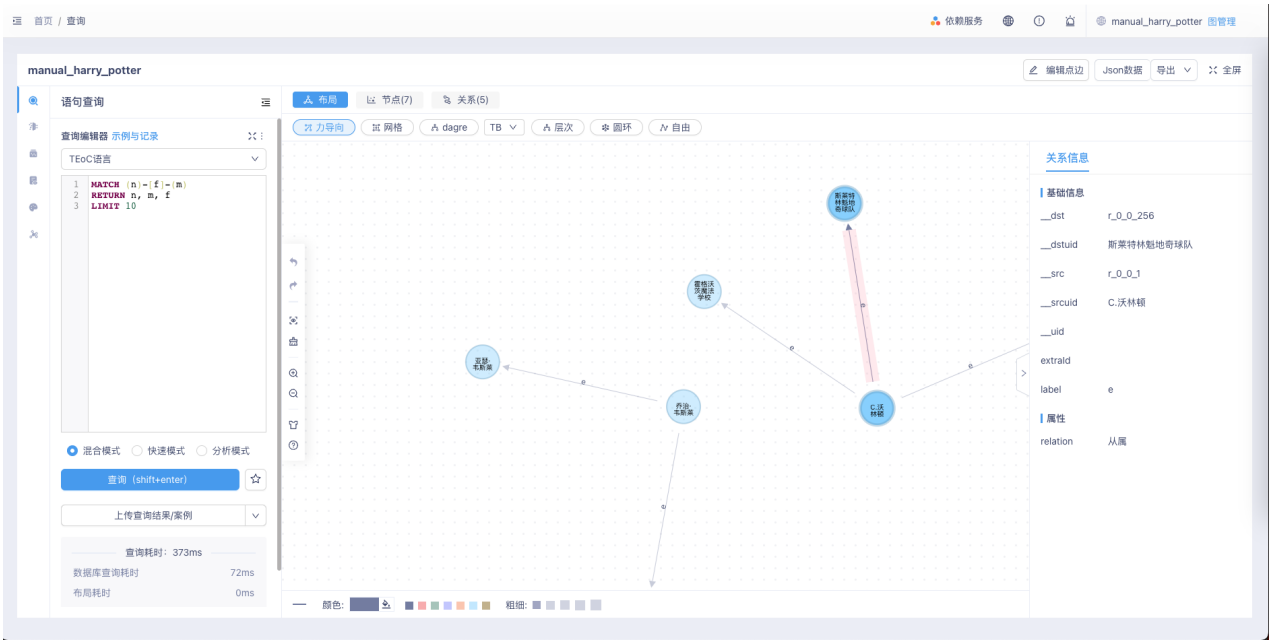
- 查询结果相关操作

1. 查询信息：点击查看该查询的图名、执行语句、查询模式、是否为多模查询、是否为自然语言的相关信息。
2. JSON数据：以表格形式查看原始查询结果。
3. 导出图谱：将画布的查询结果导出为CSV表或PNG格式的图片。



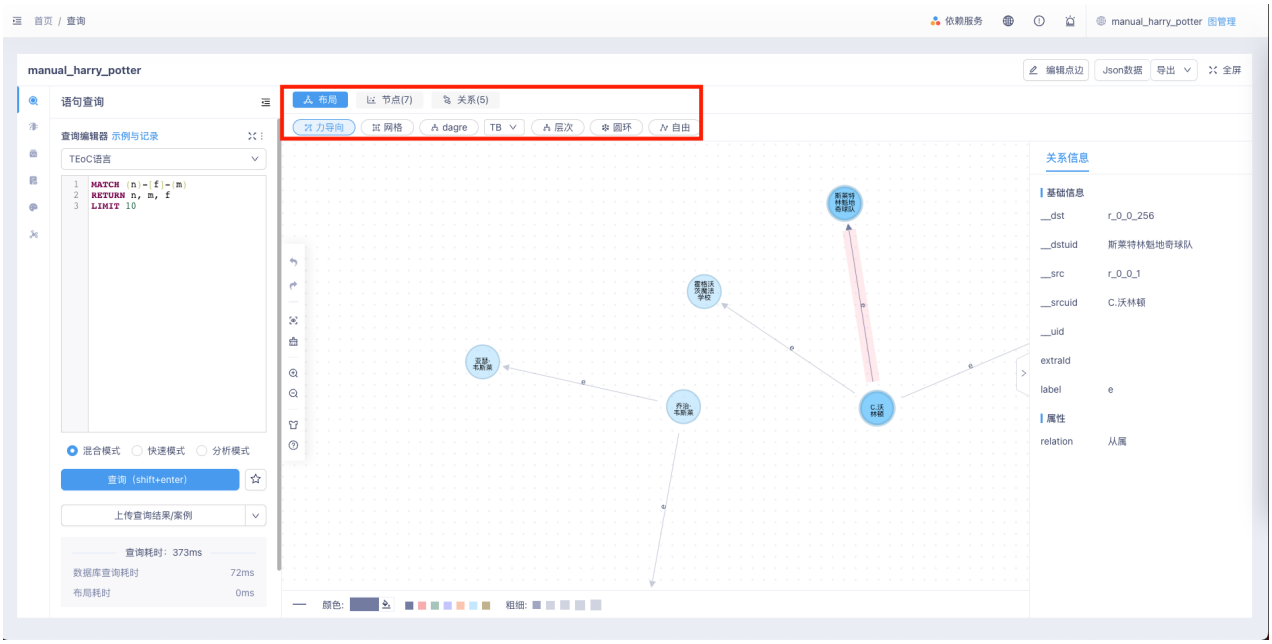
10.6.4.3. 点边信息展示

点击画布中的点或者边，画布右侧出现信息栏显示点或者边信息。



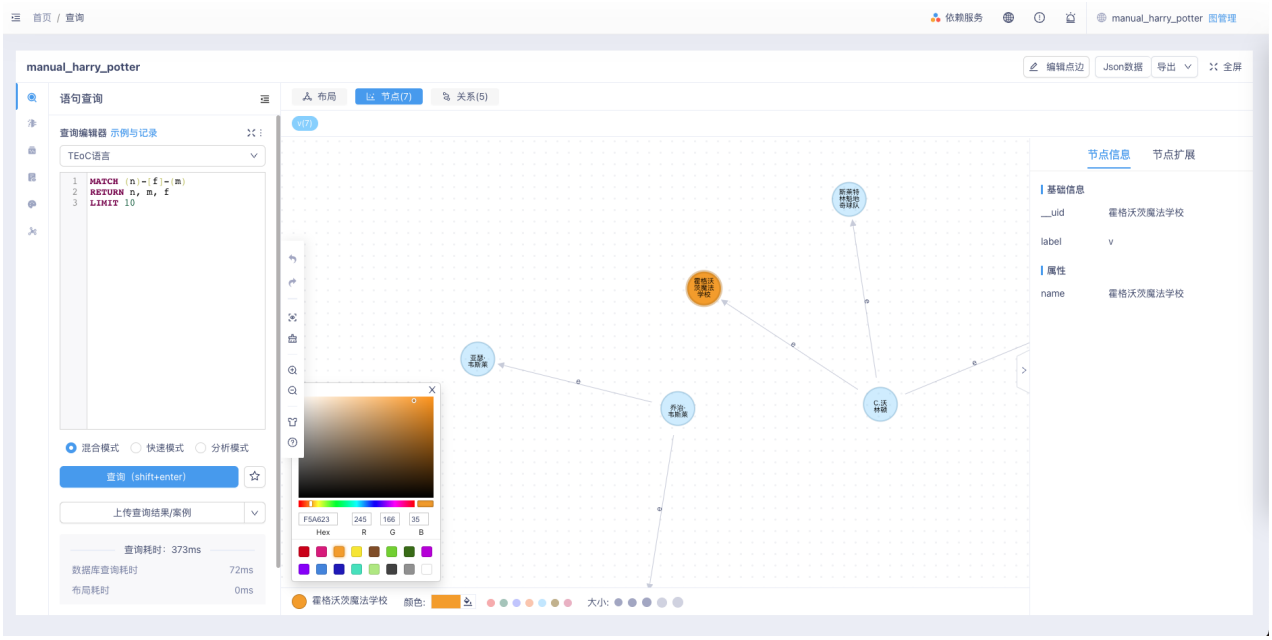
10.6.4.4. 切换布局

KG Explorer支持力布局、网格布局、层次布局、圆环布局、自由布局5种布局方式。



10.6.4.5. 自定义点边大小、颜色

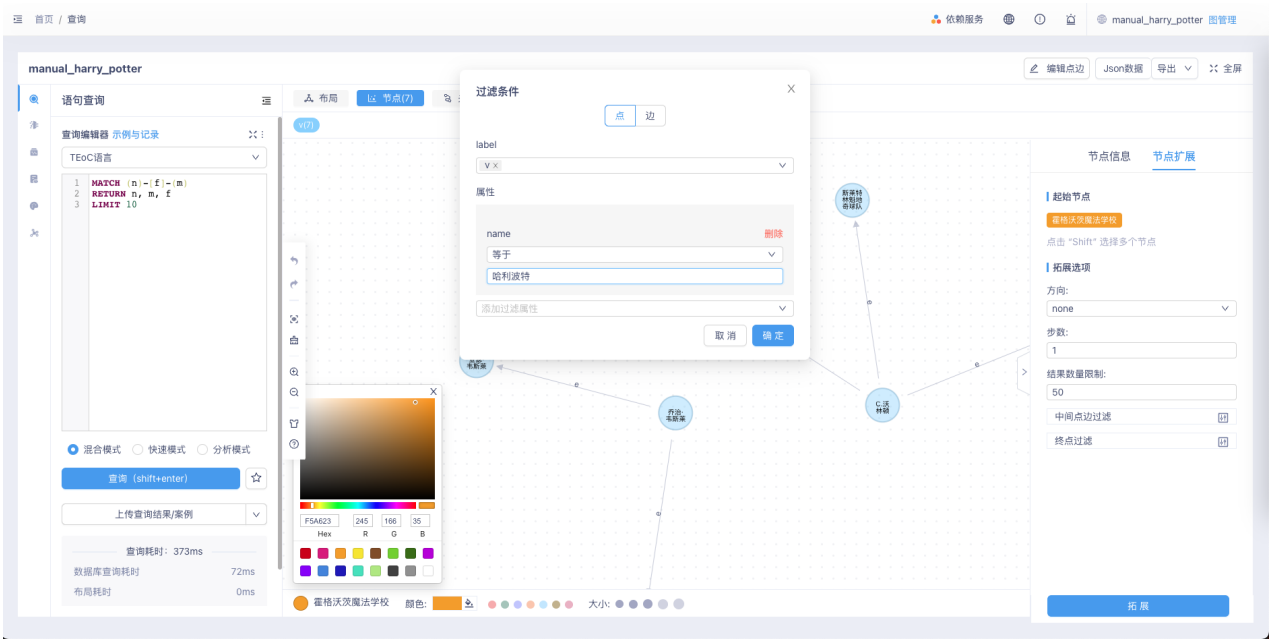
KG Explorer提供了按label和按单个点边两种自定义点边大小、颜色的形式。

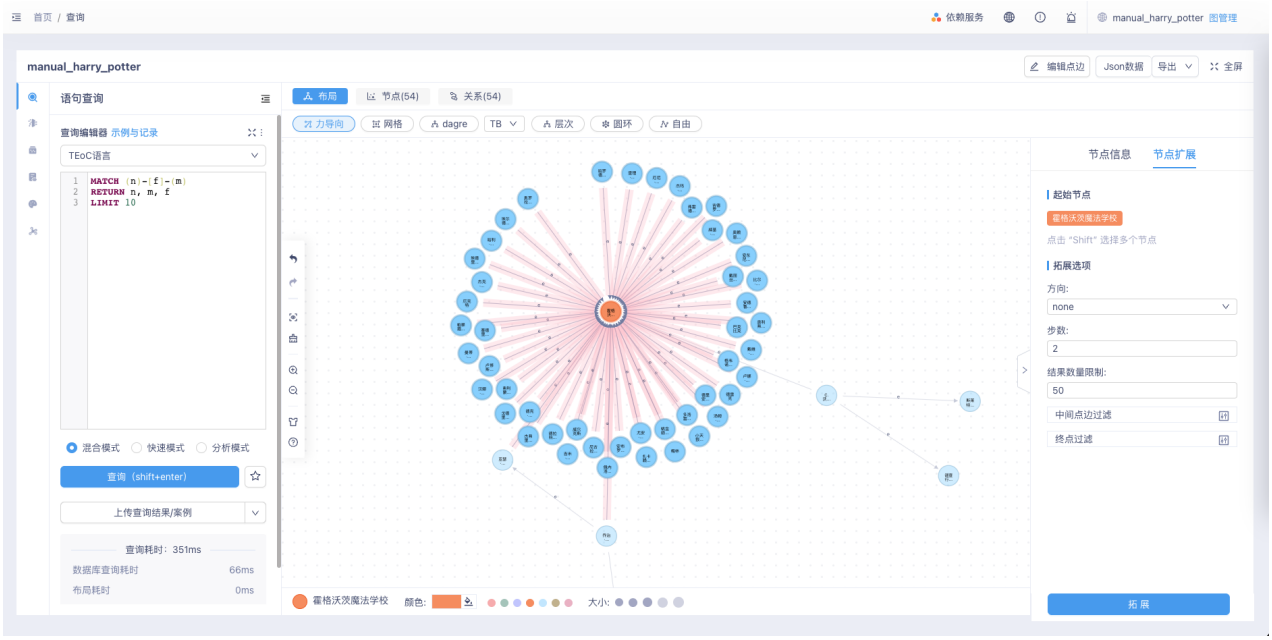


10.6.4.6. 节点扩展

在已有点的基础上，根据筛选条件拓展出相邻的点边。当前KG Explorer版本提供单层扩展和k层扩展功能：

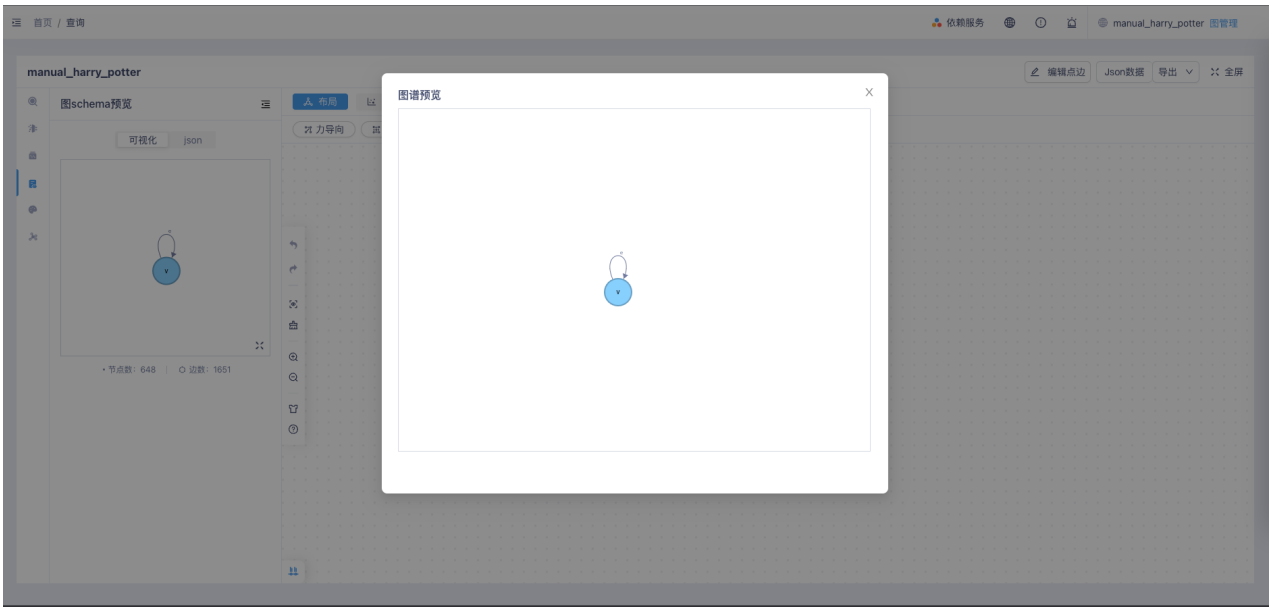
- 单层扩展：双击节点扩展
- k层扩展：可对选中节点进行k层扩展，扩展支持设置复杂过滤条件





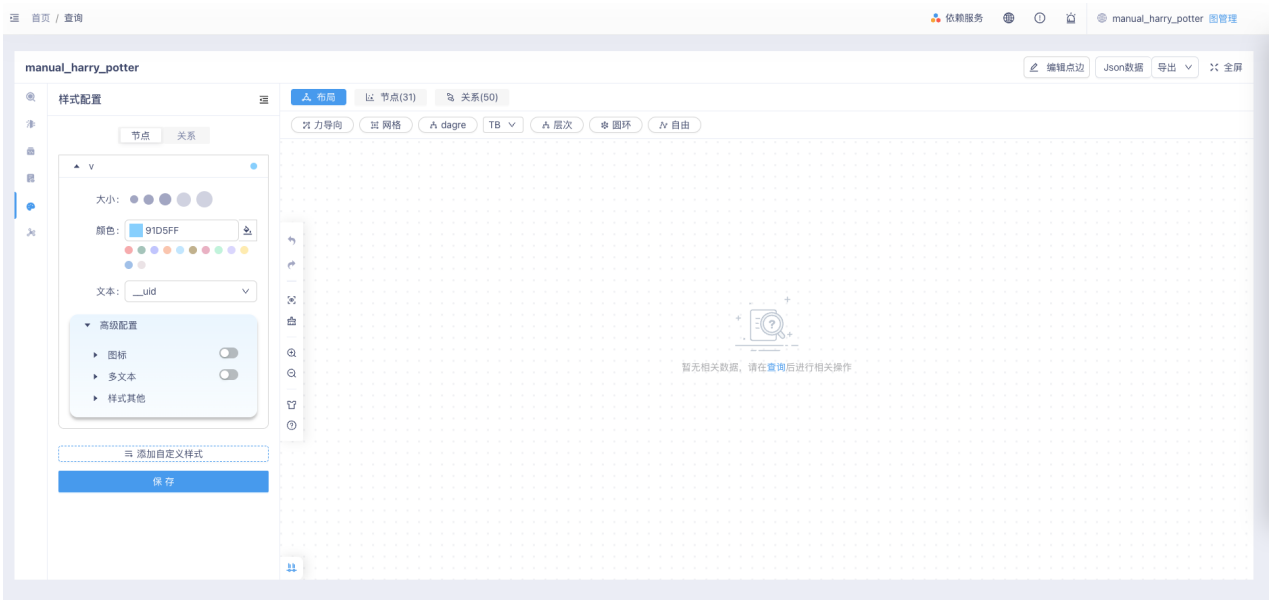
10.6.4.7. schema预览

KG Explorer提供schema预览功能，方便用户在图数据分析过程中灵活地浏览图schema。



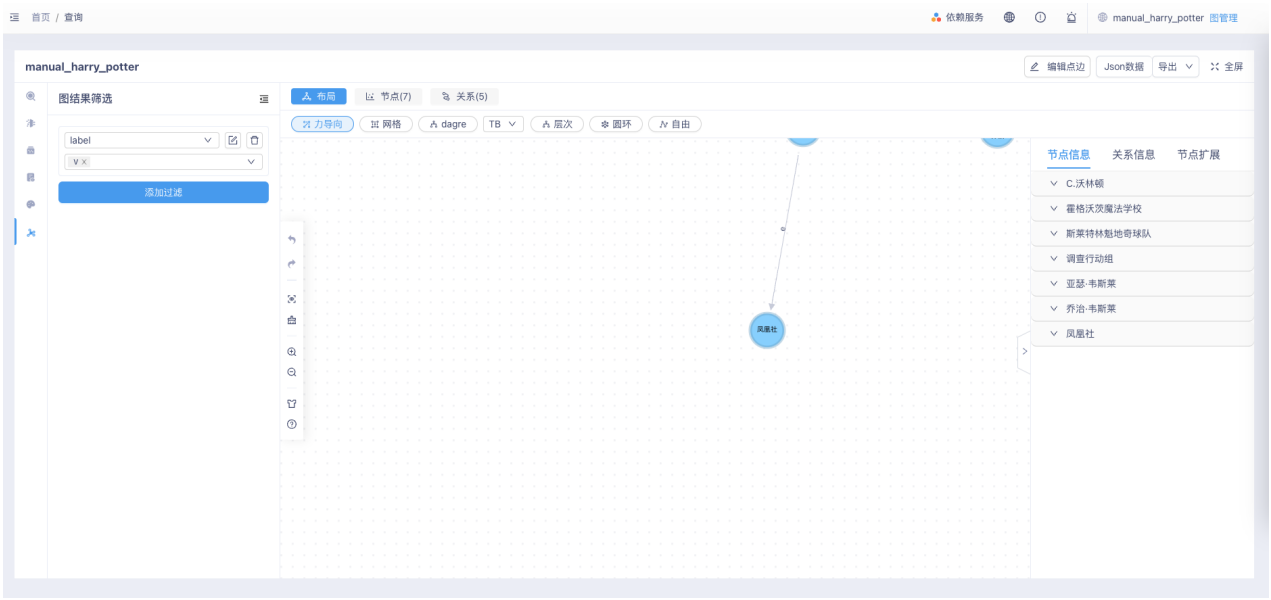
10.6.4.8. 样式配置

KG Explorer提供丰富的样式配置选项。在查询输入框左侧点击“样式配置”按钮后，会有样式配置菜单弹出。除了可以配置点/边的大小粗细、颜色等，还可以对节点应用图例、对点边属性应用数据变化的样式规则等



10.6.4.9. 结果筛选

KG Explorer提供对查询结果进行筛选高亮的功能。在查询输入框左侧点击“结果筛选”按钮打开结果筛选菜单。可以配置相应的规则，聚焦分析相关数据。

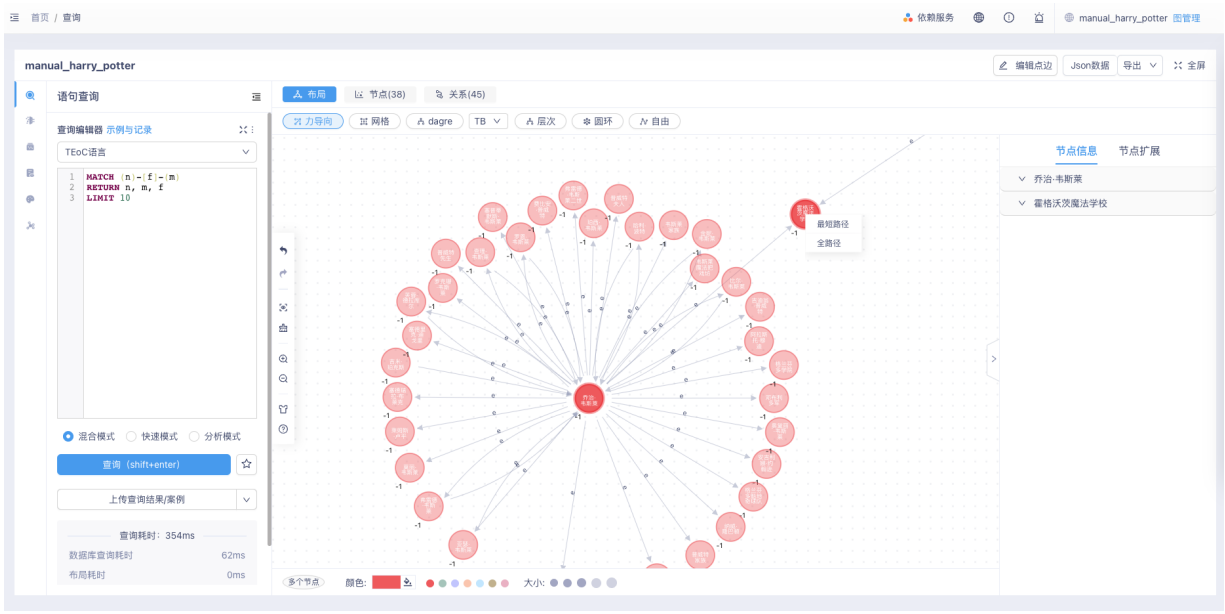


10.6.4.10. 最短路径和全路径展示

KG Explorer提供在画布中展示选中节点间的最短路径和全路径的功能。具体操作如下： 假设当画布中有至少两个节点存在时，鼠标单击选择一个节点A，按住shift键，用鼠标单击选择另一个节点B。之后，在任意选中的节点上点击鼠标右键打开菜单，可以看到有“最短路径”和“全路径”两个选项，最短路径和全路径的展示结果都是基于图数据库中的存储数据计算的，并非基于当前画布中的数据进行计算的。

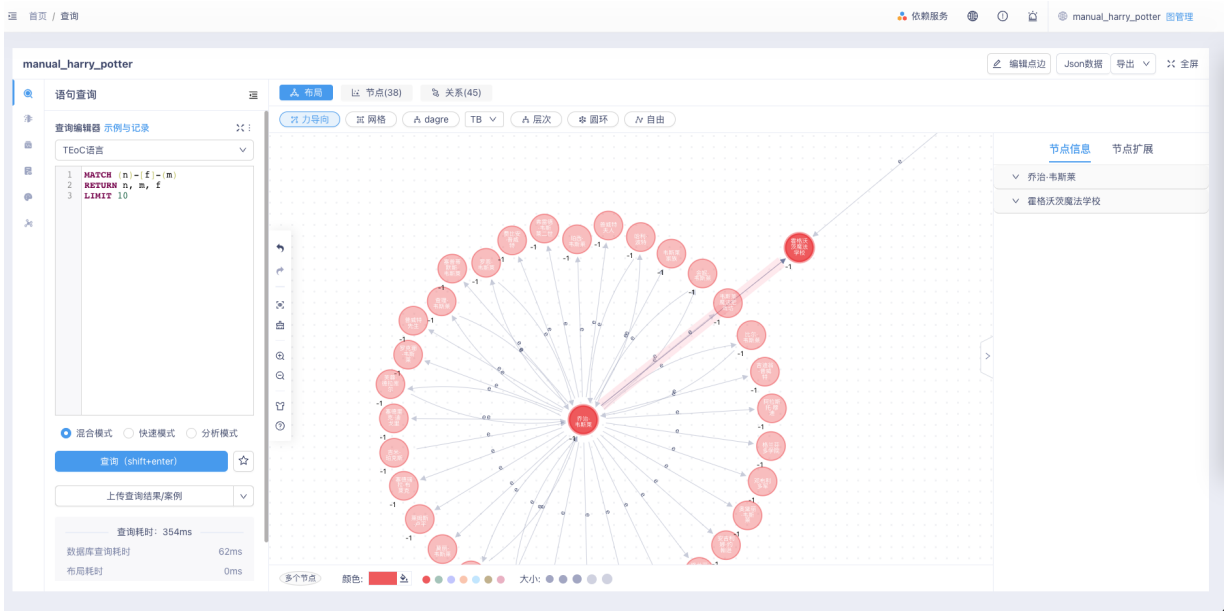
最短路径 展示两节点之间的最短路径:

1. 此处示例为从”乔治·韦斯莱“节点和”霍格沃滋魔法学校“节点之间的最短路径



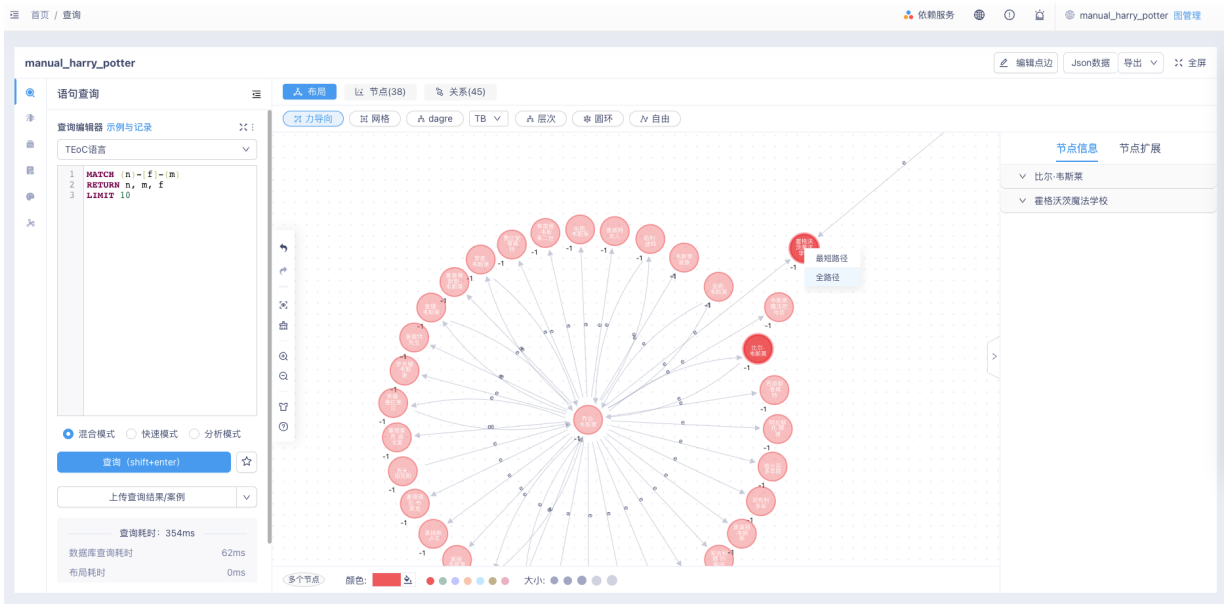
2. 点击最短路径选项后，弹出最短路径配置的选项框

3. 根据需要完成配置后，点击确定按钮，画布中会高亮显示最短路径



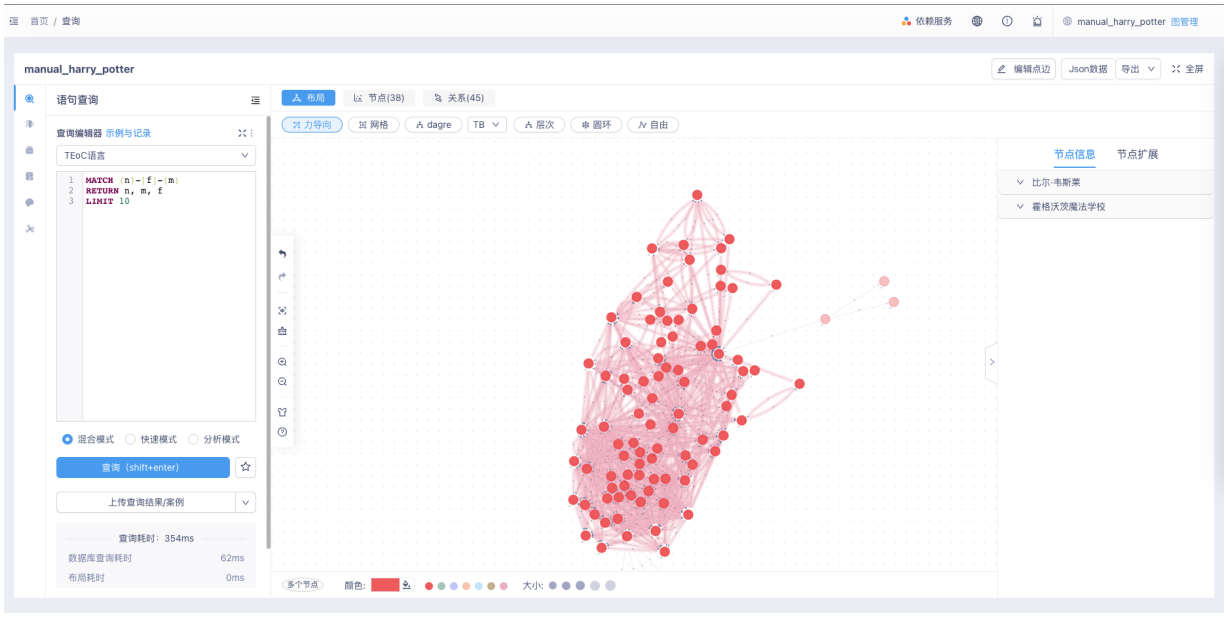
全路径 展示链接两节点之间的所有有效路径：

1. 此处示例为从”比尔 • 韦斯莱“节点和”霍格沃滋魔法学校“节点之间的所有路径

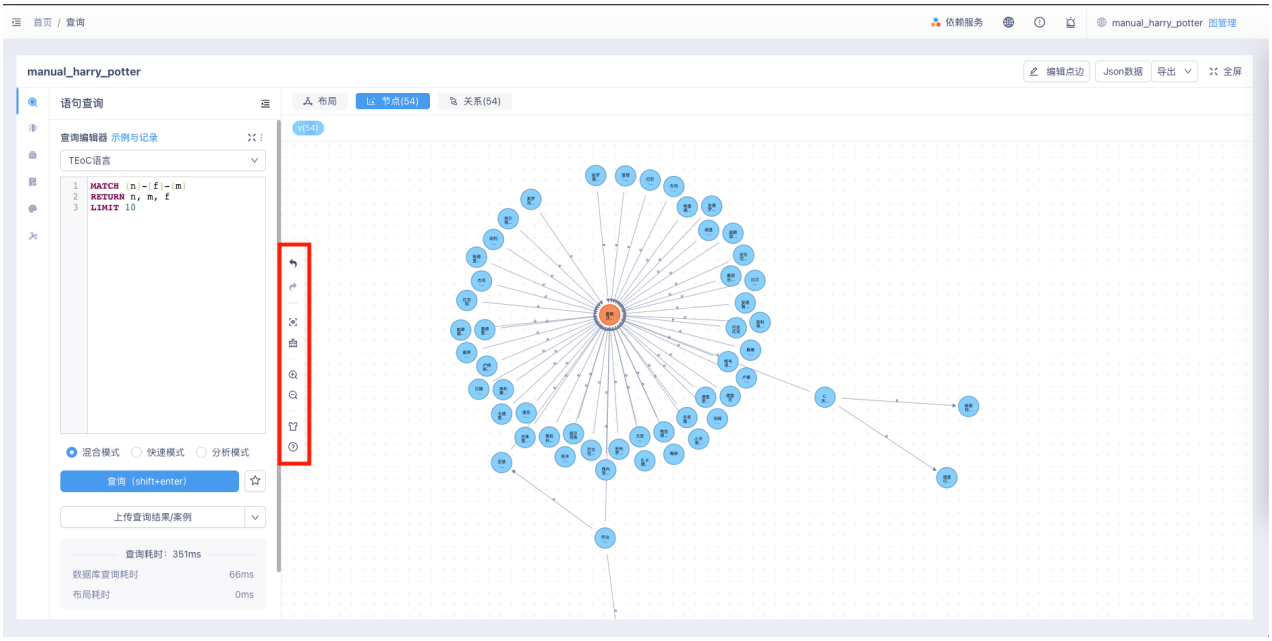


2. 点击全路径选项后，弹出全路径配置的选项框

3. 根据需要完成配置后，点击确定按钮，画布中会高亮显示所有满足条件的路径



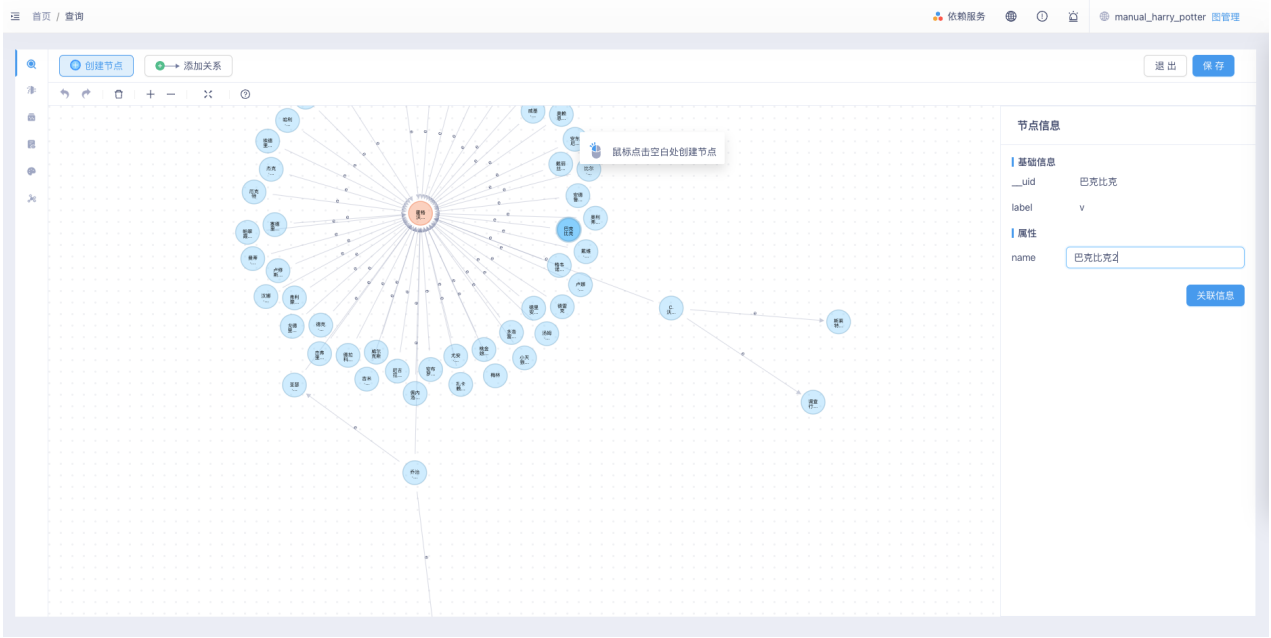
10.6.4.11. 画布工具栏

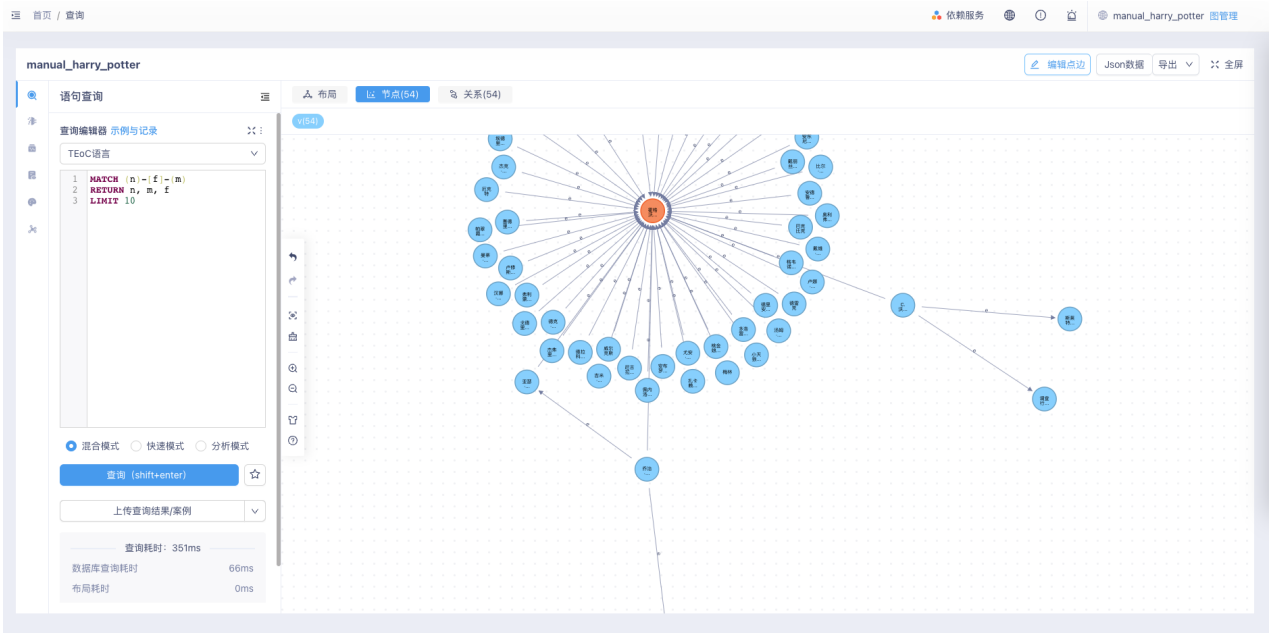


工具名称	使用说明
undo	撤销上一步操作
redo	重做上一步操作
视图居中	画布中点边居中显示
清空画布	清除画布中的内容
放大	画布中数据放大显示
缩小	画布中数据缩小显示
换肤	更改画布背景
快捷键说明	查看格式化查询语句等快捷操作说明

10.6.4.12. 编辑点边

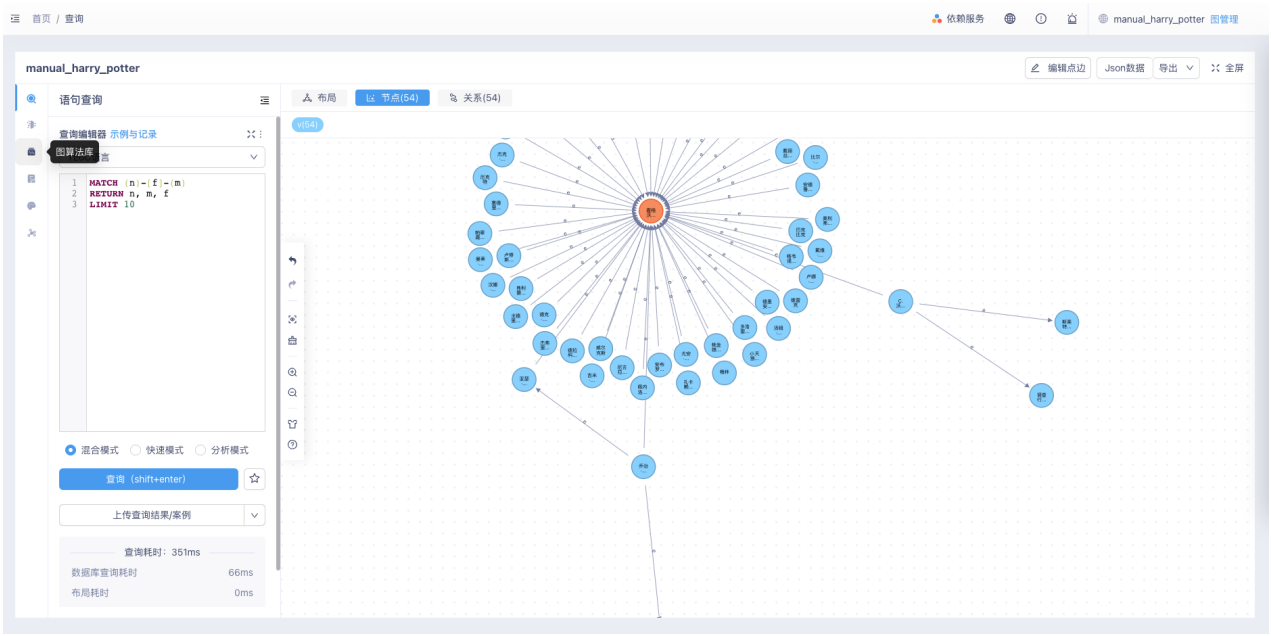
编辑点边功能主要用于增加、删除、修改点边信息。 可通过点击上方编辑菜单栏右侧按钮 **编辑点边**，修改查询出的点边的属性值以及添加点边。按照引导进行编辑后点击保存，结果即同步修改到数据库。

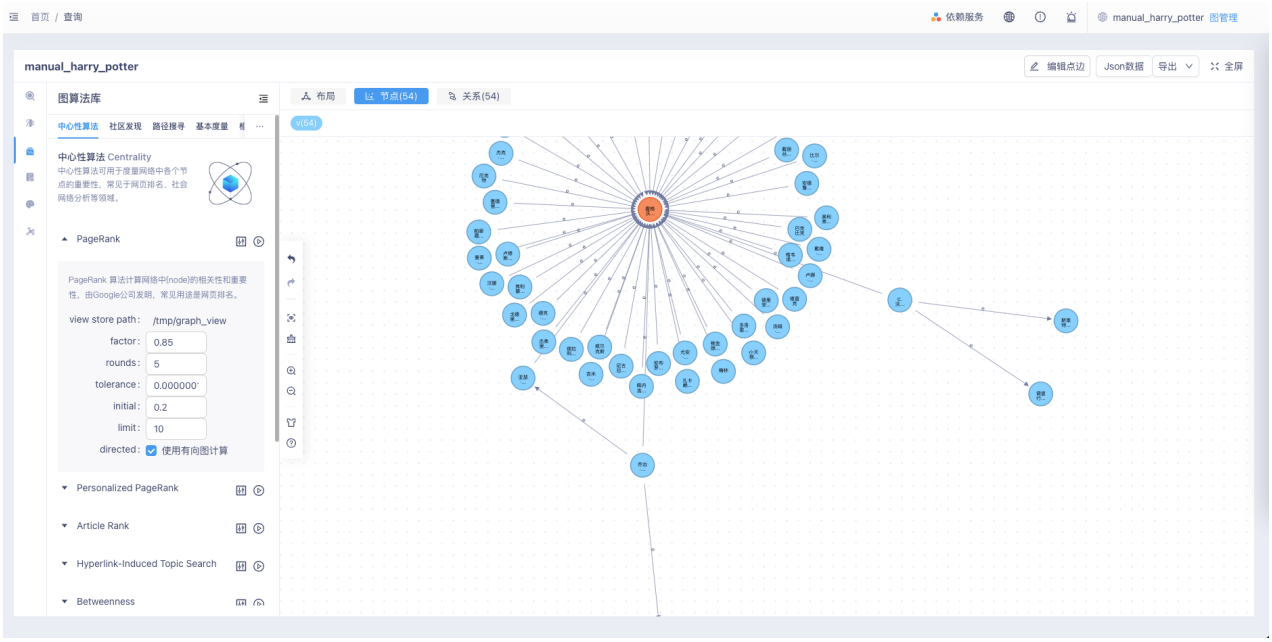


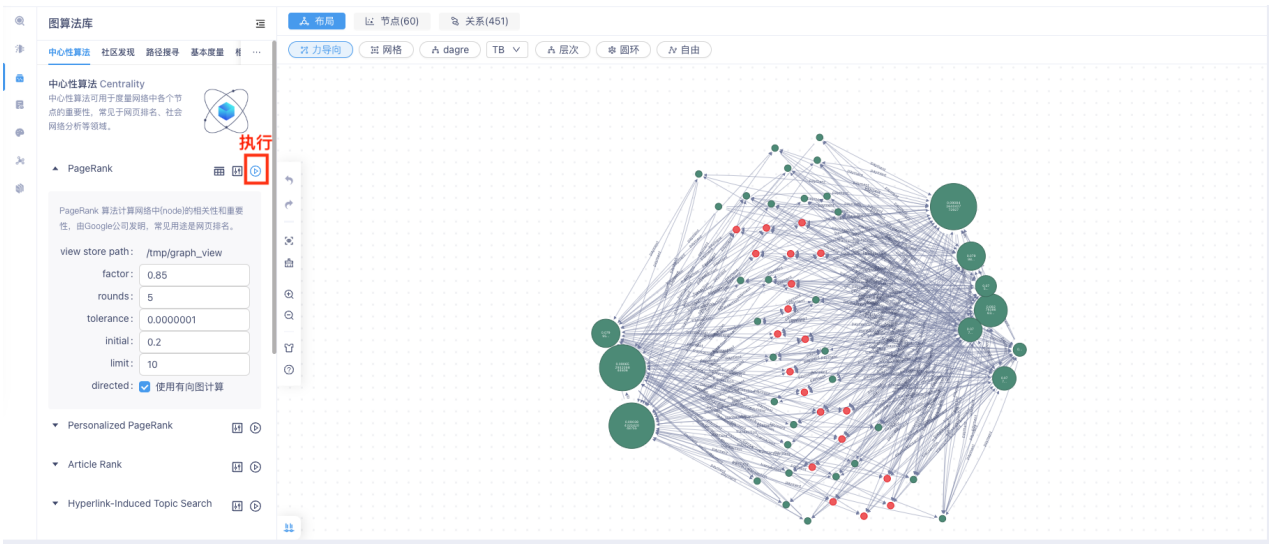


10.6.4.13. 执行图算法

点击查询编辑框左边的 **图算法库** 按钮，左侧会展示当前支持页面执行的所有图算法（图数据库支持的图算法的子集），可以在此处配置参数、配置过滤条件、执行图算法。

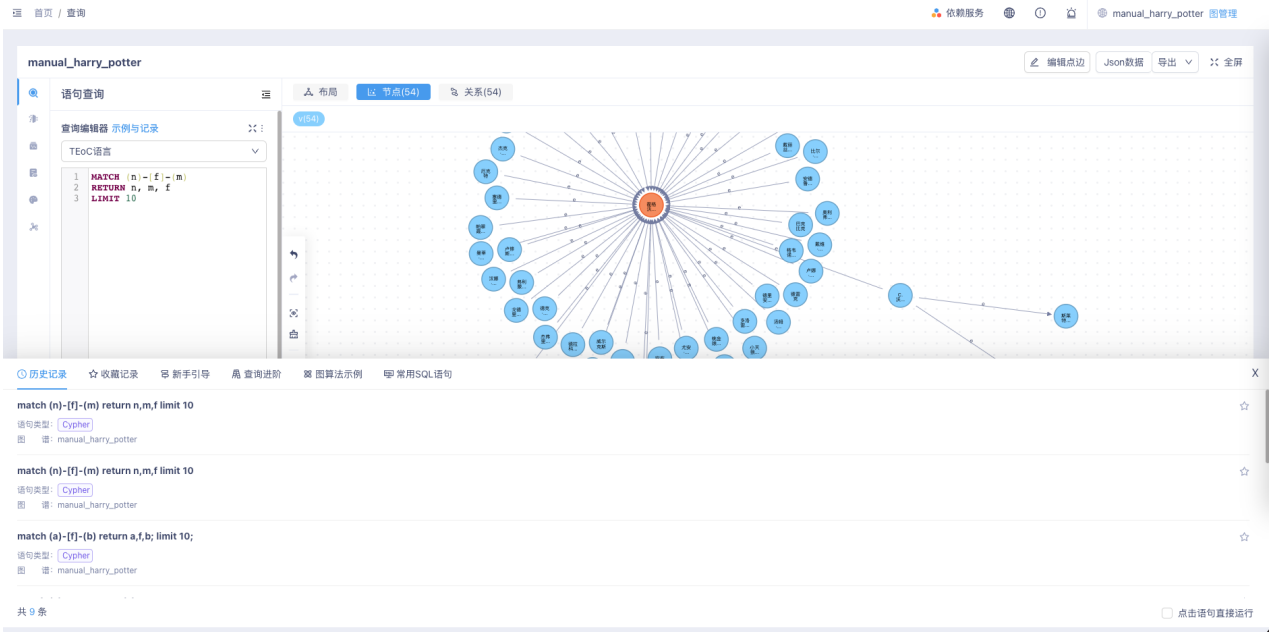

首页 / 查询
依赖服务
manual_harry_potter 图管理
manual_harry_potter
编辑点边
Json数据
导出
全屏
语句查询
图算法库
查询编辑器 示例与记录
MATCH (n)-[f]-(m)
RETURN n, m, f
LIMIT 10
布局
节点(54)
关系(54)
v(54)
混合模式
快速模式
分析模式
查询（shift+enter）
上传查询结果/案例
查询耗时：351ms
数据库查询耗时 66ms
布局耗时 0ms


首页 / 查询
依赖服务
manual_harry_potter 图管理
manual_harry_potter
编辑点边
Json数据
导出
全屏
图算法库
布局
节点(54)
关系(54)
v(54)
中心性算法
社区发现
路径搜寻
基本度量
中心性算法 Centrality
中心性算法可用于度量网络中各个节点的重要性，常见于网页排名、社会网络分析等领域。
PageRank
PageRank 算法计算网络中(node)的相关性和重要性，由Google公司发明，常见用途是网页排名。
view store path: /tmp/graph_view
factor: 0.85
rounds: 5
tolerance: 0.000000
initial: 0.2
limit: 10
directed: 使用有向图计算
Personalized PageRank
Article Rank
Hyperlink-Induced Topic Search
Betweenness


图算法库
布局
节点(60)
关系(451)
力导向
网格
dagre
TB
层次
圆环
自由
中心性算法
社区发现
路径搜寻
基本度量
中心性算法 Centrality
中心性算法可用于度量网络中各个节点的重要性，常见于网页排名、社会网络分析等领域。
执行
PageRank
PageRank 算法计算网络中(node)的相关性和重要性，由Google公司发明，常见用途是网页排名。
view store path: /tmp/graph_view
factor: 0.85
rounds: 5
tolerance: 0.0000001
initial: 0.2
limit: 10
directed: 使用有向图计算
Personalized PageRank
Article Rank
Hyperlink-Induced Topic Search

10.6.4.14. 查看查询语句记录

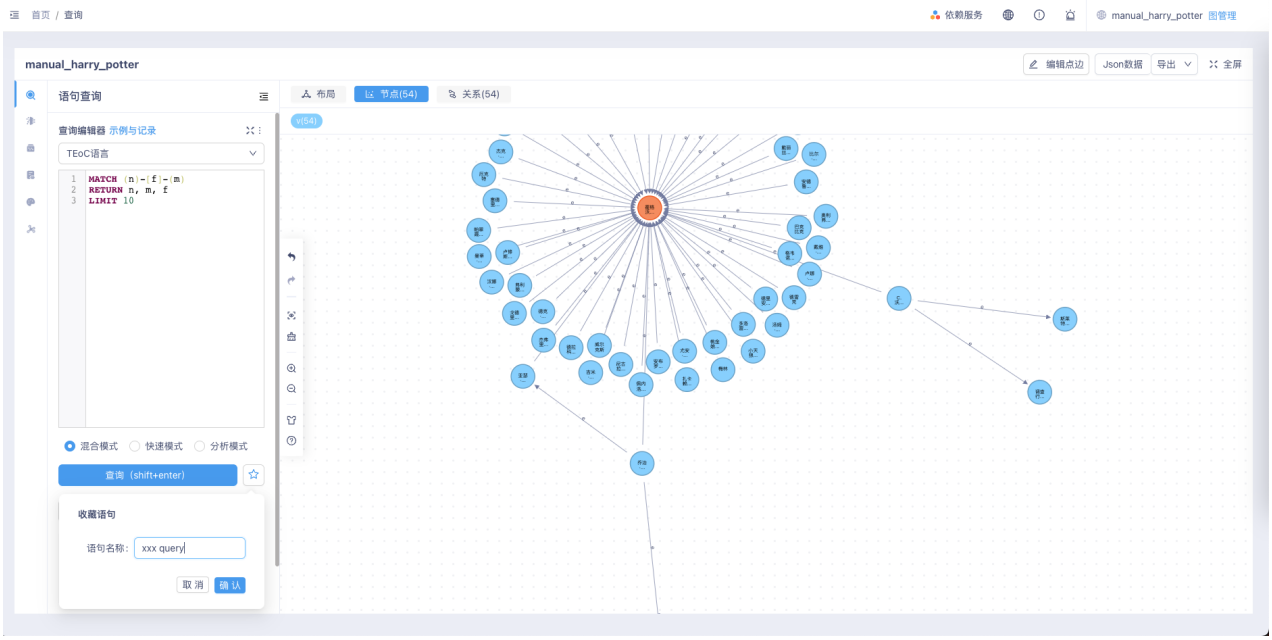
点击查询编辑框上方 **查询与记录** 按钮可以查看最近的查询记录。



在弹出窗口中还可以查看所有示例语句以及保存过的语句。

10.6.4.15. 语句收藏

在查询编辑框中输入查询语句后，点击 **查询** 按钮右侧的 **收藏** 按钮，可以对当前输入框中的语句进行收藏。点击收藏按钮后，只需要输入查询的命名即可。

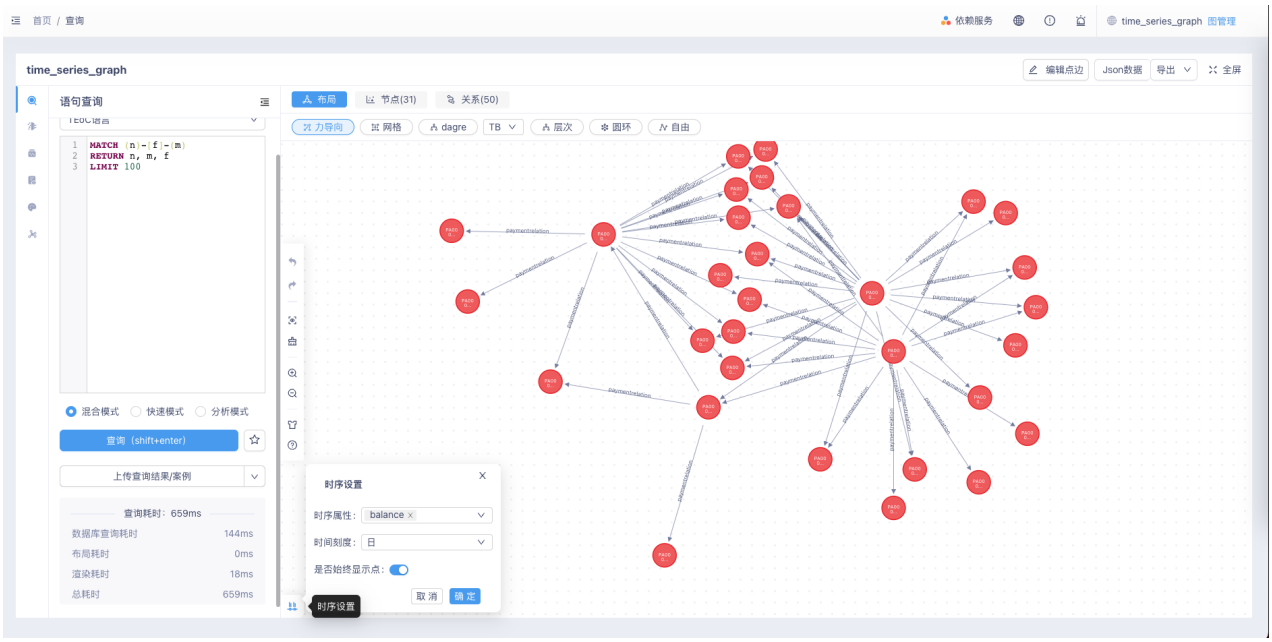


10.6.5. 动态图时间轴可视化

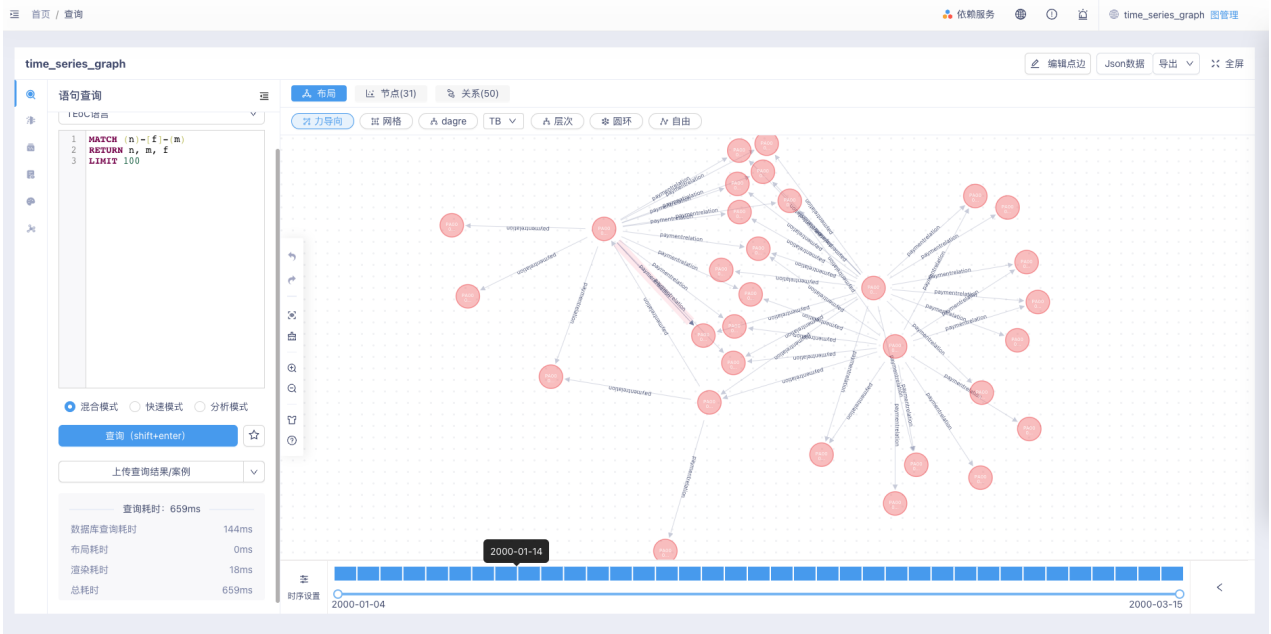
KG Explorer引入了动态图的时间轴可视化功能，用于可视化展示动态图数据变化过成。

10.6.5.1. 配置时间轴

在查询过程中，KG Explorer会自动判断当前图是否为动态图，并且在画布左下角提供时间轴配置按钮。



时间轴配置按钮可配置动态图动态展示的主要变化的点或者边的属性，时间轴中的时间片为所有选中的属性所有的时间片；其次，时间轴的配置还可以配置时间片的尺度：年、月、还是日，那么在时间轴变化过程中，只有时间篇对应的尺度发生变化后，图才可以可视化地看到图数据的动态变化。



时间轴可以选择连续的时间片，或者离散的若干时间片/段。在选择时间片和取消时间片的操作过程中，便可以看到图数据的动态变化。

需要注意的是，当前版本中，点/边的动态变化还遵循下述规则：当一条边和其两个端节点出现任意两个对象的时候，第三个对象必须出现。所以会出现某条边不在所选时间片出现的情况。

10.6.5.2. 案例上传和下载

动态图时间轴可视化功能常常会配合丰富的样式配置和协同合作，因此需要对当前数据分析案例具备可下载以及未来可上传的功能。

动态图时间轴可视化功能的案例下载内容会包含当前画布中全量的分析数据、样式配置信息、图信息等。

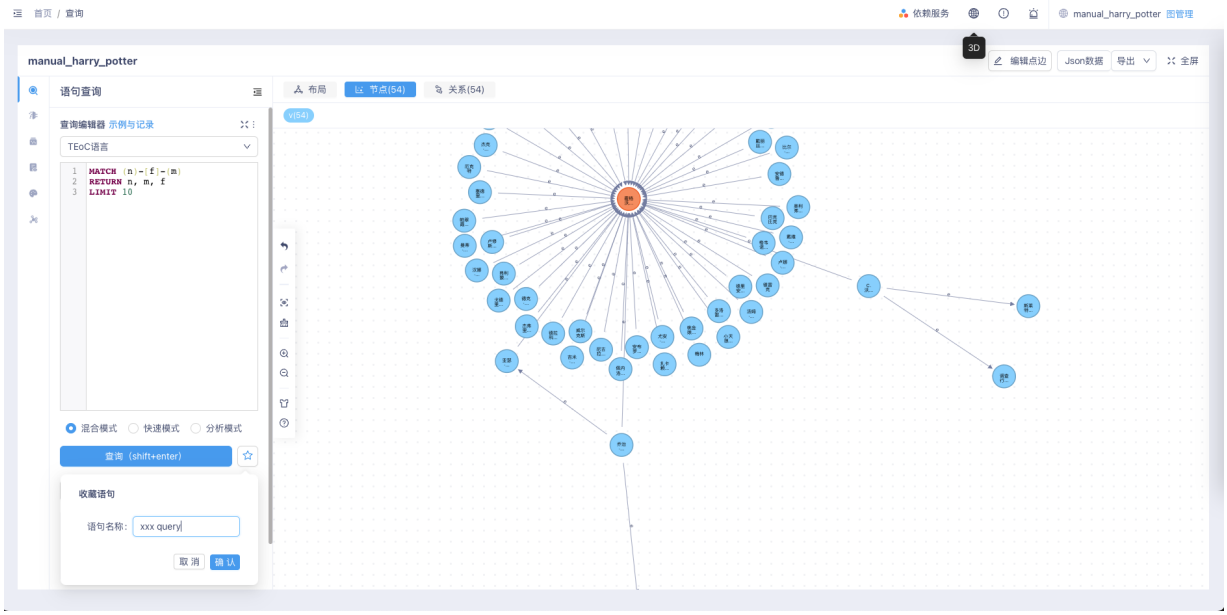
案例上传功能则通过解析文件中的信息对数据进行恢复，从而可以使得使用人员继续在原有案例上进行分析。

10.6.6. 3D查询分析展示

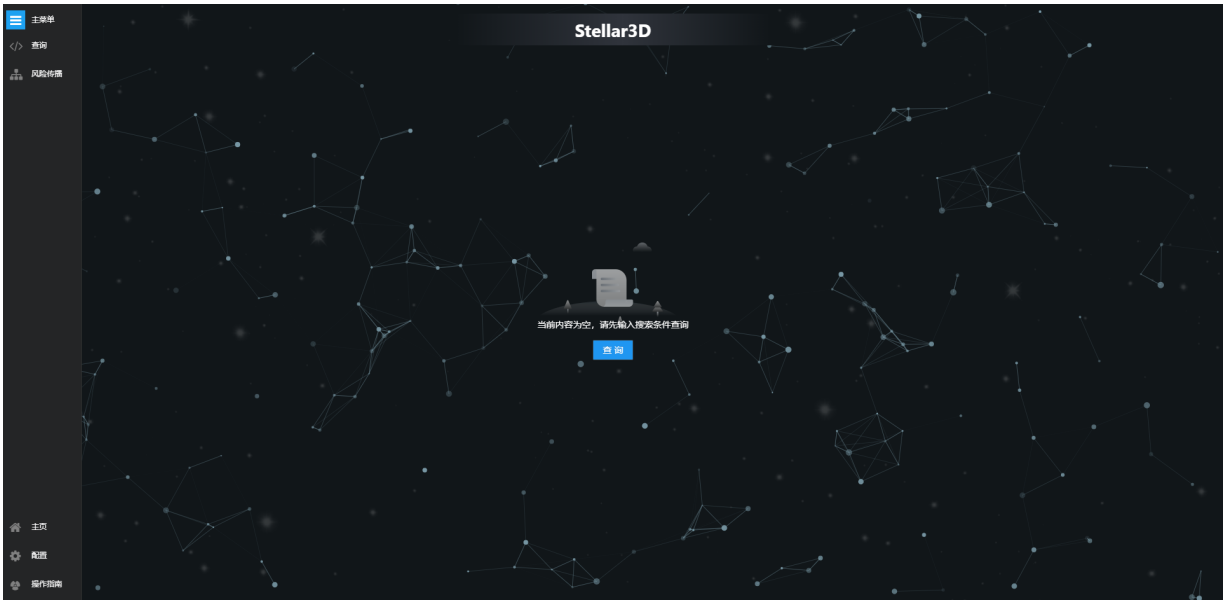
KG Explorer提供TEoC查询接口，通过3D形式展示查询结果，提供针对查询结果的各项页面操作。

- 切换至Stellar3D界面

  通过KG Explorer图管理主页或者任意图的图探索页面右上方 **3D** 图标进入3D展示页面。

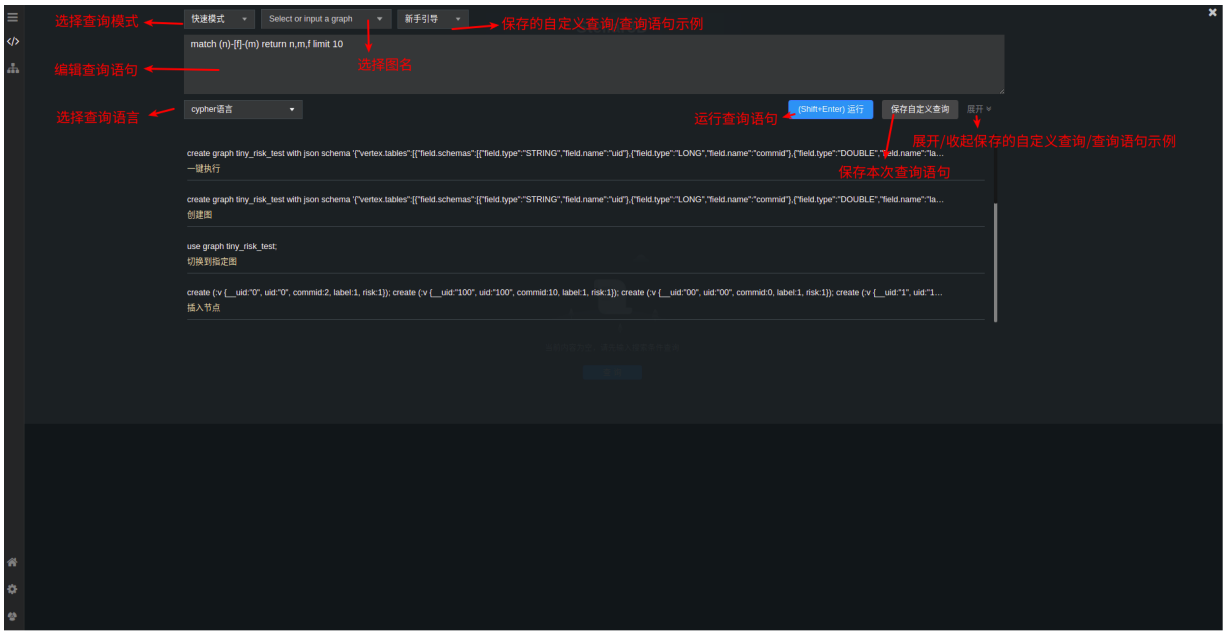


- Stellar3D主页左侧菜单栏功能说明



按钮	说明
菜单折叠按钮	收缩、展开菜单
查询按钮	点击弹出查询框，支持TEoC，SQL查询
风险传播按钮	用于展示风险传播过程，为定制化内容
返回KG Explorer主页的按钮	点击返回KG Explorer主页
参数配置按钮	可以配置最大展示数据量、风险传播需要的字段，后续将开放更多的自定义配置参数
Stellar3D操作指南按钮	包含Stellar3D相机操作、节点操作、节点簇操作等的操作说明

- Stellar3D查询界面介绍

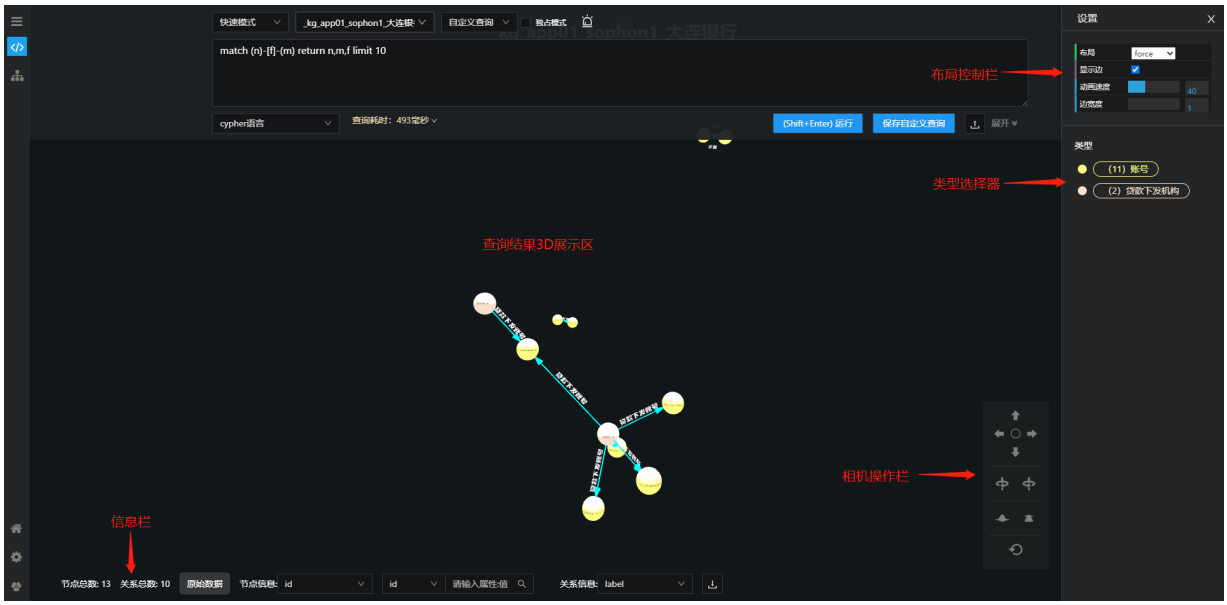


页面组件	说明
查询模式选择下拉框	快速模式:单机执行查询，适合简单查询，一般在60s以内。 分析模式:集群模式执行查询，适合数据量超过100000或者时间大于30s的查询。 测试模式:用于查看前端3D效果，非真实数据，由前端直接计算，查询语句中的数字表示此次观测的数量。
图名选择下拉框	选择查询的图名
查询历史记录以及查询示例下拉框	自定义查询：用户保存的自定义查询语句。 新手引导：基本的TEoC语句示例，让用户快速上手查询。 TEoC进阶：稍有难度的TEoC语句示例，让用户进一步熟悉查询。 图算法示例：基本的图算法示例。 自然语言示例：基本的自然语言示例。 常用SQL语言：基本的SQL语句示例。
语句输入框	在此输入查询语句
语言选择下拉框	可以切换查询语言，包括TEoC语言、SQL语言，以及自然语言（自然语言为定制功能，需要定制开发）
运行查询按钮	点击执行输入的查询语句
保存自定义查询	将当前查询语句保存到自定义查询
上传JSON按钮	可以将导出的查询结果JSON上传，画布上会恢复展示对应的结果内容

- 基本的查询操作流程：

  选择查询语言 → 选择查询模式 → 选择图谱（测试模式下可不选） → 编辑查询语句 → 运行查询语句 → 得到查询结果。同时语句编辑器也支持多语句查询，只需在输入框中输入多条语句，按分号 ; 分割语句即可。

- Stellar3D查询结果展示页面按钮功能说明



1. 点击右上角的 **布局控制** 按钮可以打开布局展示设置功能栏，操作说明如下：

按钮	说明
布局选择下拉框	切换不同的布局，查询结果3D展示区的点边会按该布局的方式排列 布局包括： normal（初始布局）、layer（层级布局）、grid（栅格布局）、cluster（簇布局）、cylinder（圆柱体布局）、circle（圆环布局）、sphere（球形布局）、random（随机布局）
是否显示边单选框	可以控制画布上是否展示边
边透明度控制条	可以调节画布上边的透明度
动画速度控制条	可以控制画布上动画的速度
边宽度	可以调节画布上边的宽度
类型选择器	点击右边的小圆圈可以自定义对应类型节点的颜色。点击类型名可以在画布上高亮选中这一类型的点

2. 下方功能栏上的组件功能说明如下：

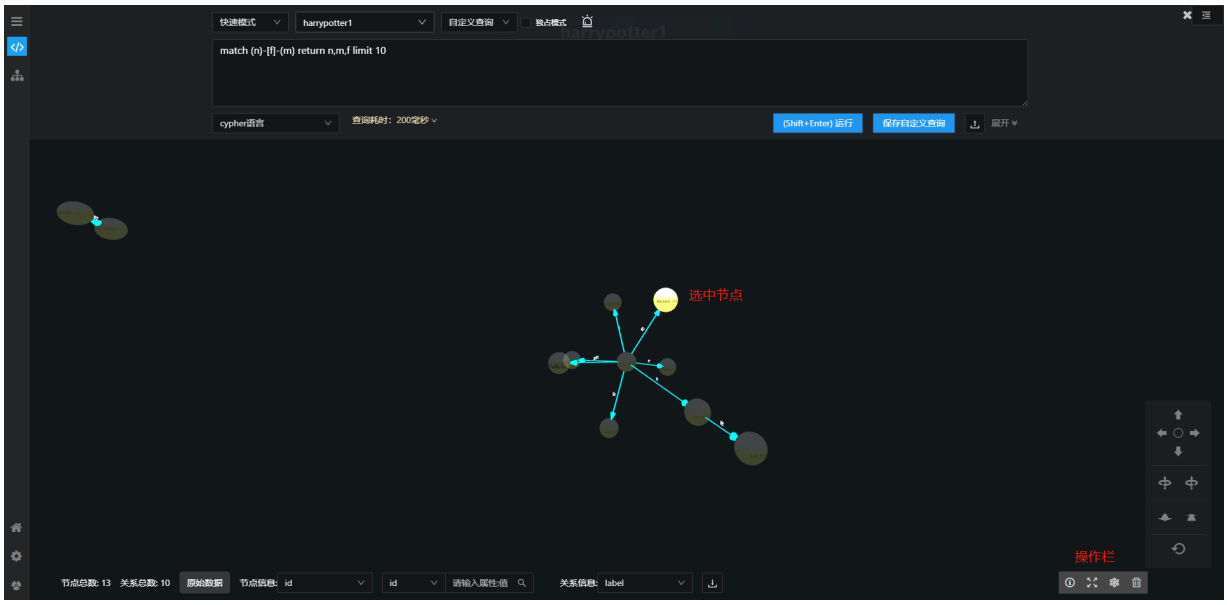
按钮	说明
点边数目展示	包括点总数、边总数
原始数据按钮	点击可以查看查询返回的原始数据
节点信息	可以选择在点上展示的内容
节点查找	按id/userId/label/属性值查找出对应点，并高亮聚焦到对应点
关系信息	可以选择在边上展示的内容
导出按钮	可以将查询结果导出，导出格式支持PNG、CSV和JSON

3. 右下侧的相机操作区域功能说明

   模拟静态图像在现实世界的运动，提供不同视角观点下的3D结果展示

   上：鼠标右键往上 +
   下：鼠标右键往下 +
   左：鼠标右键往左或键盘A +
   右：鼠标右键往右或键盘D +
   绕Y轴左旋转：键盘Q +
   绕Y轴右旋转：键盘E +
   放大：鼠标滚轮向前 或者 键盘W +
   缩小：鼠标滚轮向后键盘S +
   绕中心任意旋转：鼠标左键空白处划动 +

- Stellar3D画布上节点和边的操作介绍



选中点或边后显示操作栏，操作栏上的功能说明如下：

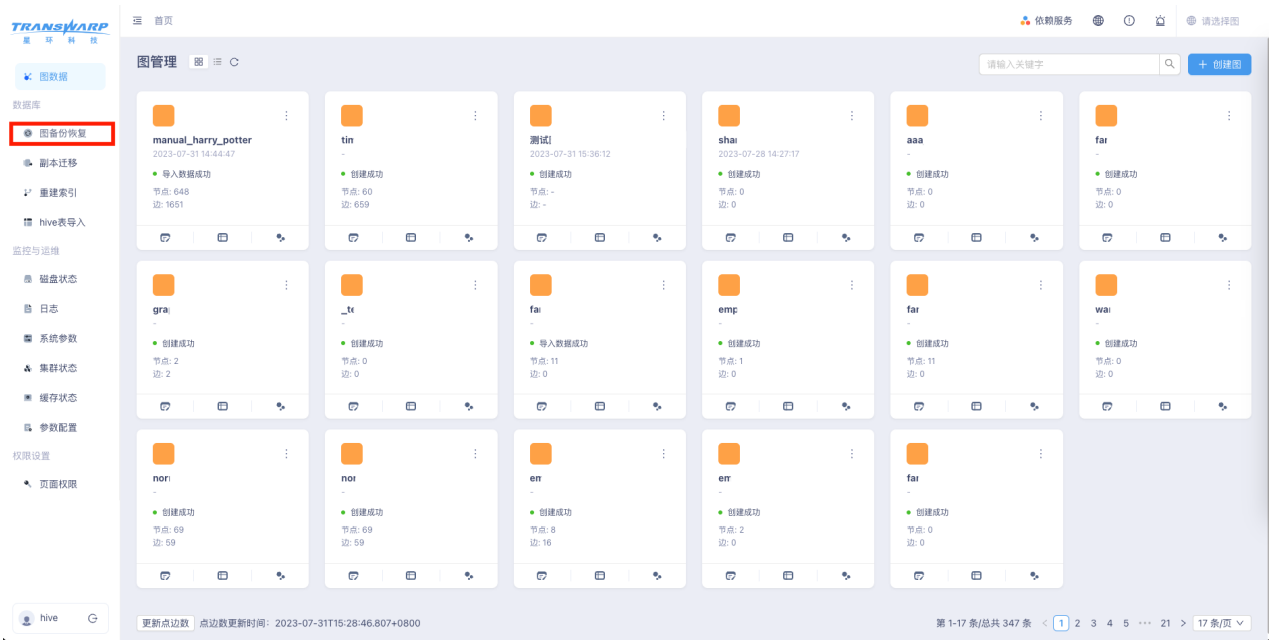
按钮	说明
节点详情	点击可查看该节点的基本信息（双击节点可快速查看节点信息）
扩展按钮	扩展选中节点的1层邻居节点
邻居查找	以当前节点或节点群为起点，查找在结果集里的一层、二层、n层邻居
删除按钮	画布上删除当前节点，不会删除图数据库里的数据

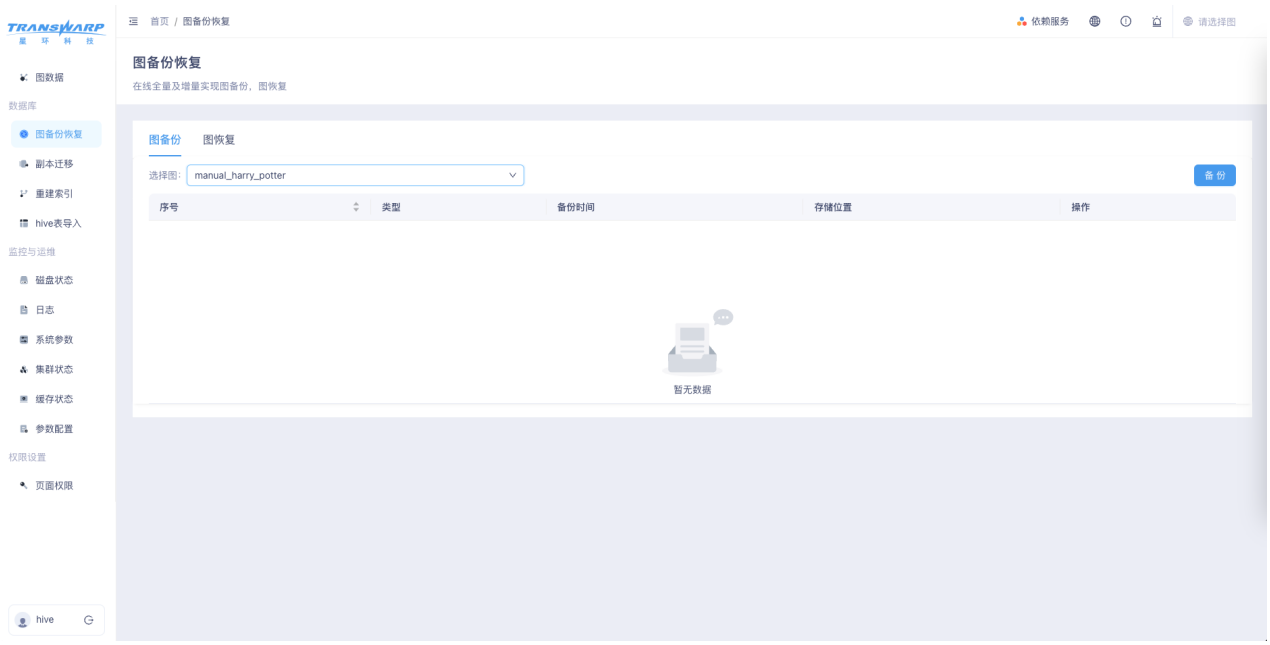
- Stellar3D操作指南界面介绍

操作指南：随时提供相机、风险等级、节点、节点簇、节点簇内部的操作说明

10.6.7. 图备份与恢复

在KG Explorer页面左侧菜单栏点击 **图备份与恢复** 进入图备份恢复页面。在该页面可以进行全量图备份、增量图备份以及备份图的恢复，图备份恢复内容详见《图备份与图恢复》章节。



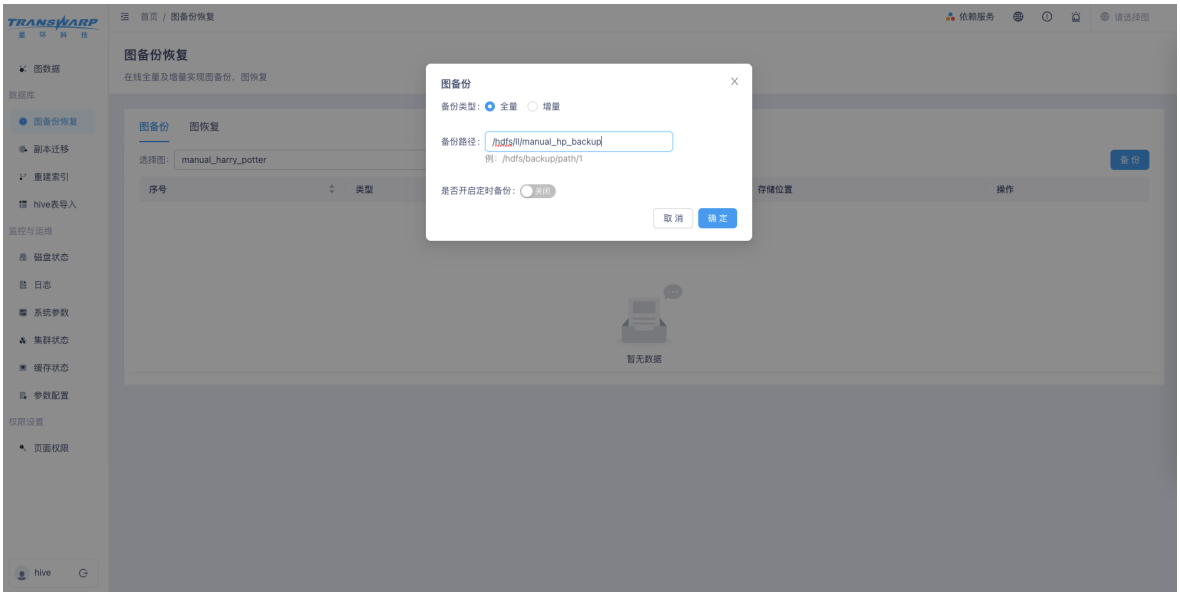


1. 图备份Tab页

   在图备份Tab页可以查看备份操作记录、以及对图进行备份操作。备份操作流程如下：

   a. 选择需要进行备份操作的图。

   b. 点击 **备份** 按钮进行参数填写。



   c. 选择备份类型、填写备份路径、设定好定时（可选）后，点击 **确定** 即可进行图备份。

d. 完成后可以在列表页面看到图的备份操作信息。

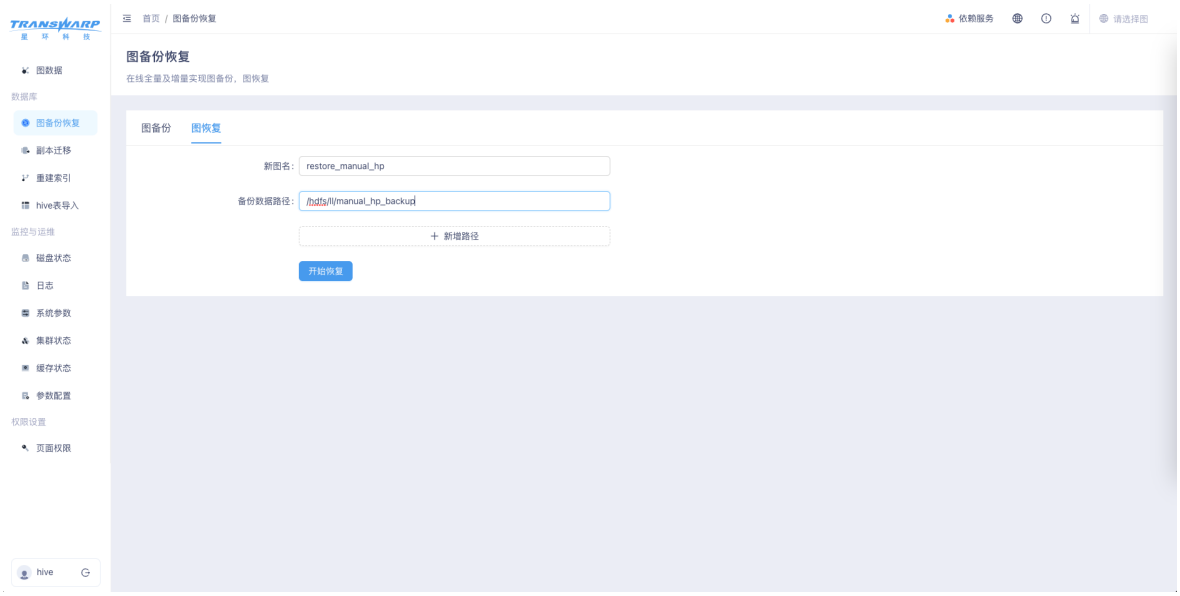
2. 图恢复Tab页

在图备份Tab页可以基于之前的备份数据对图进行恢复操作，恢复的图数据需要存到新图中。操作流程如下：

a. 填写恢复到的新图的图名。

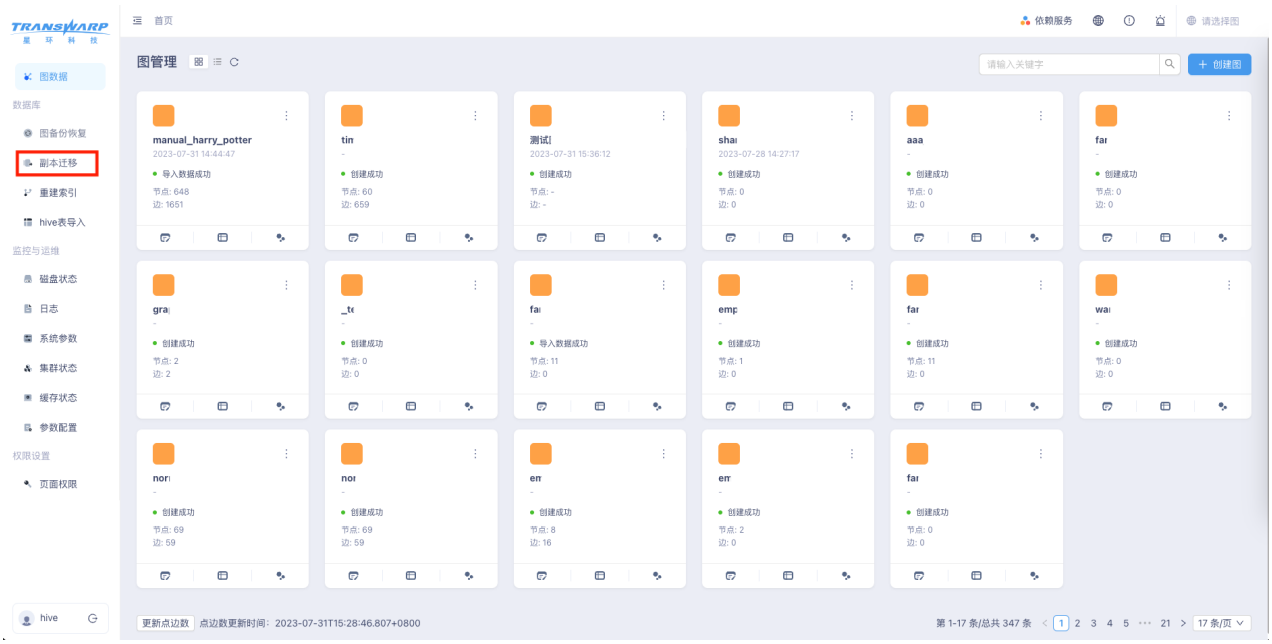
b. 填写之前已经备份的备份数据在HDFS上的存储路径（路径可以有多个）。

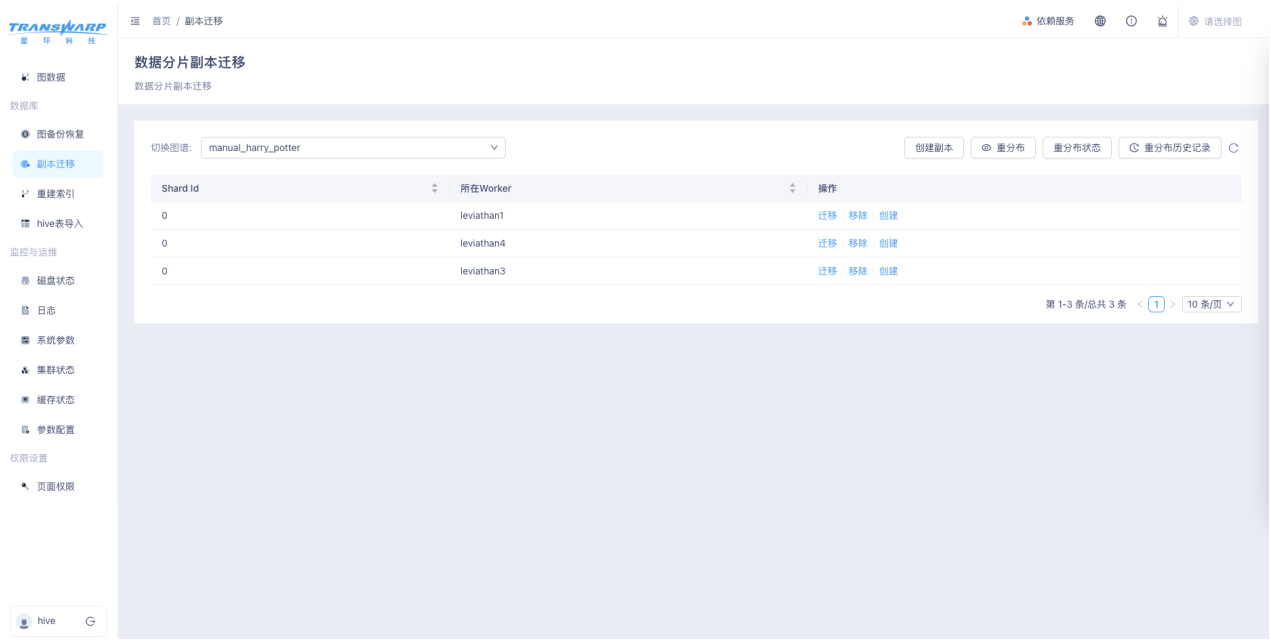
c. 点击 **开始恢复** 即可开始恢复操作。

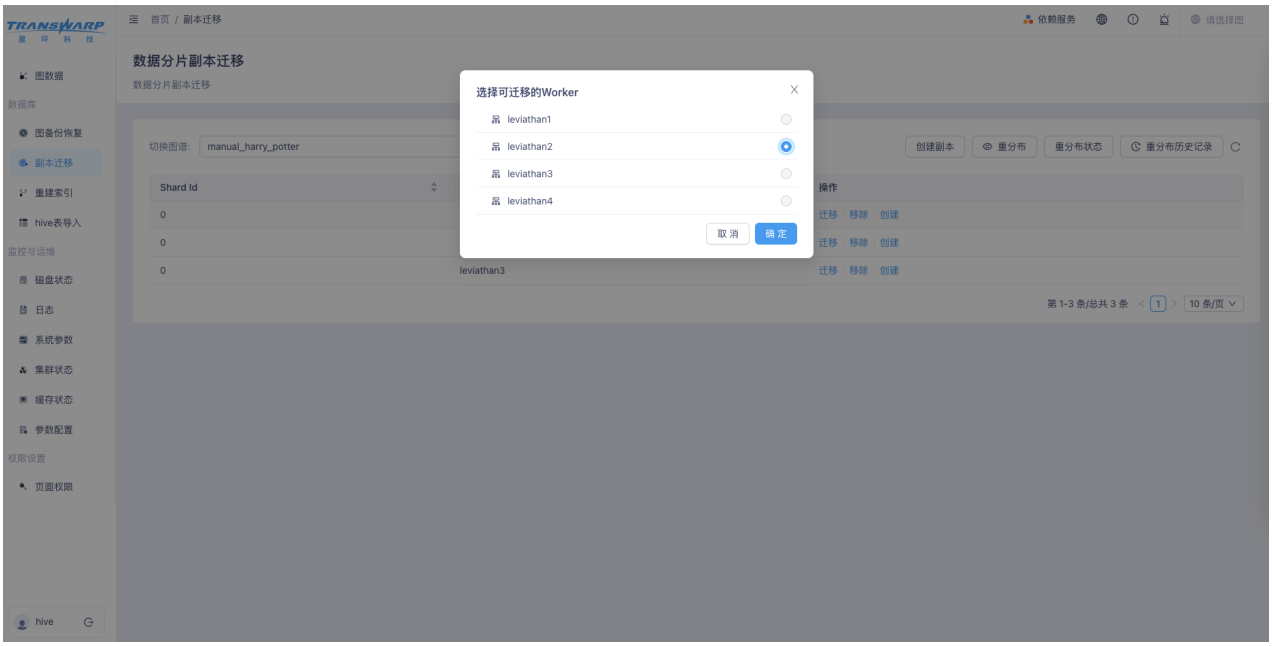


10.6.8. 副本迁移

在KG Explorer页面左侧菜单栏点击 **副本迁移** 进入数据分片副本迁移页面。可以使用迁移功能重新分配数据分片的存储节点。选择图后点击数据副本后的迁移按钮，可对该数据副本进行迁移。


TRANSWARP
首页
依赖服务
请选择图
图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
hive
图管理
请输入关键字
创建图
manual_harry_potter
2023-07-31 14:44:47
导入数据成功
节点: 648
边: 1651
tin
创建成功
节点: 60
边: 659
测试
2023-07-31 15:36:12
创建成功
节点: -
边: -
aaa
创建成功
节点: 0
边: 0
创建成功
节点: 2
边: 2
导入数据成功
节点: 11
边: 0
节点: 1
节点: 69
边: 59
节点: 8
边: 16
节点: 2
更新点边数
点边数更新时间: 2023-07-31T15:28:46.807+0800
第 1-17 条/总共 347 条
17 条/页

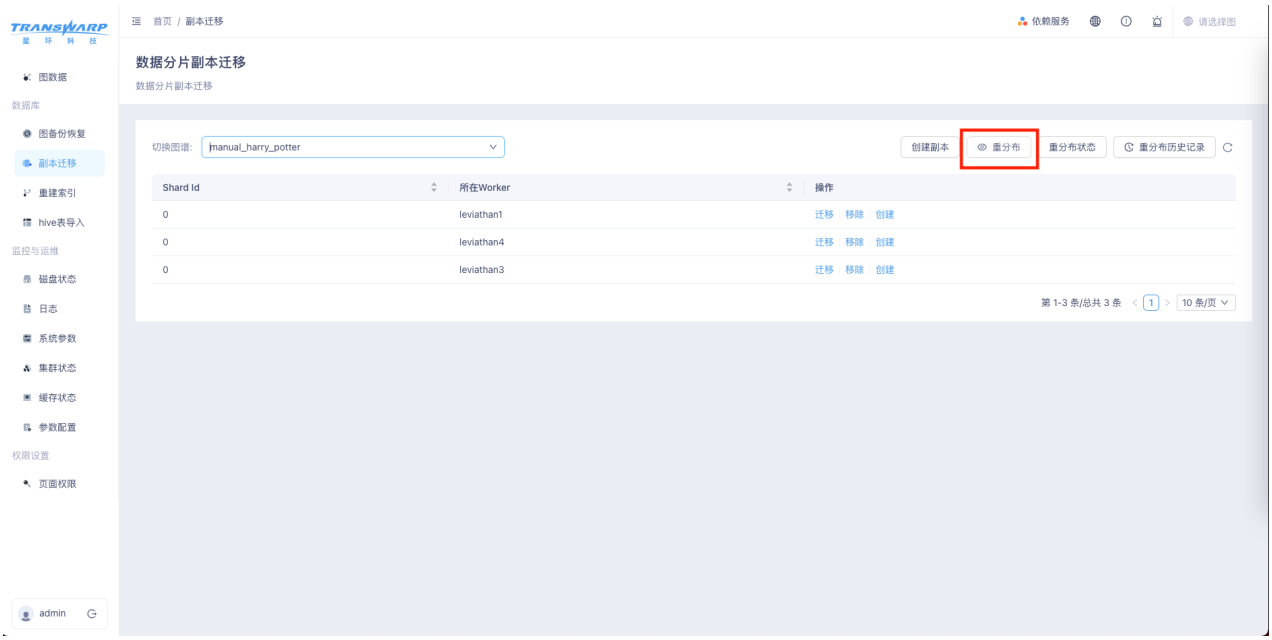

TRANSWARP
首页 / 副本迁移
依赖服务
请选择图
图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
hive
数据分片副本迁移
数据分片副本迁移
切换图谱:
manual_harry_potter
创建副本
重分布
重分布状态
重分布历史记录
Shard Id
所在Worker
操作
0
leviathan1
迁移 移除 创建
leviathan4
leviathan3
第 1-3 条/总共 3 条
10 条/页



10.6.9. 副本重分布

当前版本新增副本重分布功能，主要使用场景为：当集群增加节点后，将集群中已有的数据副本均匀地分配到集群中所有的节点上。

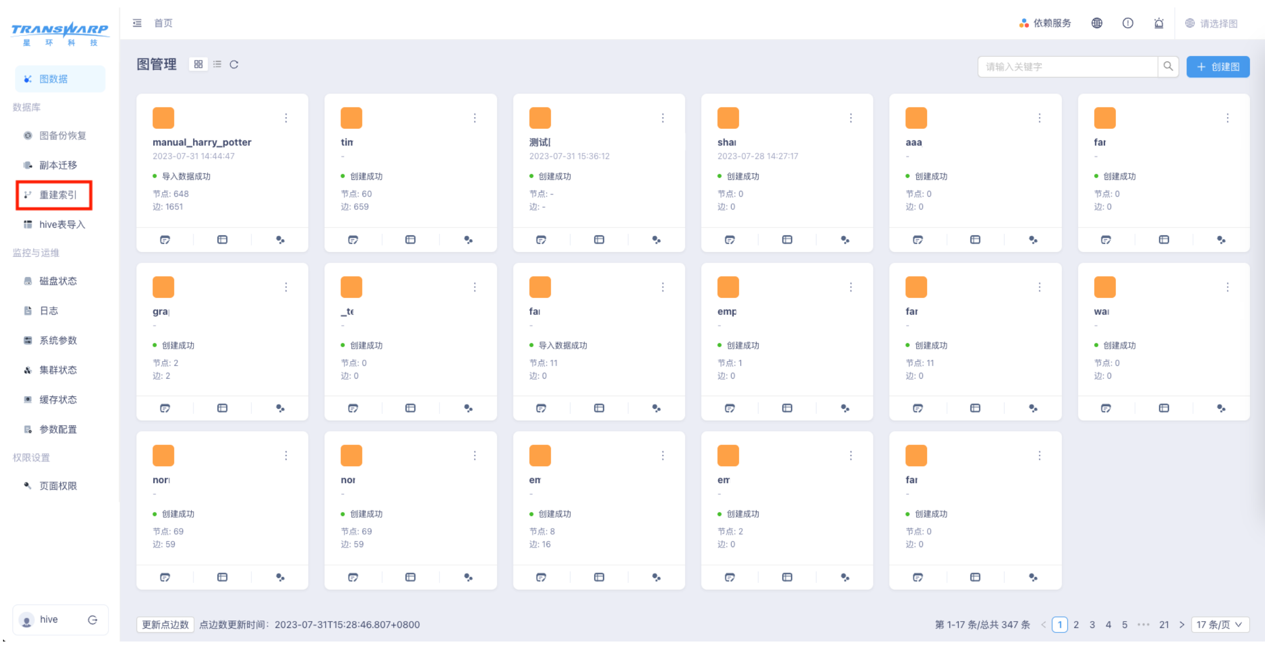
在副本迁移页面，可以点击"重分布“进行副本均衡操作。

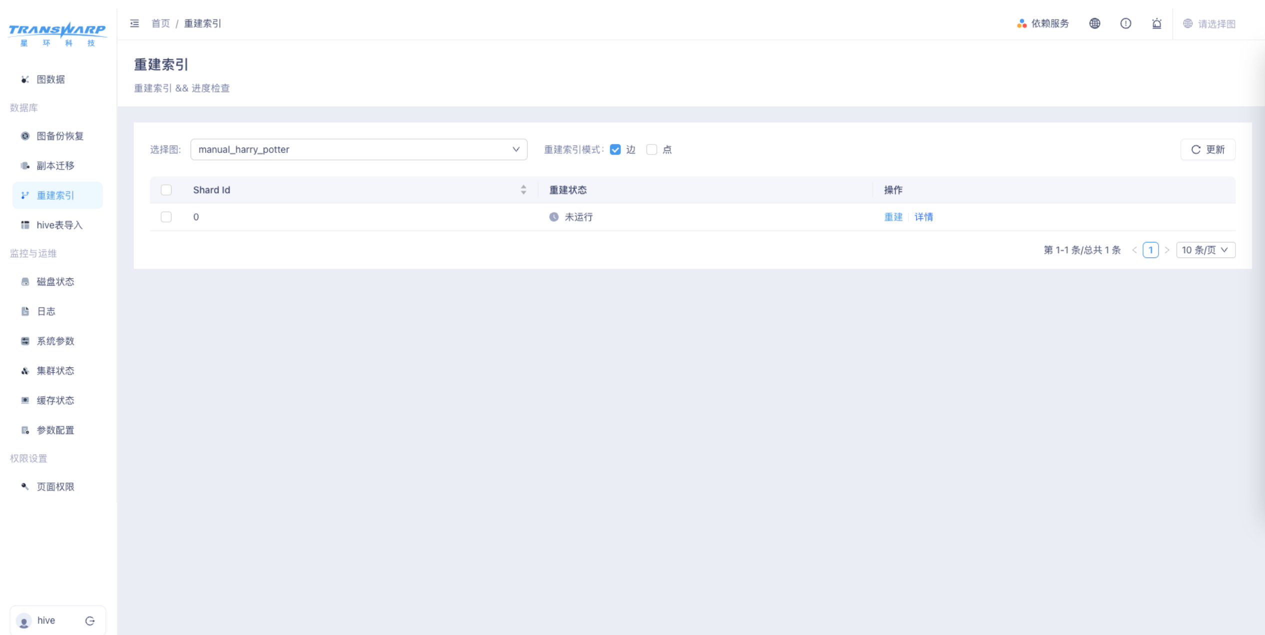


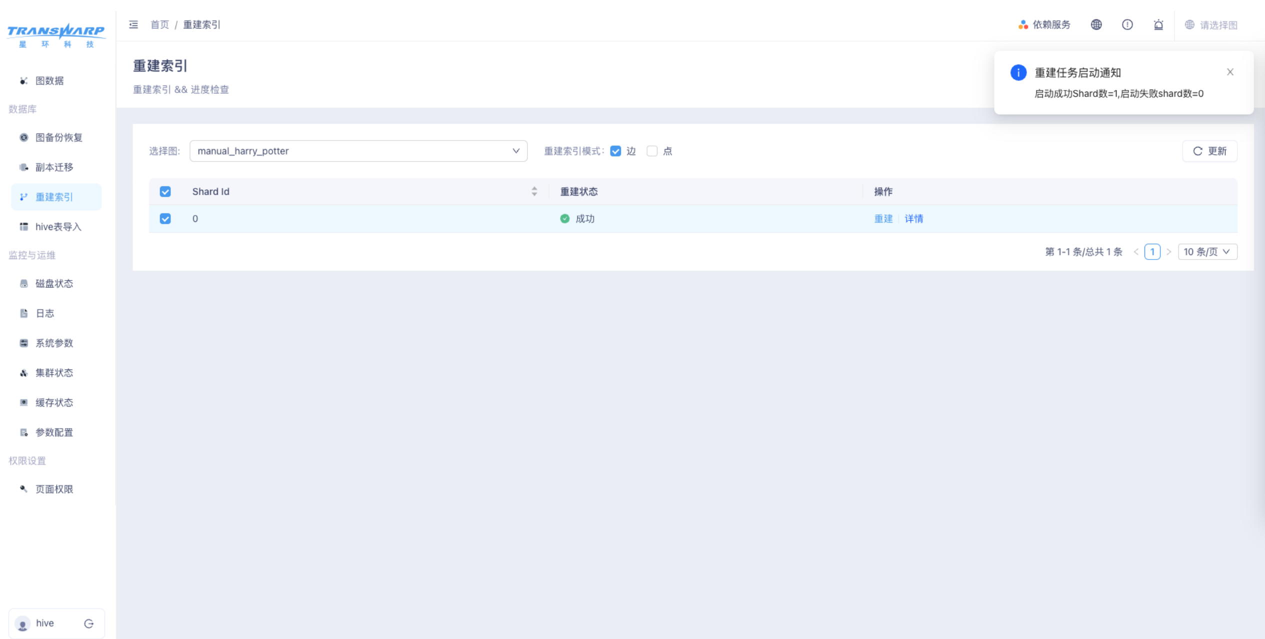
可通过"重分布状态“查看当前集群副本均衡状态，完成的会高亮显示。

10.6.10. 索引重建

在KG Explorer页面左侧菜单栏点击 **索引重建** 进入索引重建页面。在该页面选择图谱、选择点边类型后，可以点击 **重建** 按钮为对应的数据分片重新创建索引。重建索引详见《索引重建》章节。


首页
依赖服务
请选择图
图管理
请输入关键字
+ 创建图
图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
manual_harry_potter
2023-07-31 14:44:47
导入数据成功
节点: 648
边: 1651
创建成功
hive
更新点边数
点边数更新时间: 2023-07-31T15:28:46.807+0800
第 1-17 条/总共 347 条
17 条/页

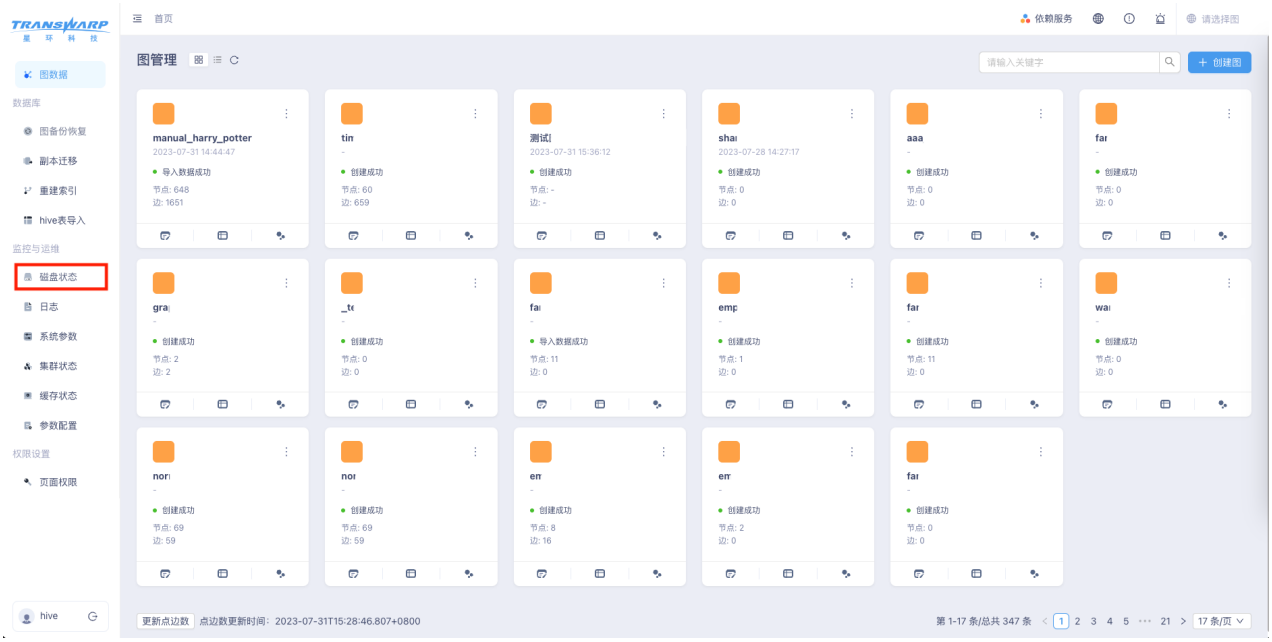

首页 / 重建索引
重建索引
重建索引 && 进度检查
选择图:
manual_harry_potter
重建索引模式:
边
点
更新
Shard Id
重建状态
操作
0
未运行
重建
详情
第 1-1 条/总共 1 条
10 条/页

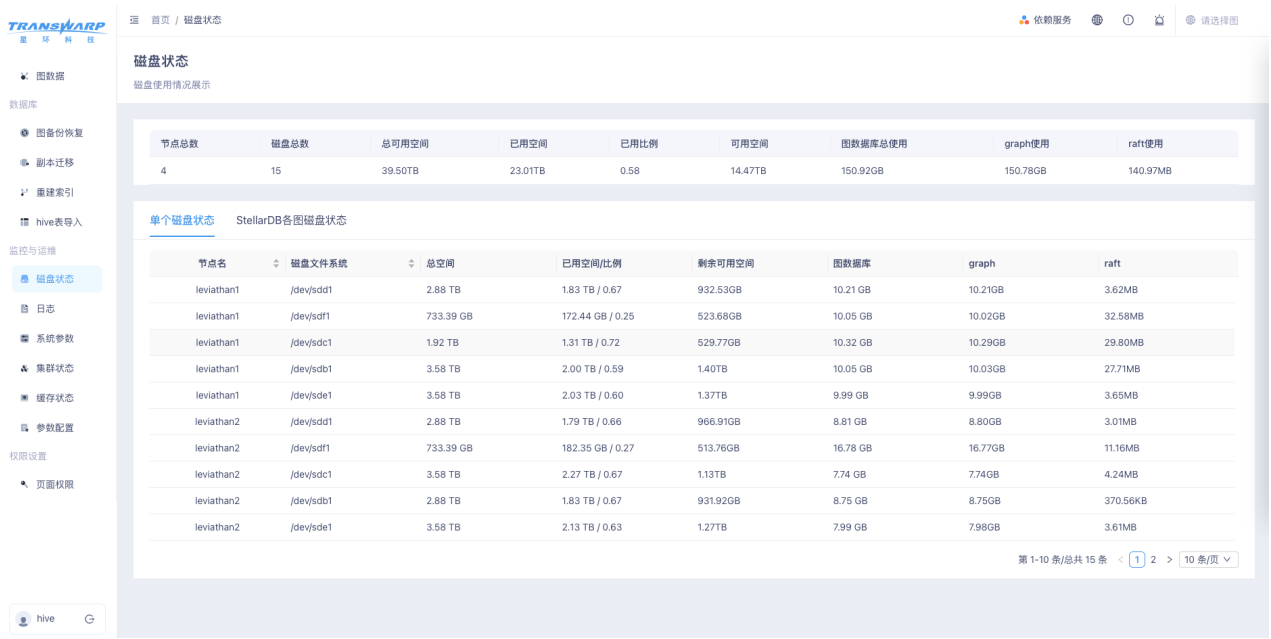

首页 / 重建索引
重建索引
重建索引 && 进度检查
重建任务启动通知
启动成功Shard数=1,启动失败shard数=0
选择图:
manual_harry_potter
重建索引模式:
边
点
更新
Shard Id
重建状态
操作
0
成功
重建
详情
第 1-1 条/总共 1 条
10 条/页

10.6.11. 监控与运维

10.6.11.1. 磁盘状态监控

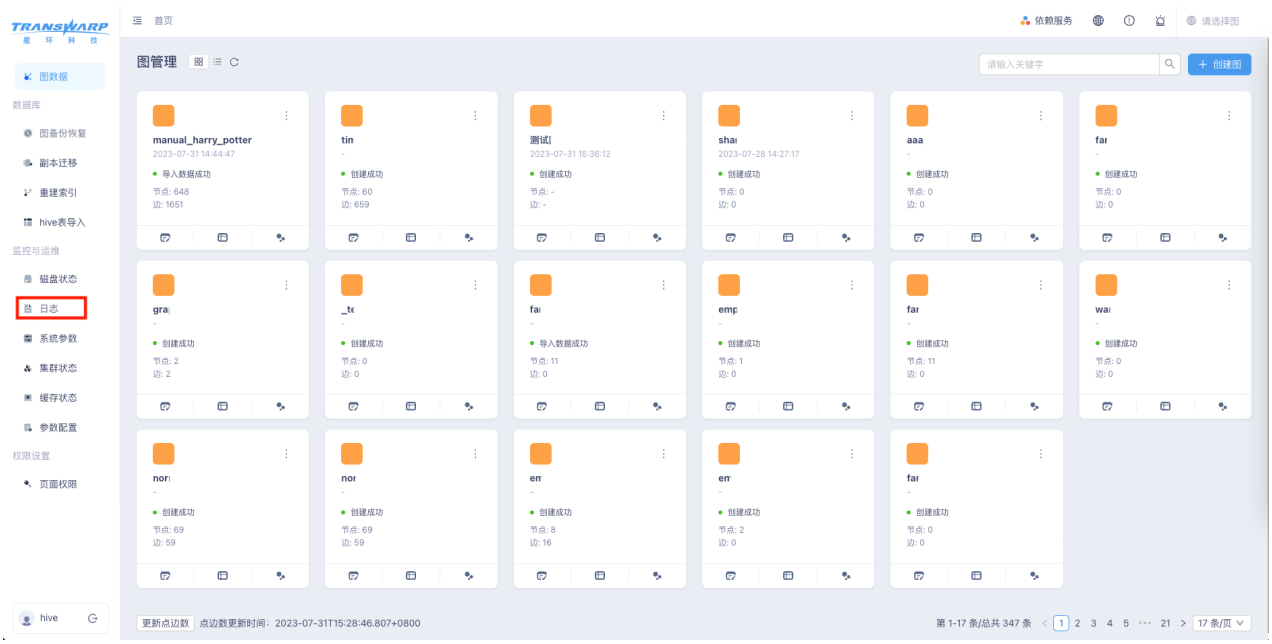
在KG Explorer页面左侧菜单栏点击 **监控与运维** → **磁盘状态** 进入磁盘状态监控页面。在该页面可以查看当前StellarDB集群的磁盘使用状态，包括以磁盘为单位和以图为单位的统计信息。

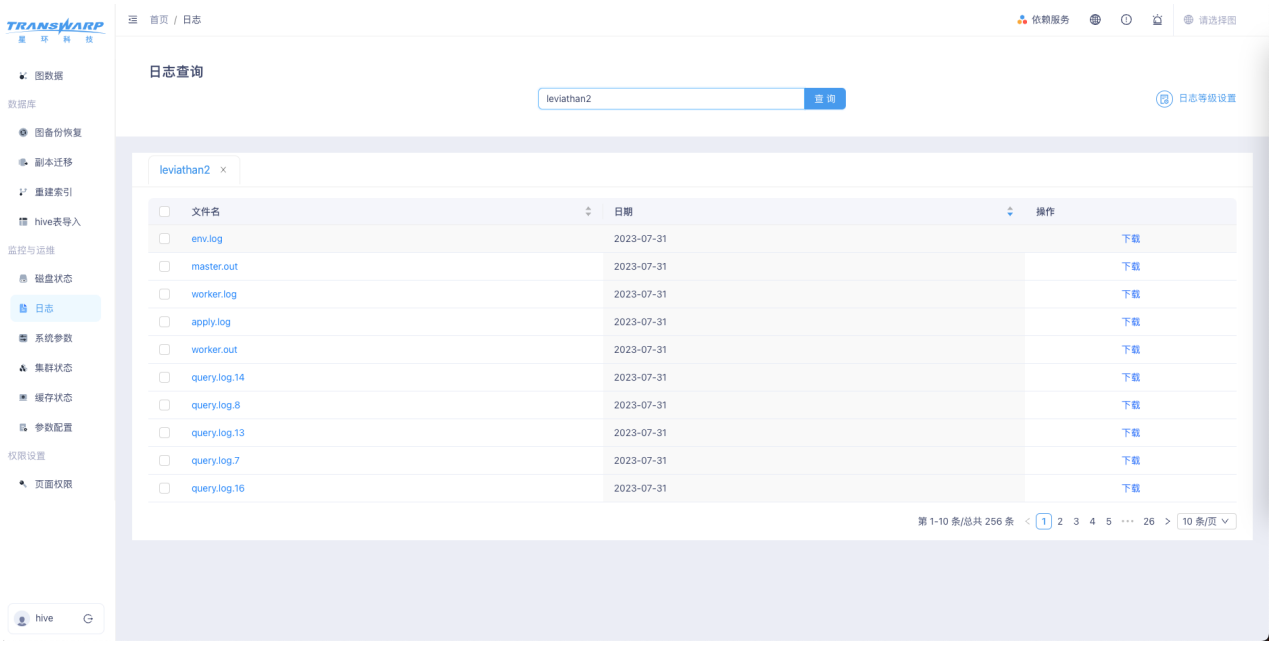


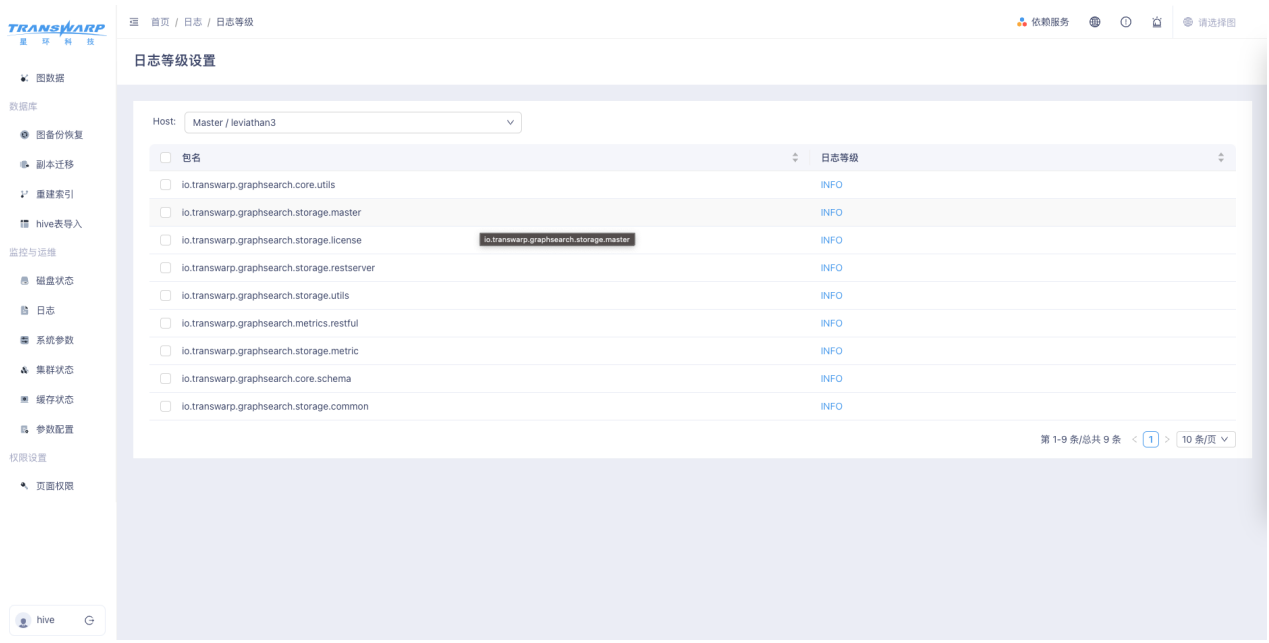


10.6.11.2. 日志查看

在KG Explorer页面左侧菜单栏点击 **监控与运维** → **日志** 进入日志查看页面。在该页面可查看和下载各个节点的日志。点击左右角设置日志等级按钮可以设置输出日志的等级。


图管理
创建图
manual_harry_potter
tin
测试
aaa
更新点边数
点边数更新时间：2023-07-31T15:28:46.807+0800

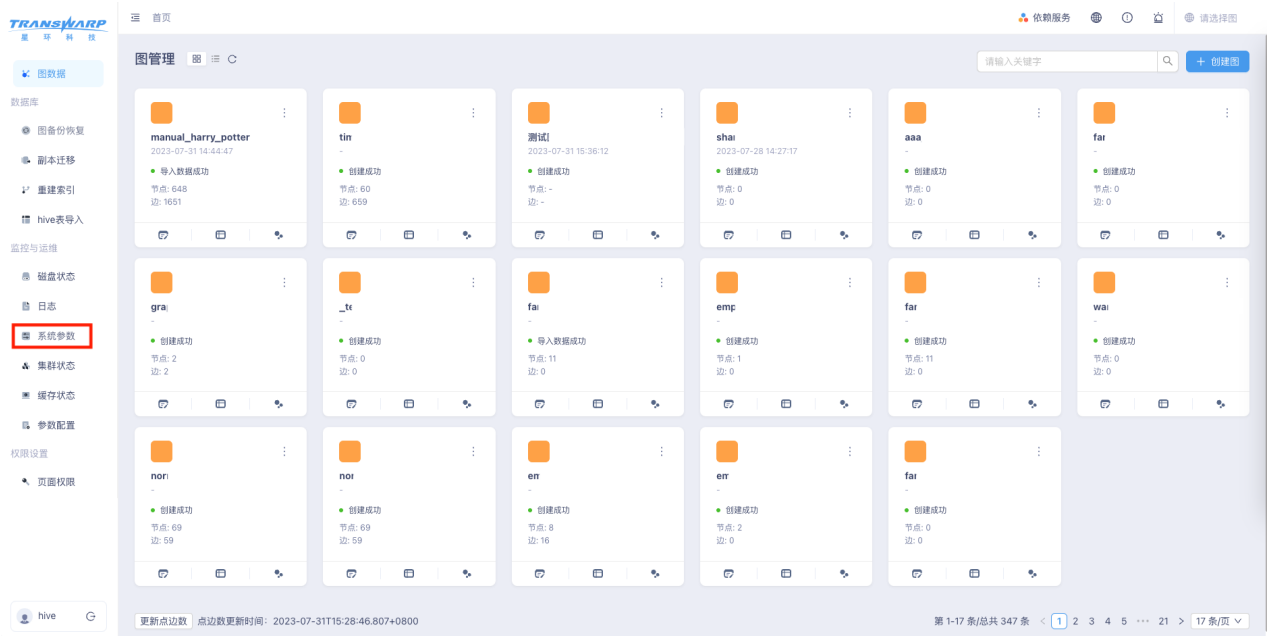

日志查询
leviathan2
查 询
日志等级设置
文件名
日期
操作
env.log
master.out
worker.log
apply.log
worker.out
query.log.14
query.log.8
query.log.13
query.log.7
query.log.16
2023-07-31
下载

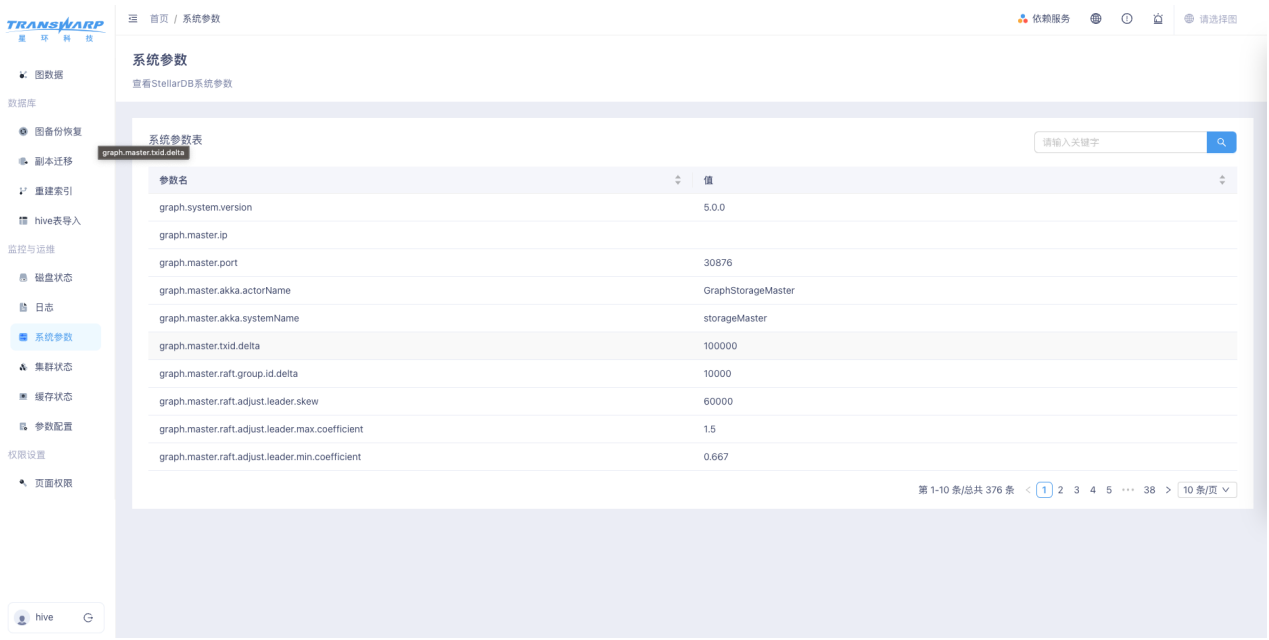


KG Explorer的日志下载接口会被Firefox的AdBlocker Ultimate插件当成广告拦截，导致功能不可用。如遇AdBlocker拦截，推荐关闭AdBlocker后下载。

10.6.11.3. 系统参数查看

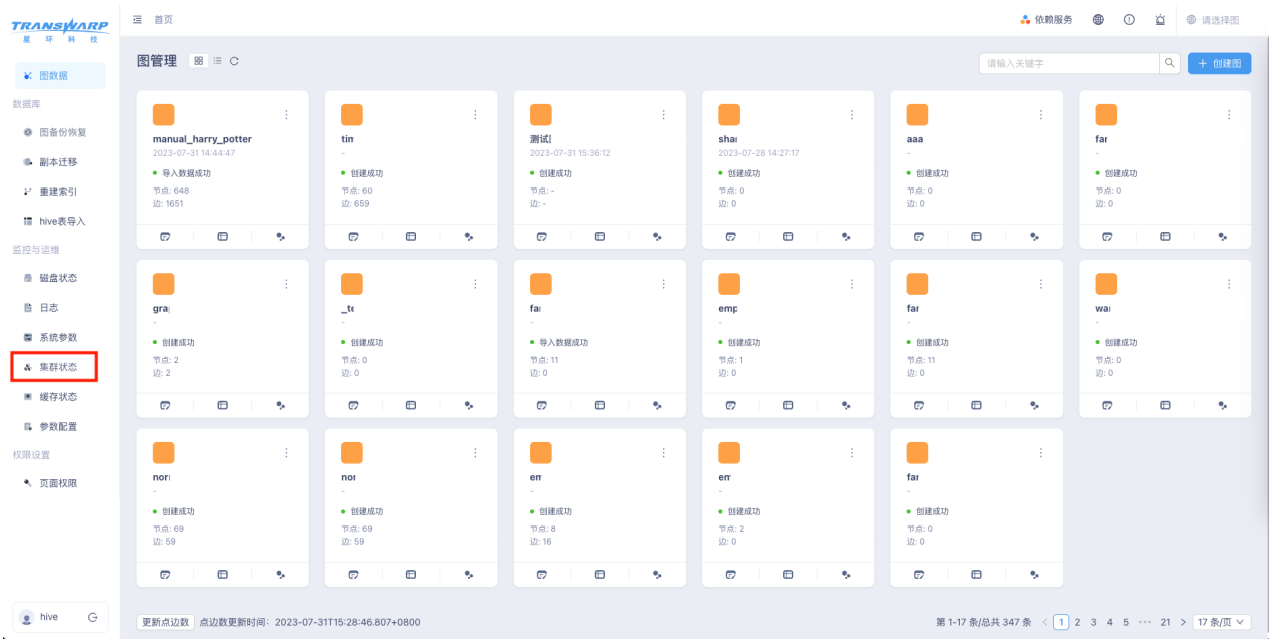
在KG Explorer页面左侧菜单栏点击 **监控与运维** → **系统参数** 进入系统参数查看页面。可以查看当前StellarDB集群的参数名称和参数值。

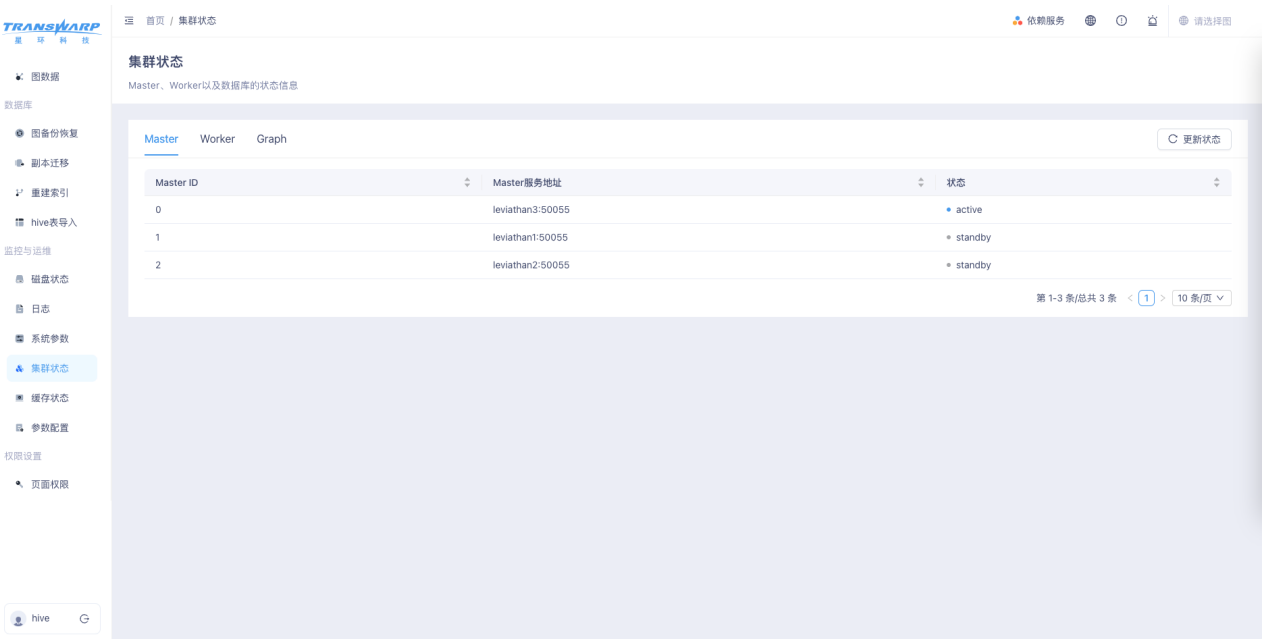




10.6.11.4. 集群状态查看

在KG Explorer页面左侧菜单栏点击 **监控与运维** → **集群状态** 进入集群状态查看页面。在该页面可以查看StellarDB集群的各个Master、Worker角色的状态信息以及数据库中各个图的状态信息。

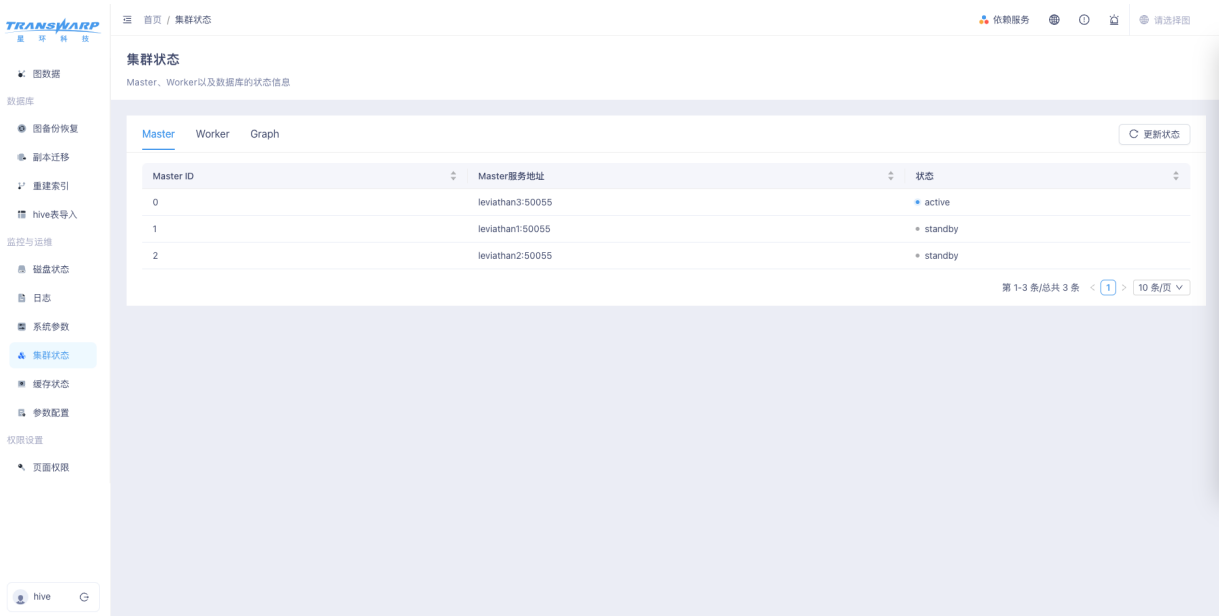




该页面包含三个Tab页，内容是 **定时** 从图数据库获取的，点击右上角的更新按钮可以更新列表信息。各个Tab页的内容说明如下：

1. Master Tab页

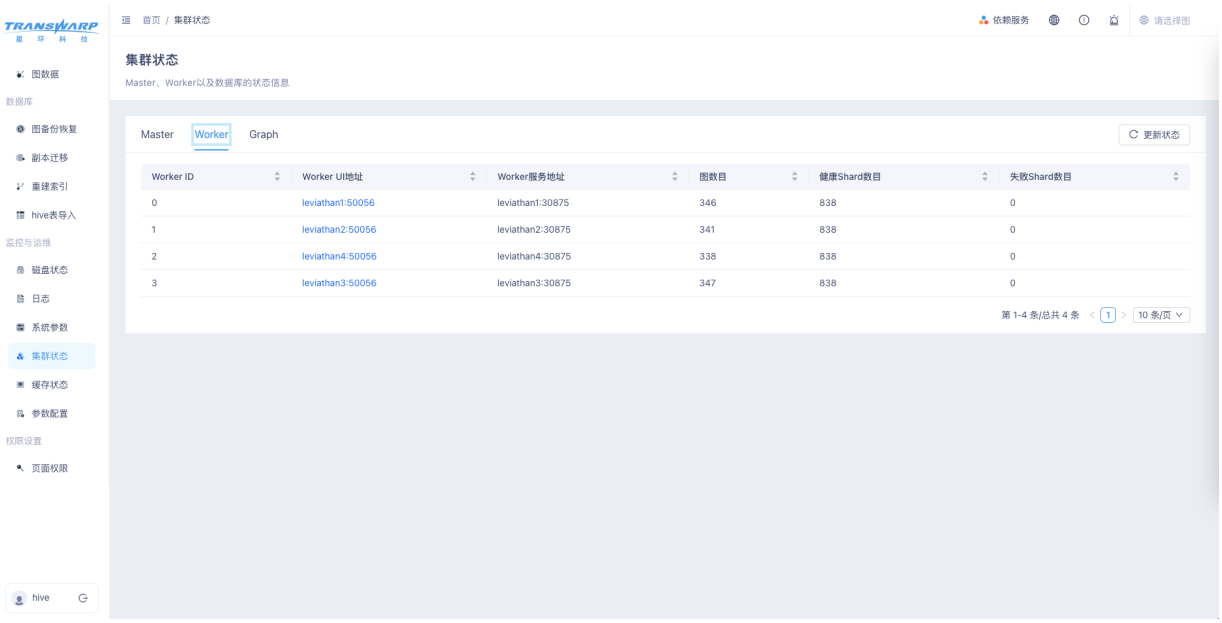
   该页面可以查看图数据库集群的各个Master角色的基本信息，包括：角色ID、服务地址、角色状态。



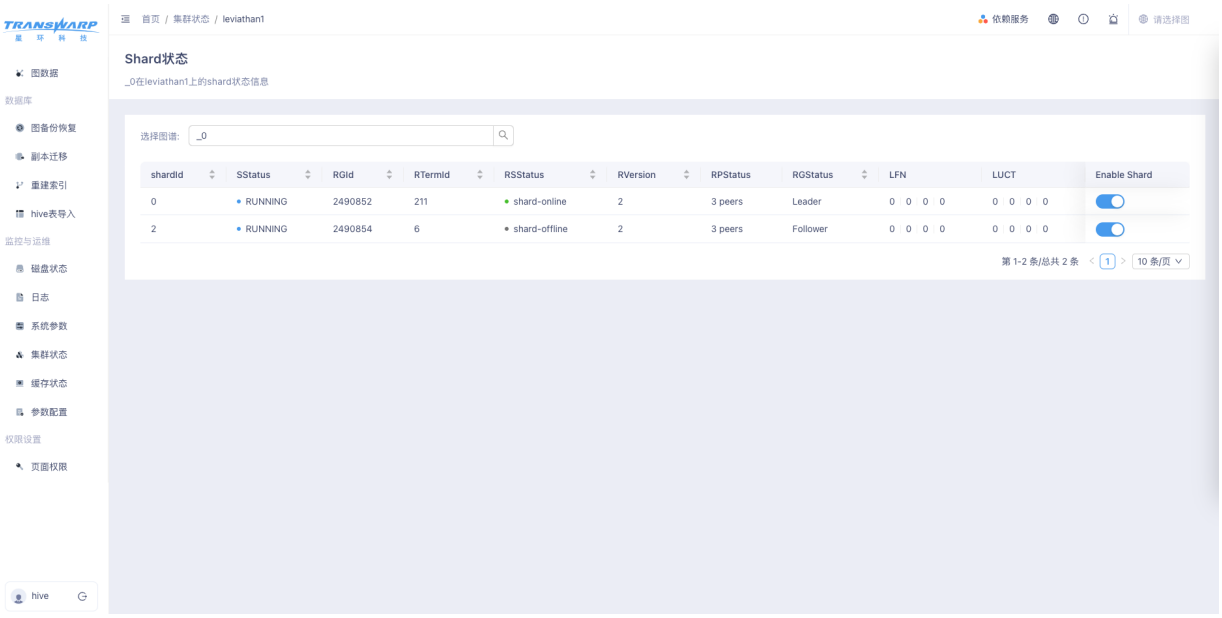
Master在StellarDB中承担了调度和管理功能，StellarDB集群可以启动多个Master角色组成Master HA Group，由选举出的Active Master提供集群服务（详情请查阅《StellarDB系统角色与配置》）。

2. Worker Tab页

   该页面可以查看各个Worker角色的基本信息，包括Worker角色id、UI页面地址、服务地址、当前Worker上图的数目、当前Worker上健康的shard（数据分区）数目、当前Worker上有问题的shard数目



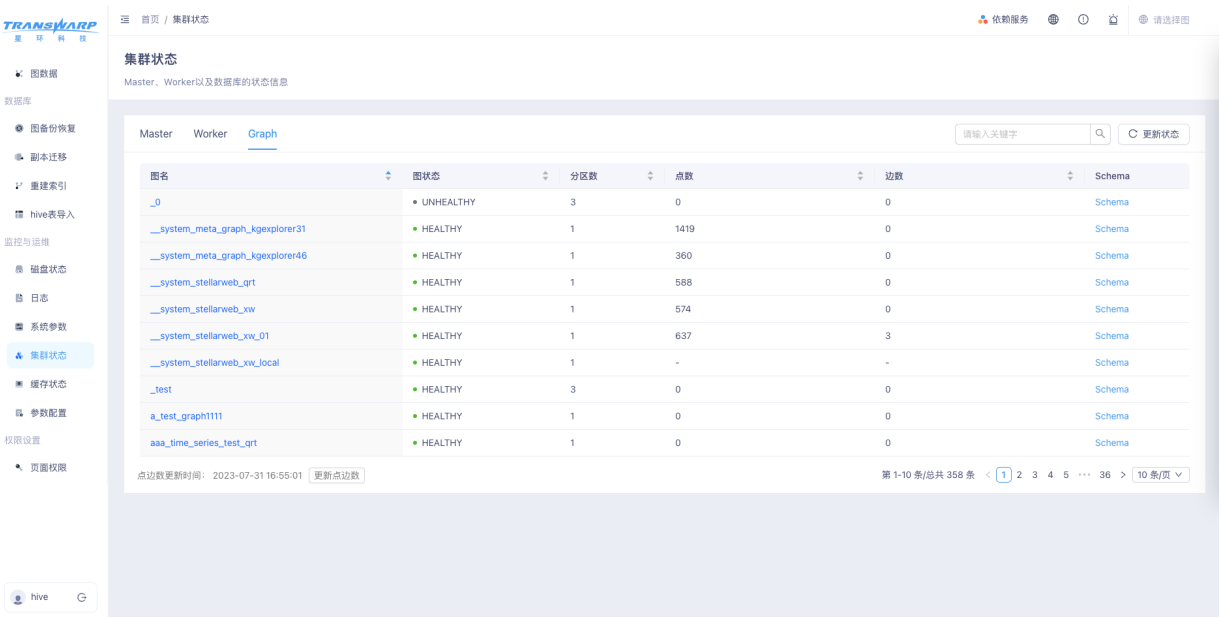
点击Worker页面中相应的UI页面地址，可以查看对应Worker上各个图数据分区（Shard）的详细信息，可以通过下拉框选择图名查看对应图的Shard信息：



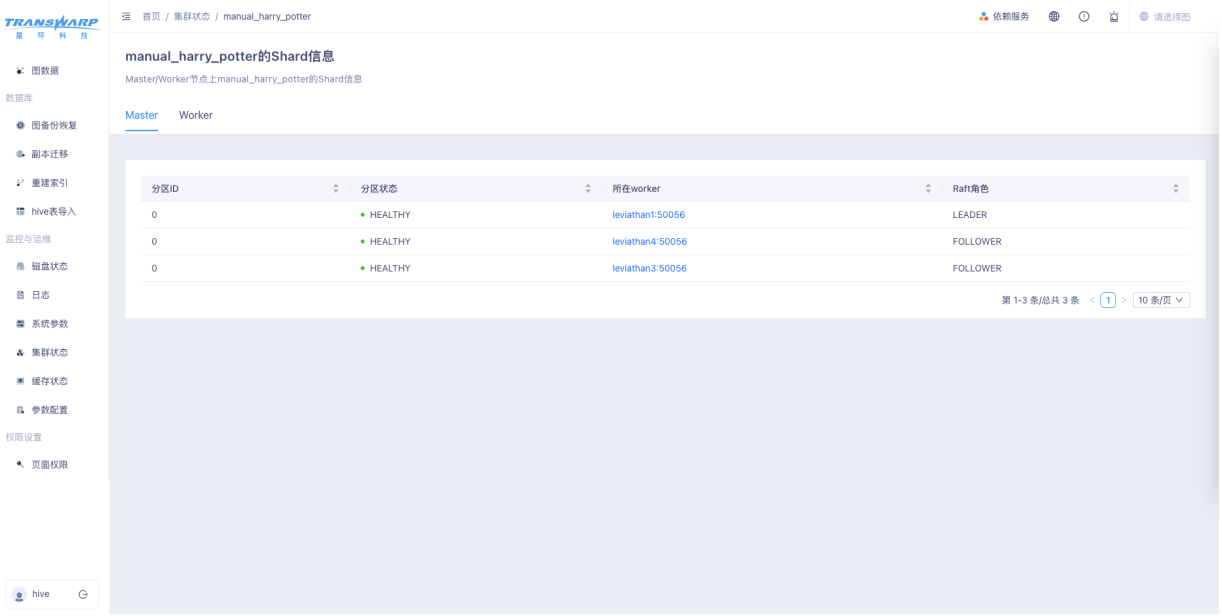
Worker在StellarDB中承担了数据存储、数据查询的功能，StellarDB集群可以启动多个Worker提供服务（详情请查看《StellarDB系统角色与配置》）。

3. Graph Tab页

可以查看StellarDB上各个图的基本状态，包括图名、图状态、分区数、点边数目（估计值，并非精确点边数目，仅供参考）、图schema信息。点击下方的 **更新点边数** 按钮可以更新各个图的点边数目（图数量多时更新时间较长）。

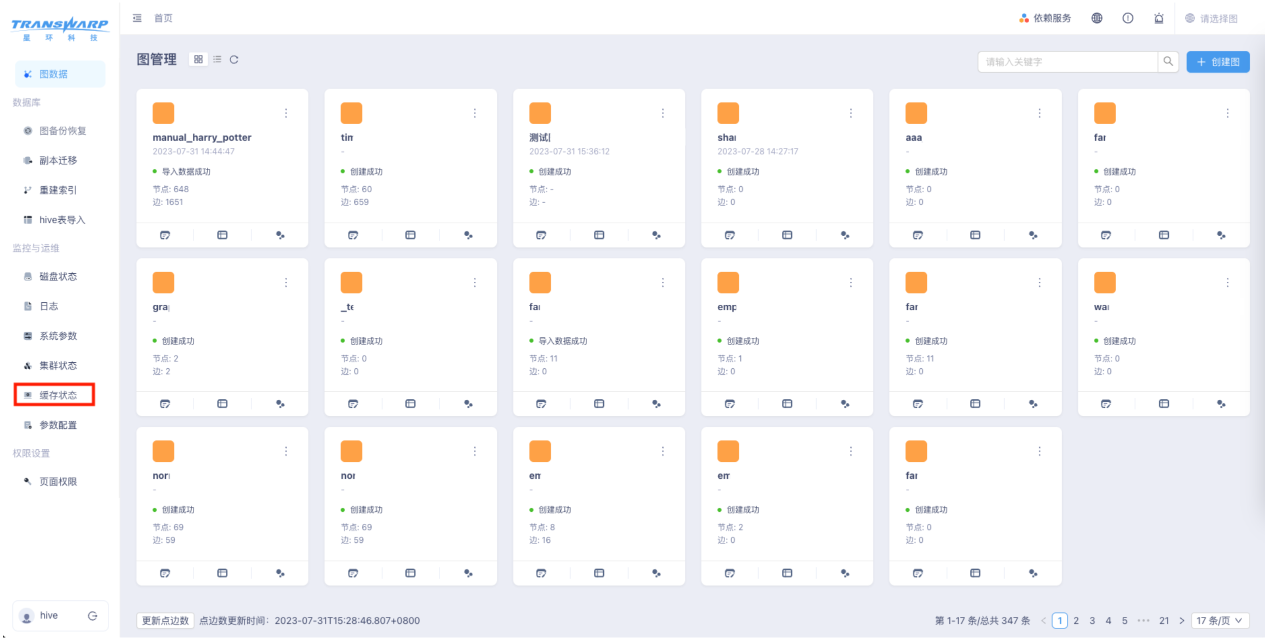


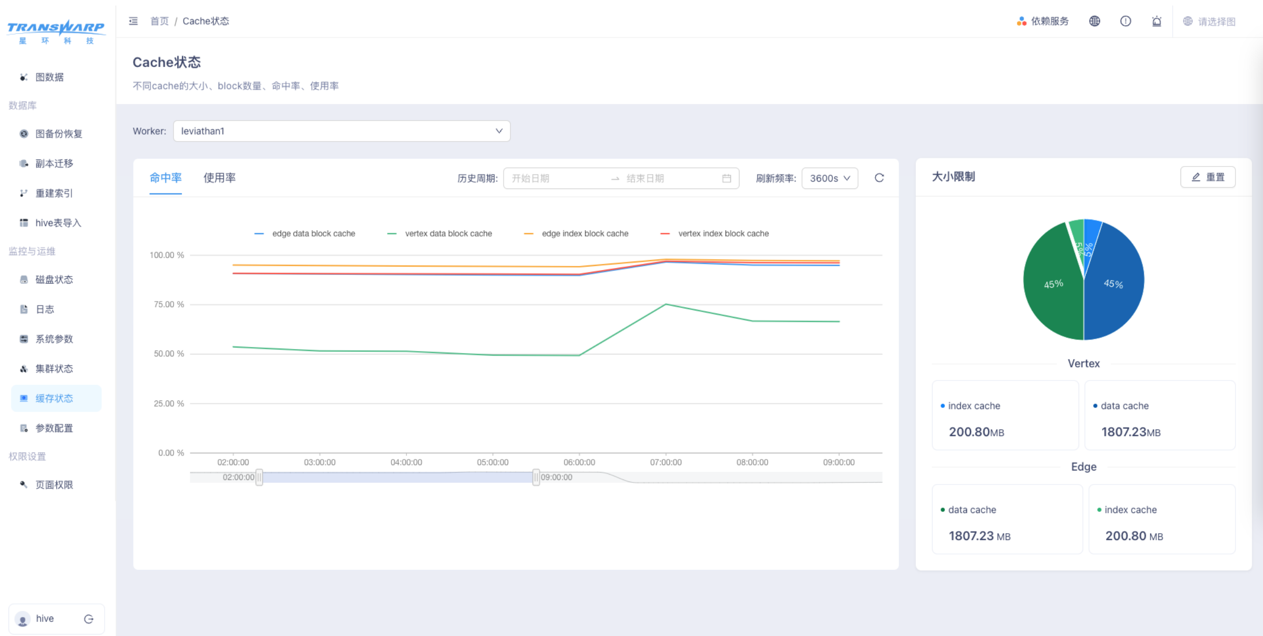
点击图名，可以查看该图各个数据分区（shard）信息：

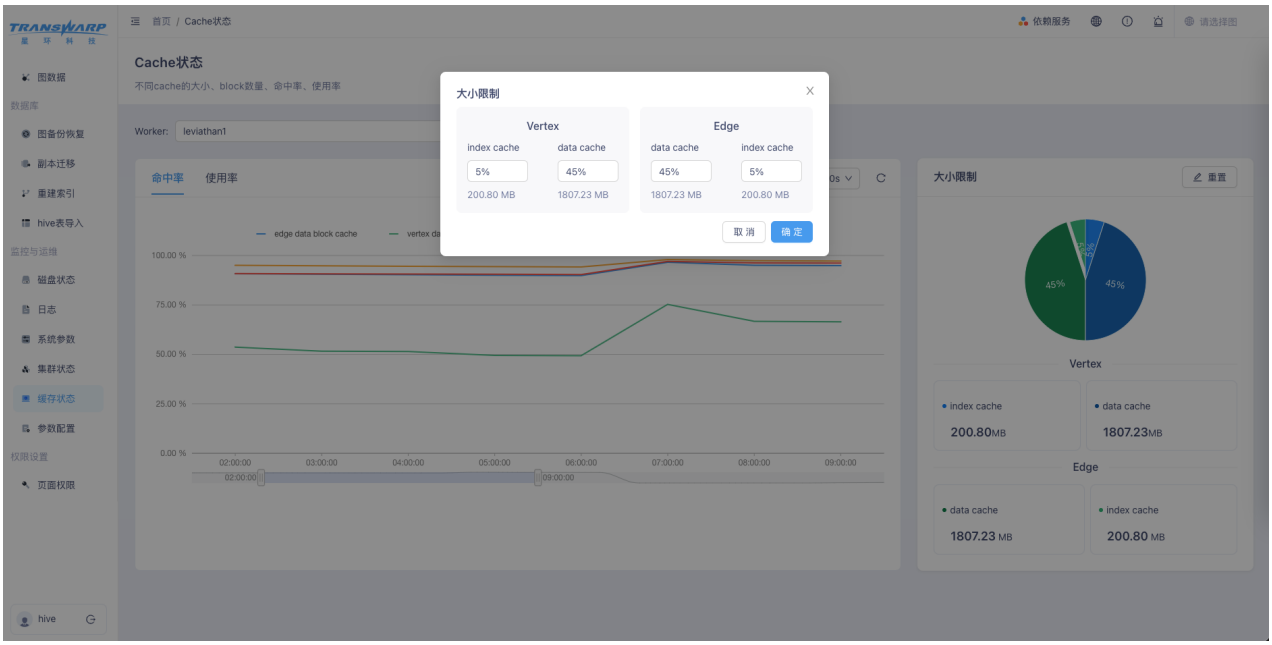


10.6.11.5. 缓存状态查看

在KG Explorer页面左侧菜单栏点击 **监控与运维 → 缓存状态** 进入缓存状态查看页面。在该页面可以查看不同集群上数据block缓存的命中率和使用率，并且可以对缓存的大小进行设置。

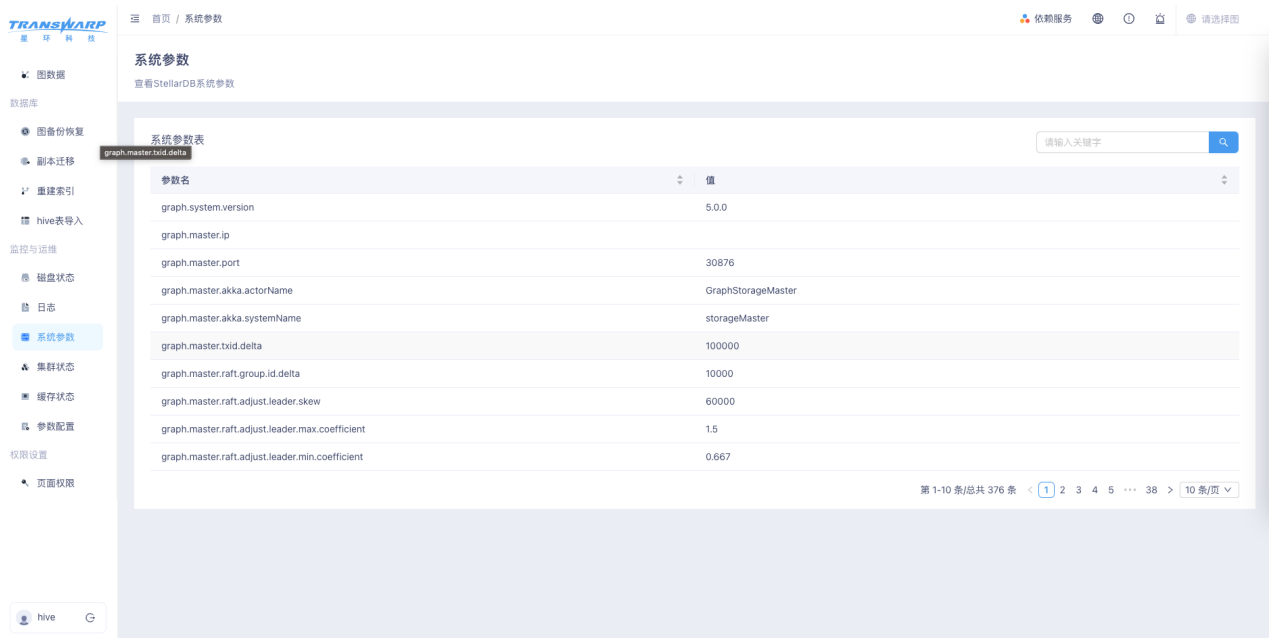

图数据
数据库
图备份恢复
副本迁移
重建索引
hive表导入
监控与运维
磁盘状态
日志
系统参数
集群状态
缓存状态
参数配置
权限设置
页面权限
首页
依赖服务
请选择图
图管理
请输入关键字
创建图
manual_harry_potter
2023-07-31 14:44:47
导入数据成功
节点: 648
边: 1651
创建成功
更新点边数
点边数更新时间: 2023-07-31T15:28:46.807+0800
第 1-17 条/总共 347 条
17 条/页


首页 / Cache状态
Cache状态
不同cache的大小、block数量、命中率、使用率
Worker: leviathan1
命中率
使用率
历史周期:
开始日期
结束日期
刷新频率:
3600s
edge data block cache
vertex data block cache
edge index block cache
vertex index block cache
大小限制
重置
Vertex
index cache
200.80MB
data cache
1807.23MB
Edge
data cache
1807.23 MB
index cache
200.80 MB



10.6.11.6. 图数据库参数配置

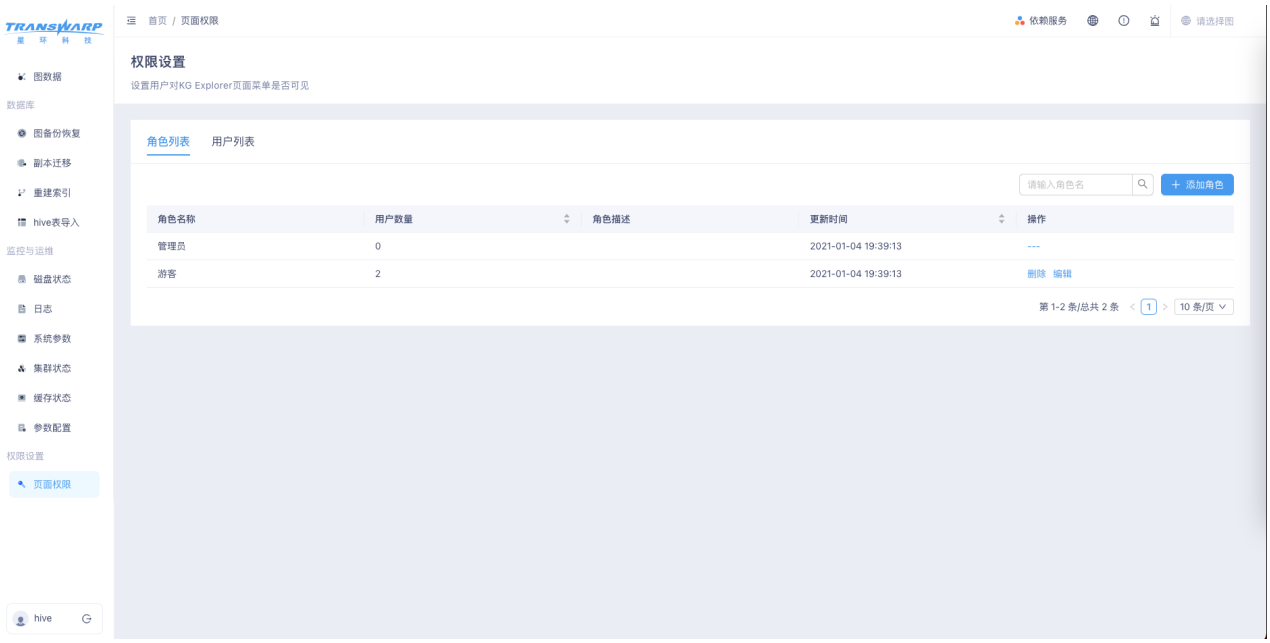
在KG Explorer页面左侧菜单栏点击 **监控与运维** → **参数配置** 进入图数据库参数配置页面。在该页面中可以在此设置StellarDB可调的相关参数。



10.6.12. 页面权限设置

开启KG Explorer页面权限功能后，左侧菜单栏会增加 **页面权限设置** 子页面，该页面可设置用户角色以及管理用户对KG Explorer各子页面的访问权限。建议该功能和用户认证功能同时开启或关闭。

需要注意的是，该页面只有 在Guardian上拥有依赖Quark服务的Global Admin权限的用户 以及 在KG Explorer配置了该页面访问权限的用户 可以访问，这些用户可以在左侧菜单栏点击 **页面权限** 进入页面权限设置子页面。

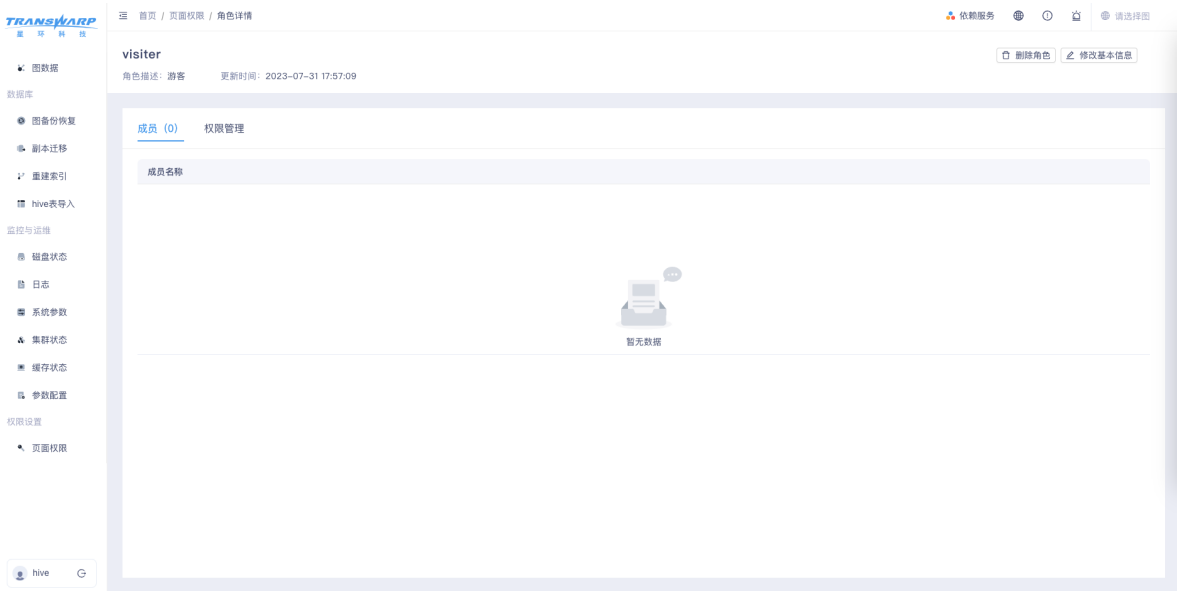


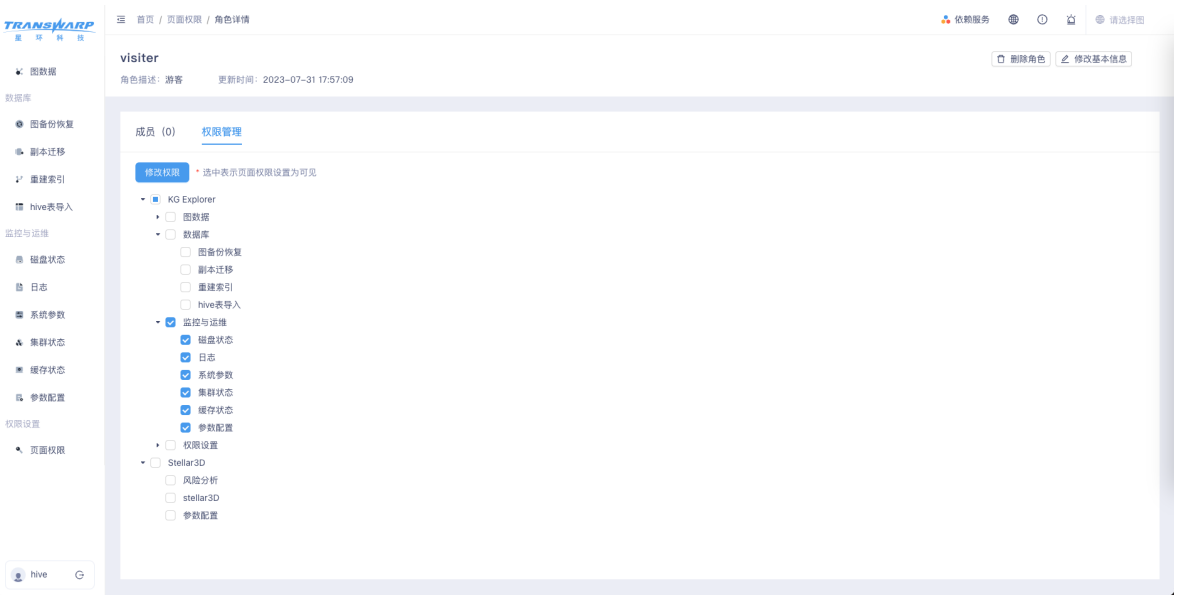
页面权限设置功能说明：

1. 用户可以在角色列表子页面查看用户角色列表、添加新角色、编辑角色信息以及删除角色。

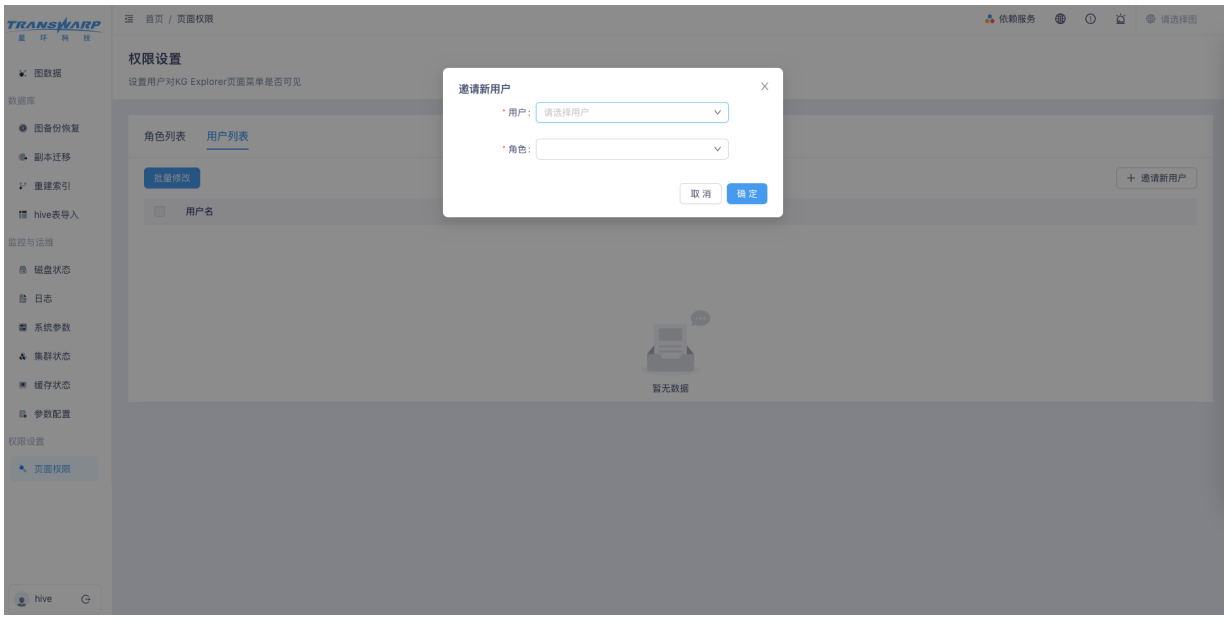
   a. 点击 **添加角色** 按钮可以创建新角色，创建成功后点击 **编辑** 按钮可以为该角色添加页面访问权限。

b. 点击 **编辑** 之后，可以查看该角色的成员列表、编辑角色基本信息和修改该用户角色对KG Explorer各个页面的访问权限。





2. 用户可以在用户列表子页面修改用户的角色信息以及邀请新用户（Guardian中存在的用户）。



11. StellarDB认证与权限控制

11.1. StellarDB安全认证

StellarDB支持的安全认证方式主要有三种：**KERBEROS**、**LDAP**、**SIMPLE**，可以针对不同的使用场景进行配置。

11.1.1. 安全认证方式说明

StellarDB切换不同的认证方式主有两方面的配置共同决定：

1. 集群整体是否开启Kerberos安全；
2. 对应Quark/QuarkGateway服务的参数 `hive.server2.authentication` 的值

- 集群所有组件（StellarDB、Quark、KG Explorer以及基础组件），要么都开启Kerberos安全，要么都不开启Kerberos安全。
- 若是配置了QuarkGateway的环境，请确保QuarkGateway以及其连接的所有Quark服务的 `hive.server2.authentication` 参数值相同。

不同认证方式对应的配置情况如下：

认证方式	说明	开启方式	备注
KERBEROS	Kerberos认证模式	集群所有组件开启Kerberos安全，Quark/QuarkGateway服务参数 `hive.server2.authentication` 设置为 **NONE/KERBEROS**	可以在 Guardian 组件页面一键开启或关闭集群所有组件的Kerberos安全
LDAP	LDAP认证模式	Quark/QuarkGateway服务参数 `hive.server2.authentication` 设置为 **LDAP**，对于集群整体是否开启Kerberos安全没有要求	
SIMPLE	不进行用户认证	集群整体关闭Kerberos安全（可以在Guardian组件页面一键关闭），Quark/QuarkGateway服务参数 `hive.server2.authentication` 设置为 **NONE**	可以在Guardian组件页面一键开启或关闭集群所有组件的Kerberos安全

- 关于安全认证的详细介绍请参考 Transwarp Guardian手册。
- 关于在不同安全认证模式下访问Quark/QuarkGateway服务的方式请参考《StellarDB的访问方式》章节。

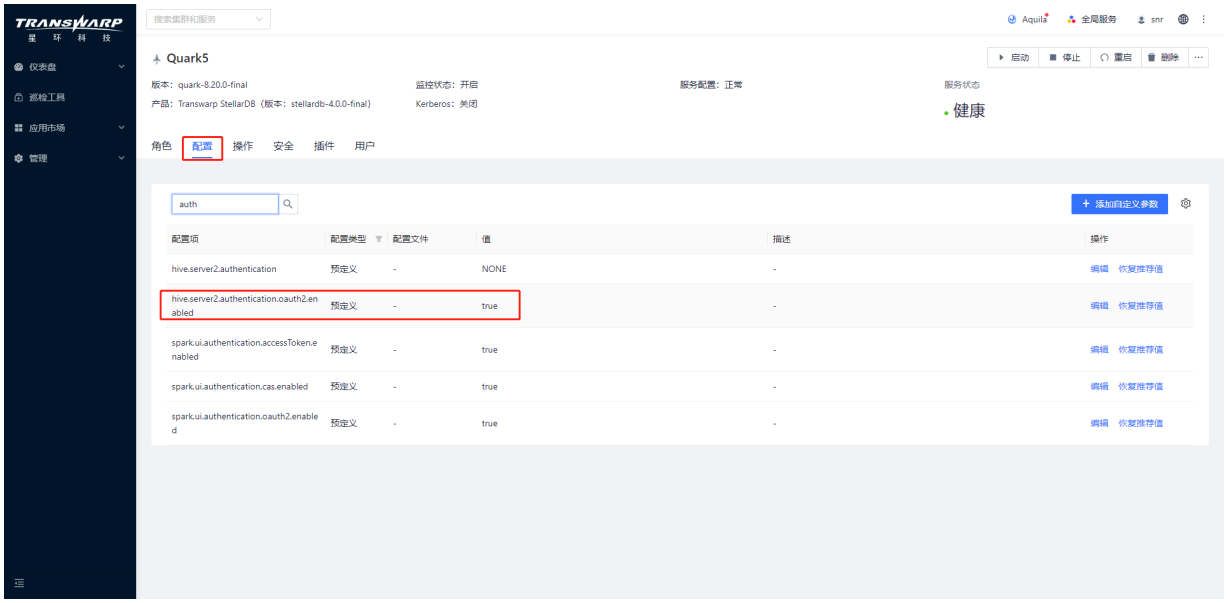
11.1.2. KG Explorer的用户登录认证

当依赖的Quark/QuarkGateway服务开启用户认证的时候（Kerberos认证或LDAP认证），KG Explorer提供了用户登录认证功能。

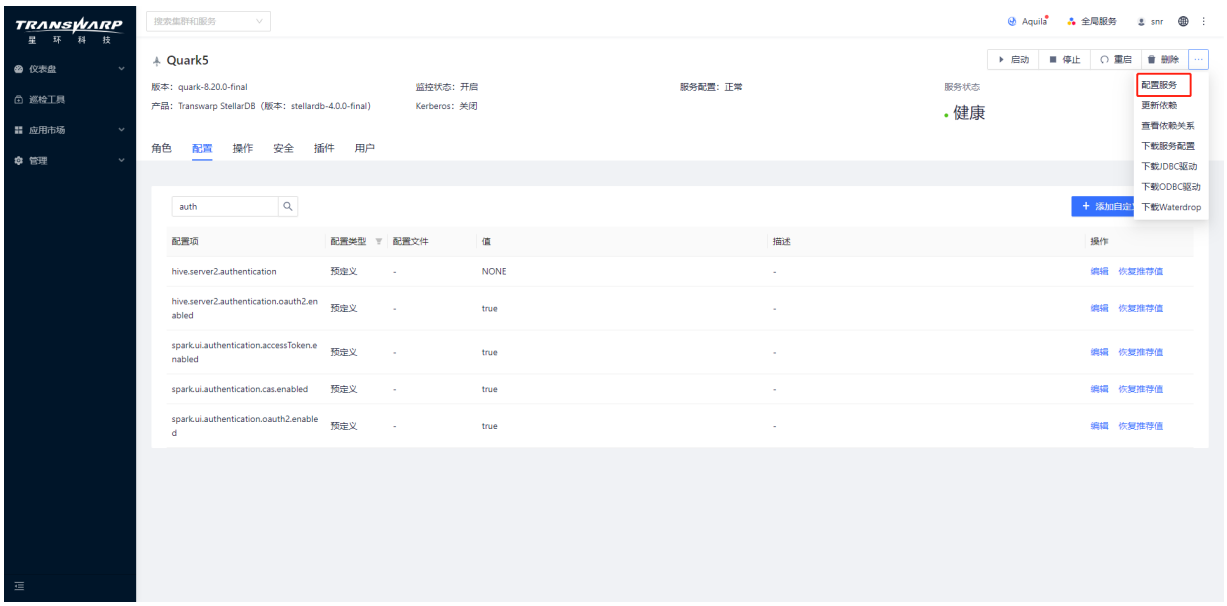
11.1.2.1. 开启KG Explorer的用户登录功能的步骤

1. 确保依赖的Quark/QuarkGateway服务的参数 `hive.server2.authentication.oauth2.enabled`

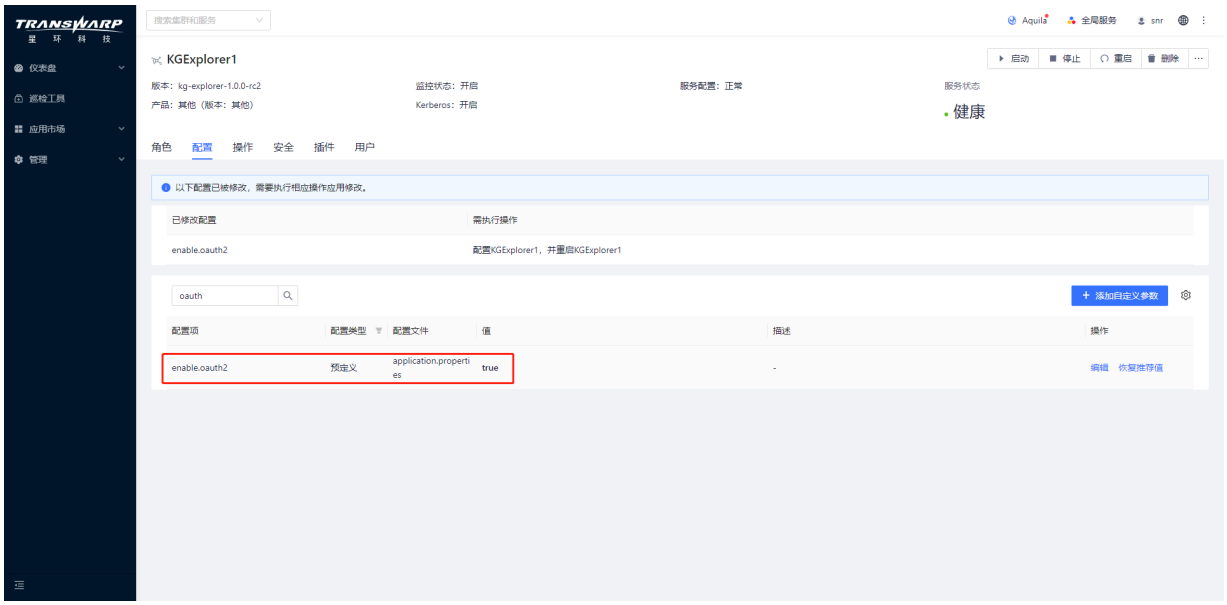
值为 **true**。



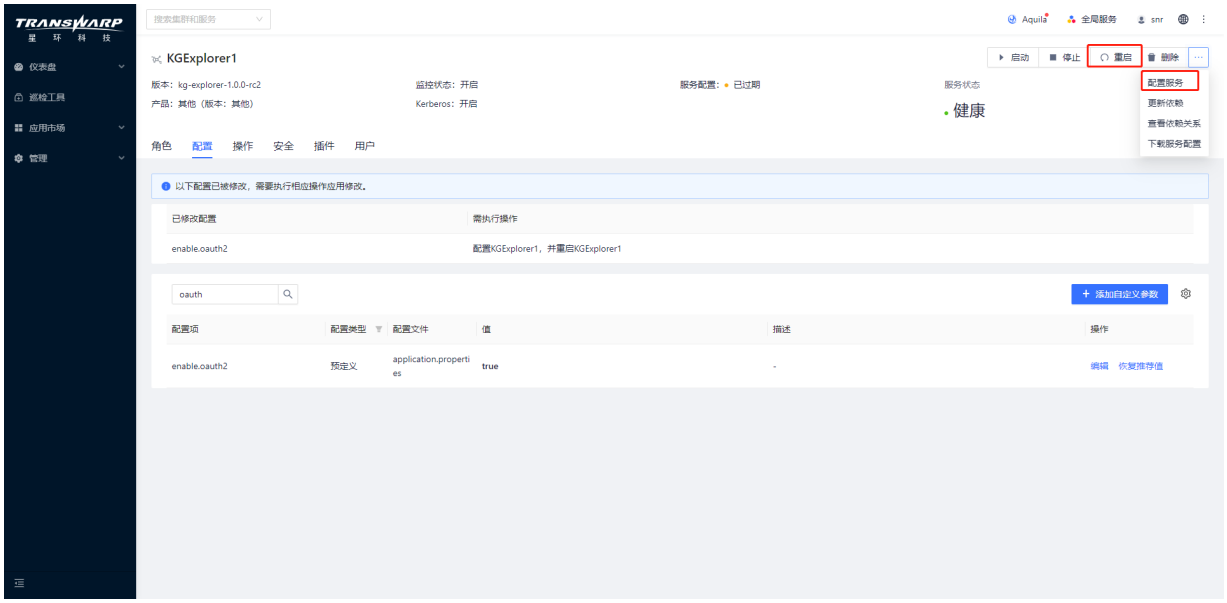
2. 若修改了Quark/QuarkGateway服务的配置，需要进行配置服务并重启Quark/QuarkGateway服务。



3. 修改KG Explorer服务的参数 `enable.oauth2` 值为 **true**。



4. 进行配置服务并重启KG Explorer。

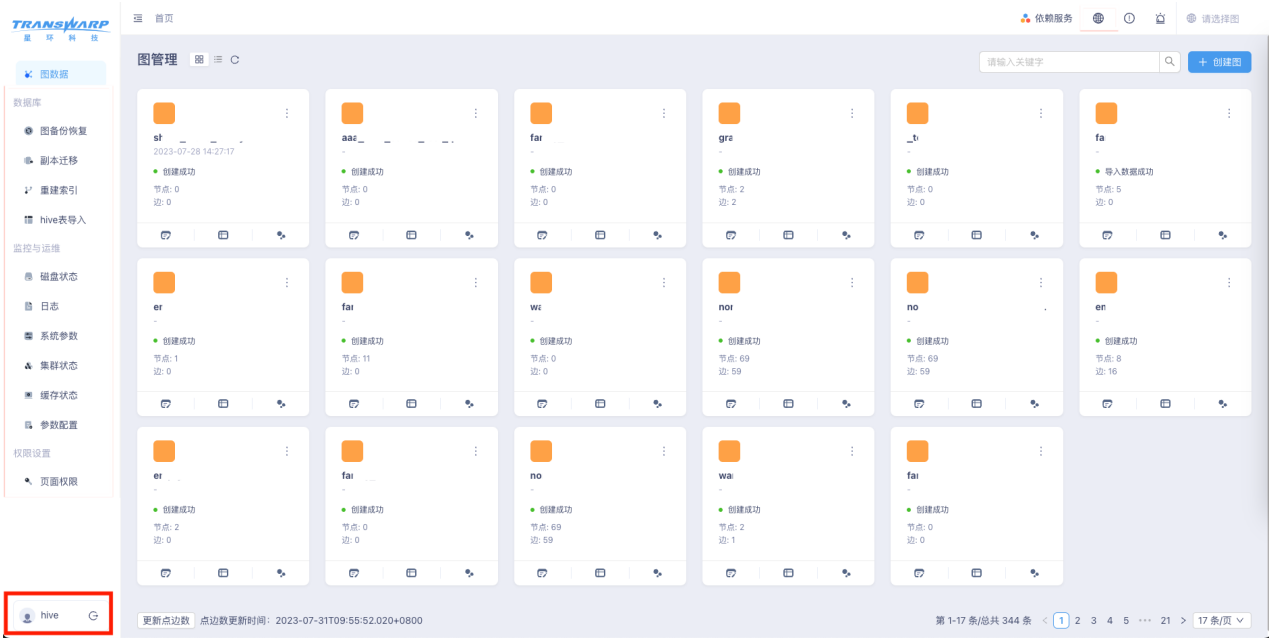


至此KG Explorer已经开启了用户登录功能。

11.1.2.2. KG Explorer的用户登录以及登出

打开KG Explorer页面会跳转到Guardian Federation的登录页面，用户可以按如图方式登录：

若KG Explorer开启了用户认证，登陆后想切换登录用户，可以鼠标移动到右上角的用户名，在弹出框中点击**登出登录** 按钮进行登出，页面会自动跳转到登录页面，即可切换用户进行登录



11.2. StellarDB权限控制

11.2.1. 开启图权限

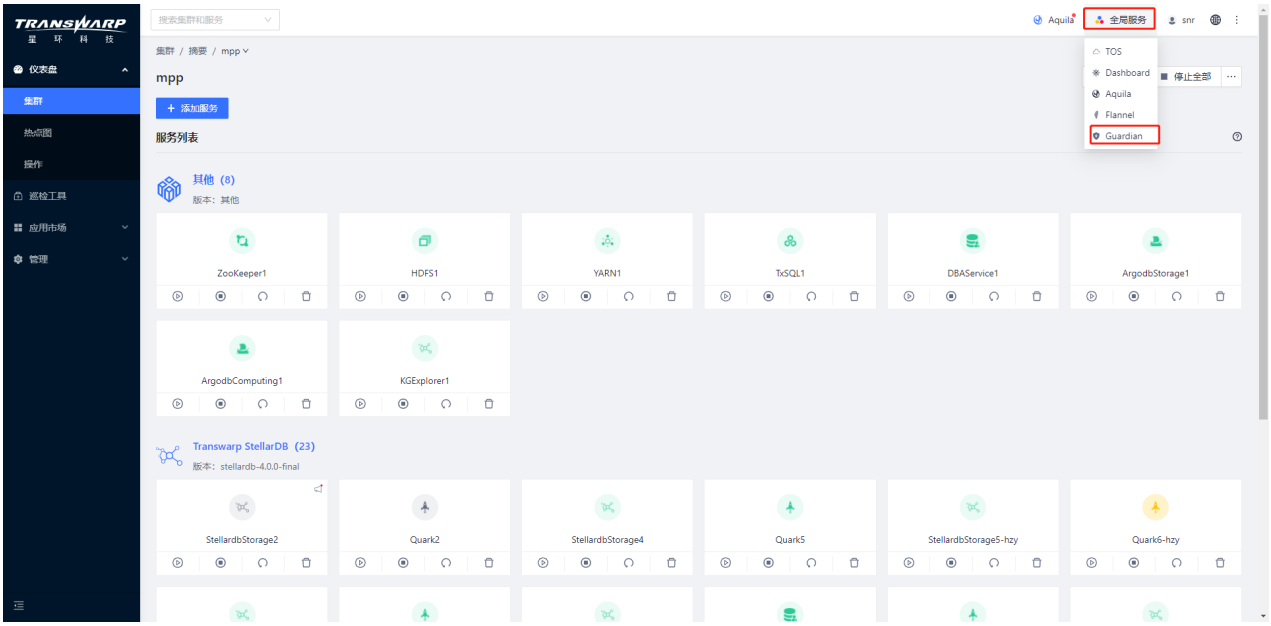
图权限功能需要安装Guardian 3.1.6以上或Guardian 3.2.2以上版本，且保证Guardian正常运行。

在Transwarp Manager的Quark服务中，添加自定义参数 `crux.auth.enable` 为 **true**，即可使用图权限功能.

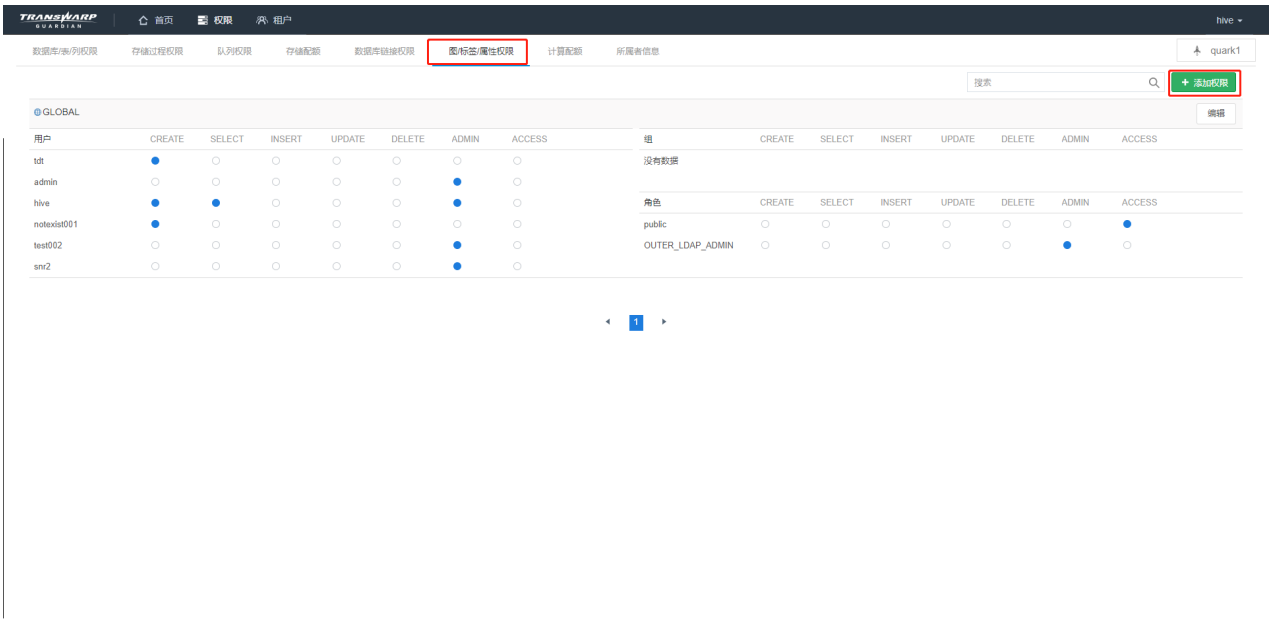
同时使用 `config crux.execution.mode analysis;` 切换至分析模式时，可以获得因权限失败的计数。

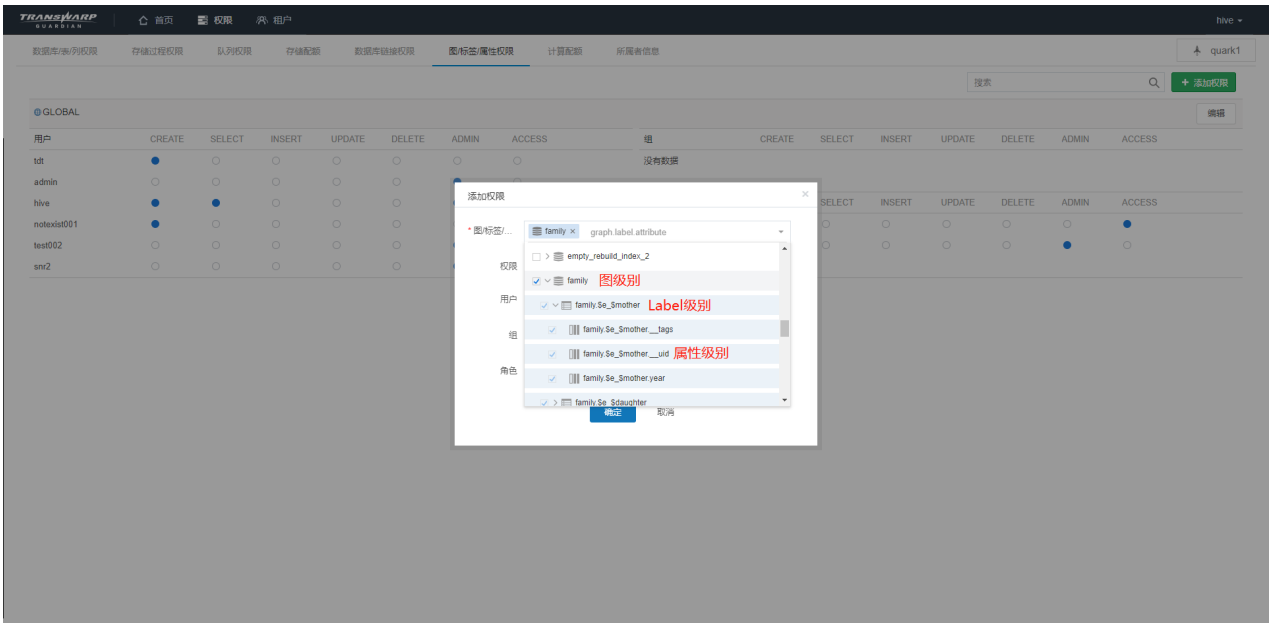
11.2.2. 设置用户权限

StellarDB的权限通过Guardian插件进行管理，在Transwarp Manager界面中点击 **全局服务** → **Guardian** → **Guardian Server** → 查看，使用具有 **global admin** 权限的用户登录Guardian。



点击 **权限** → **Inceptor** → 选择对应的Quark进入权限配置页面，点击 **图/标签/属性权限** → 添加权限





上图中权限格式:<GraphName>.$<e/v>_$<Label>.<PropName> ，其中$<e/v>用于区分边(edge)或点(vertex)的同名Label

11.2.3. 权限粒度与类别

StellarDB支持的权限控制分为全局（Global）、图（Graph）、标签（Label）、属性（Props）四种粒度，每个粒度对应的权限类别也有所区别，对应关系见下表：

表 3. 权限类别

权限类别	功能
SELECT	可以做查询，查询数据的粒度可以通过操作粒度权限实现
INSERT	可以做数据插入，插入数据的粒度通过操作粒度权限实现
DELETE	可以删除数据，删除数据的粒度通过操作粒度权限实现。
UPDATE	可以更新数据，更新数据的粒度通过操作粒度权限实现。
ADMIN	有上面所有的用户操作权限，且拥有全部的操作粒度权限

表 4. 权限粒度

Guardian权限	Global	Graph	Label	Props
CREATE	创建图的权限	/	/	/
SELECT	对全局所有图的点/边的SELECT权限	对图内所有Label的点/边的SELECT权限	对该Label的点/边的SELECT权限	对该属性的SELECT权限
INSERT	对全局所有图的点/边的INSERT权限	对图内所有Label的点/边的INSERT权限	对该Label的点/边的INSERT权限	/
UPDATE	对全局所有图的点/边的UPDATE权限	对图内所有Label的点/边的UPDATE权限	对该Label的点/边的UPDATE权限	对该属性的UPDATE权限

Guardian权限	Global	Graph	Label	Props
DELETE	对全局所有图的点/边的DELETE权限	对图内所有Label的点/边的DELETE权限	对该Label的点/边的DELETE权限	/
ADMIN	对全局所有图的所有权限，包括以上权限和管理权限（删除、重命名、修改schema、图备份和恢复等）	对图的ADMIN权限	/	/
ACCESS	对当前Quark服务的访问权限。ACCESS是服务级别的权限，只能在GLOBAL中进行权限编辑	/	/	/

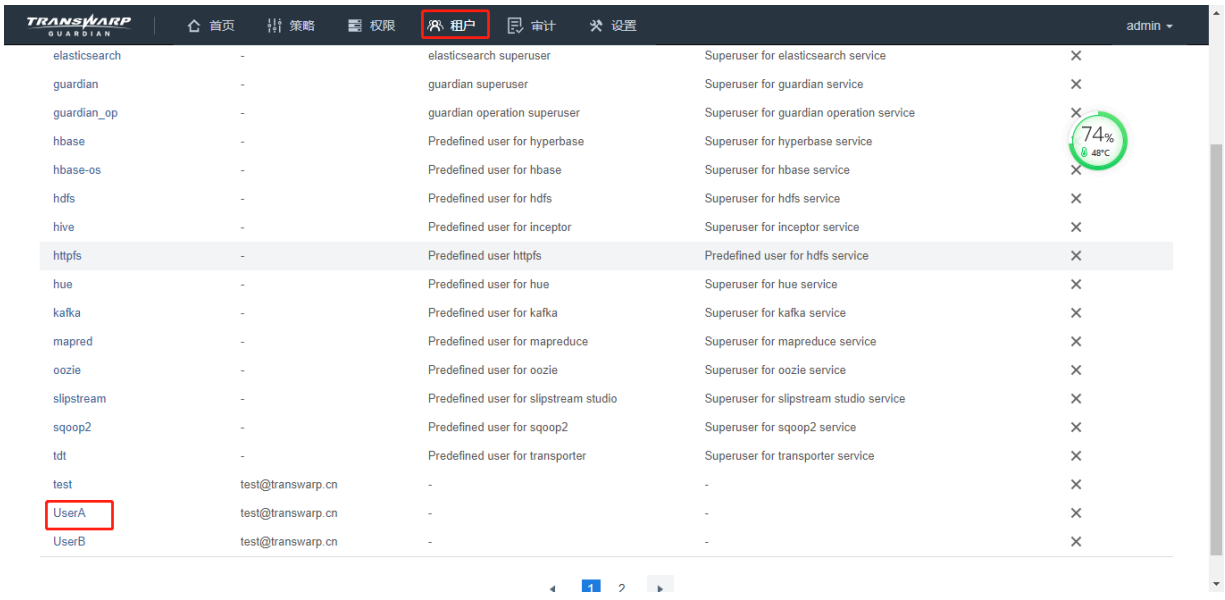
权限粒度从全局（Global）到属性（Props）逐渐细分，粗粒度的权限设置会覆盖更细粒度，如用户对某label设置权限，则该用户对该label下所有的Props都具有该权限。

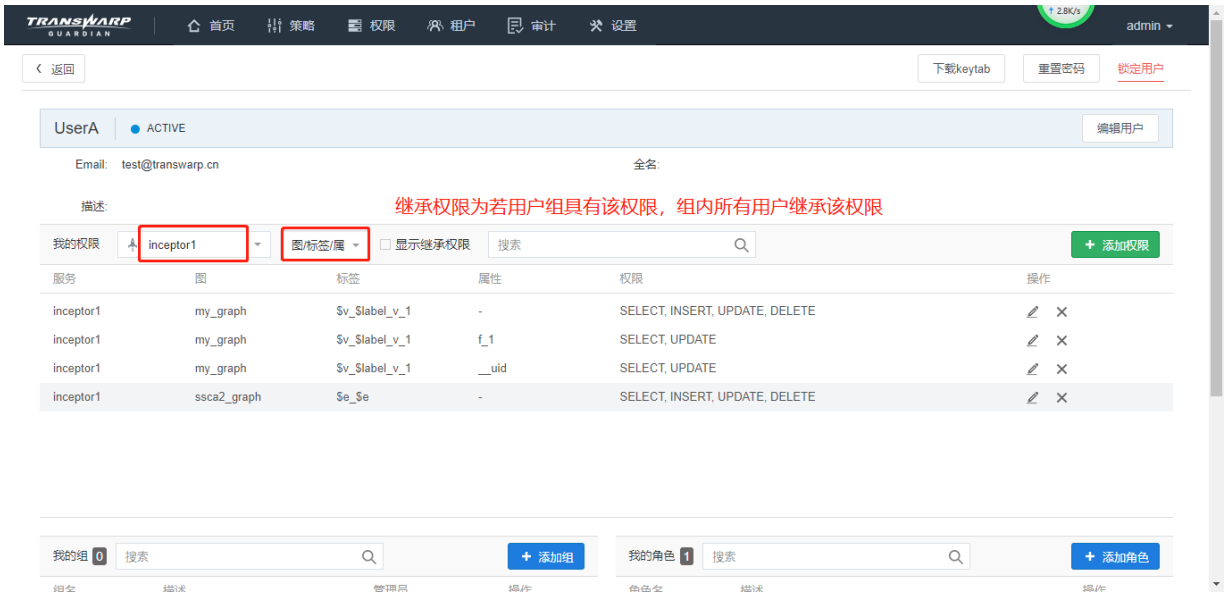
若需对点或者边的特定属性设置权限，除了需要设置的属性外，还要赋予点或者边的对应label的__uid字段对应权限，该字段权限决定整个点或者边是否对用户可见，任何查询更新操作都需要有该字段的权限。

11.2.4. 查看用户权限

1. 在Guardian界面中查看用户权限

   可以按照前述步骤在Guardian界面进行权限设置，并在租户界面选择对应用户查看不同服务的权限，下图以自定义用户UserA为例查看图权限：





2. 在命令行中查看用户权限

   除了Guardian外，您也可以在beeline客户端、KG Explorer等常用访问工具中使用TEoC语句查看登录用户的图权限：

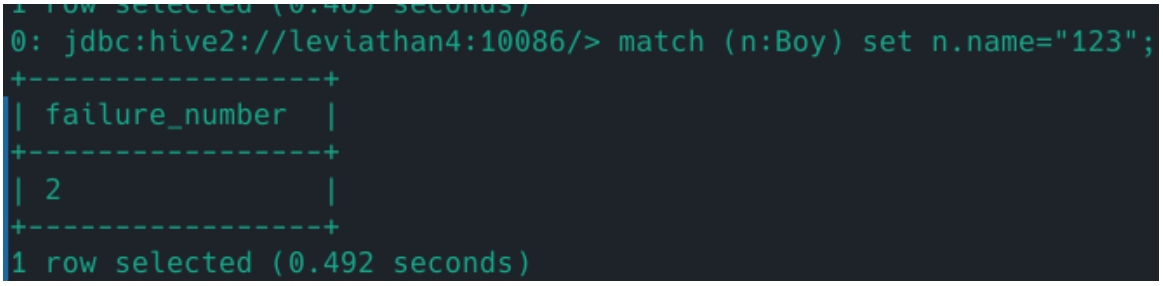
   ```
   show graph [graph_name] permission;
   ```

3. 当权限不足时显示失败计数

   设置Quark中 `crux.auth.enable` 为 **true**，同时 `config crux.execution.mode analysis;`，即可显示因权限不足而失败的操作数。

   ```
   set crux.auth.enable=true;
   config crux.execution.mode analysis;
   match (n:Boy) set n.name="123";
   ```

   结果如下图：



11.2.5. 应用分析

语句	分析
match (a:PAGE) set a.url="transwarp"	1.有PAGE点__uid、Label的SELECT权限，以及PAGE点__uid、Label、url属性的UPDATE权限（三个权限缺一则无法做UPDATE操作），此时可以成功更新。 2.没有PAGE点uid或Label的SELECT权限，此时就算有PAGE点uid、Label、url属性的UPDATE权限，也会因为没办法读取到完整的数据而不会执行任何的更新操作。
match (a:PAGE) set a.url=a.url1	1.有PAGE点__uid、Label、url1的SELECT权限，以及PAGE点__uid、Label、url属性的UPDATE权限（三个权限缺一则无法做UPDATE操作），此时可以成功更新，将url1属性写入url中。 2.有PAGE点__uid、Label的SELECT权限，但是没有url1的SELECT权限。如果此时拥有__uid、Label、url属性的UPDATE权限，因为PAGE点的uid，LABEL均为可读的，只是读不到url1的属性，可以执行更新操作，但是会将url的属性全部置为null。 3.没有PAGE点__uid或Label的SELECT权限，此时就算有PAGE点__uid、Label、url属性的UPDATE权限，也会因为没办法读取到完整的数据而不会执行任何的更新操作。
create (:PAGE2{uid:'99',url:'transwarp'})-[:PAGE1{uid:'100',url:'transwarp'}]→(:PAGE3{__uid:'101',url:'transwarp'});	同时拥有PAGE1,PAGE2,PAGE3的SELECT和UPDATE权限，并且拥有PAGE1.url、PAGE2.url、PAGE3.url的prop权限。若缺失Label权限则不会执行对应Label的create，若缺失prop权限，则对应prop不会被create。

- hive/admin用户是超级用户，拥有最高权限。
- 对于自定义用户，可以给一个用户开放多个图，同一个图也可以开放给多个用户。
- 图的创建者默认拥有该图的所有权限，并且他们的权限无法被deny策略回收
- 只有拥有该图admin权限的用户才可以给其他用户授权操作。

11.2.6. 附录：常用TEoC关键字与Guardian权限的对应关系

Guardian权限	常用TEoC关键字
CREATE	create graph copy graph copy graph schema
SELECT	match vertex/edge
INSERT	create vertex/edge bulkload match…create…(需要配合SELECT权限)
UPDATE	match…set…(需要配合SELECT权限)
DELETE	match…delete…(需要配合SELECT权限)
ADMIN	drop graph rename graph Schema更新(add_field/delete_field/alter_graph_schema等) 以及其他所有操作

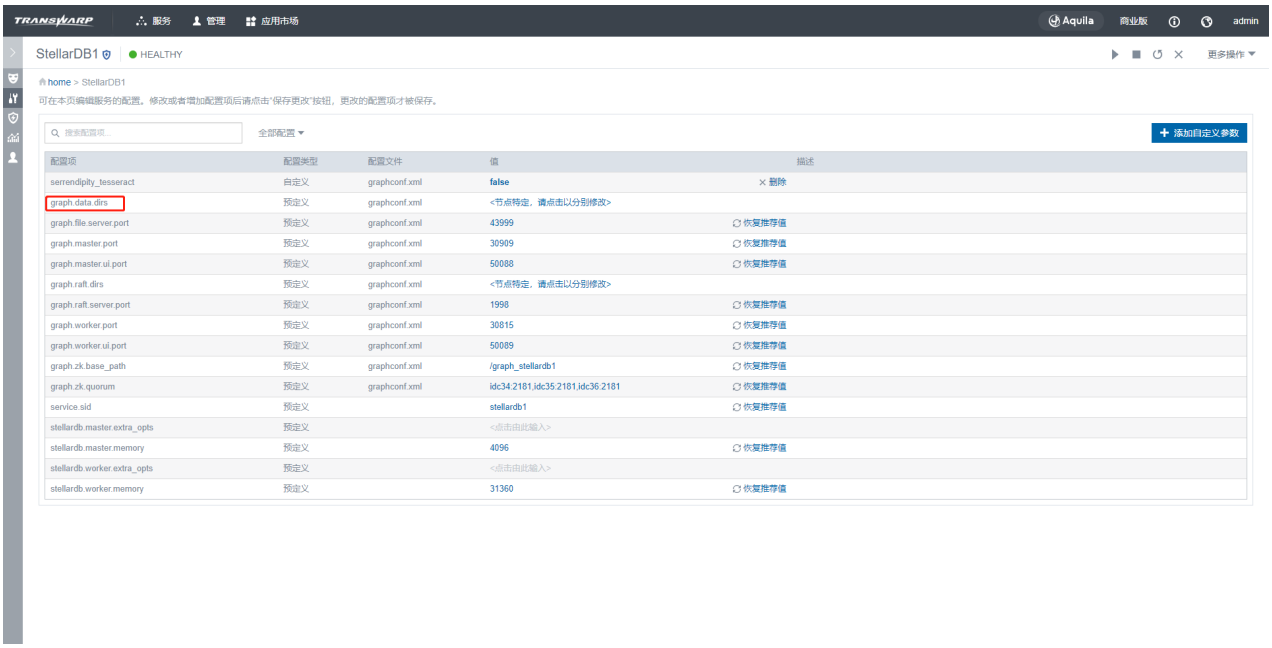
Guardian权限	常用TEoC关键字
ACCESS	explain desc graph show stellargraphs use graph

12. 常见问题及报错解决方案

12.1. 常见使用问题

Q:StellarDB存储数据文件的位置在哪里？

A:StellarDB服务配置页面 `graph.data.dirs` 配置项的值，即为每个节点存储数据文件的位置。



Q:如何配置StellarDB服务参数？

A:在配置界面找到需要修改的参数或添加自定义参数，设置完成后点击 **更多操作** → **配置服务**，配置完成后重启StellarDB或者Quark。

12.2. 常见安装问题

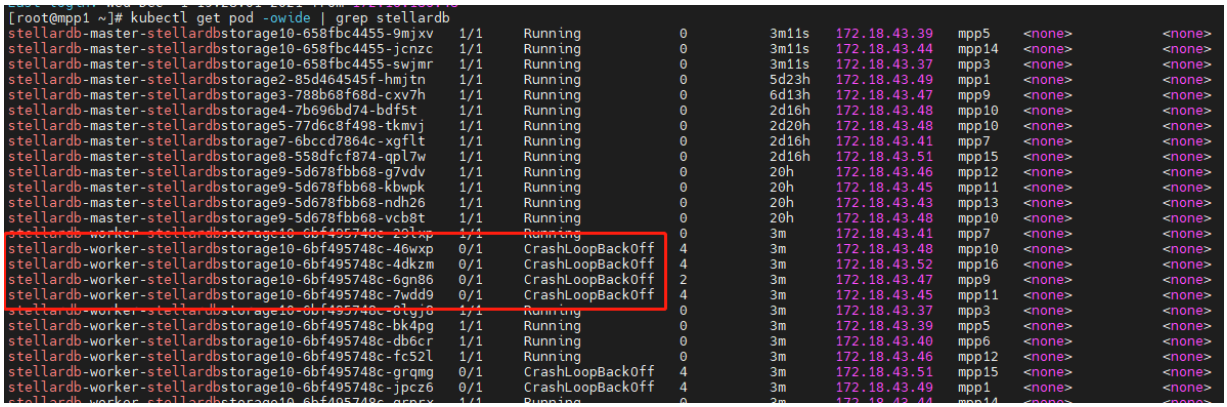
诊断流程：

1. 在任一节点执行：

   ```
   kubectl get pod -owide | grep <出问题的服务名称>
   ```

   例如：

   ```
   kubectl get pod -owide | grep stellardb
   ```



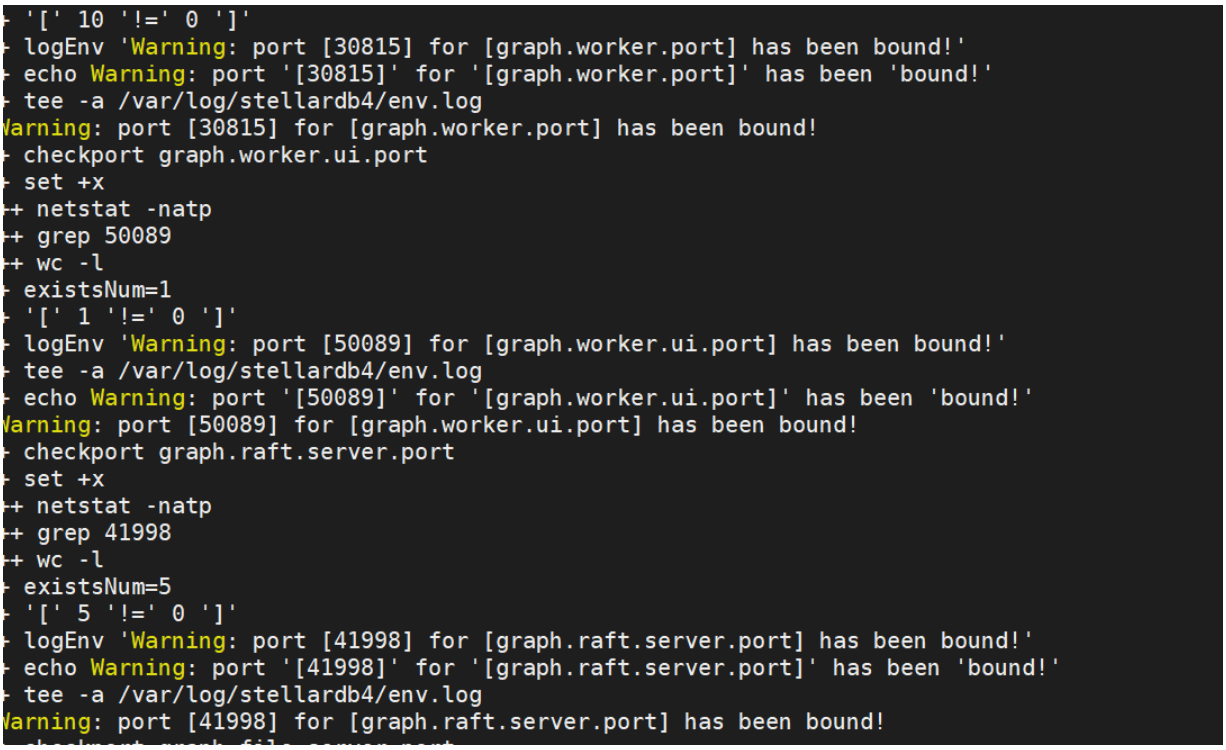
2. 找到有问题的pod，复制pod名称执行：

```
kubectl logs <pod名称>
```

例如：

```
kubectl logs stellardb-worker-stellardbstorage10-6bf495748c-46wxp
```

3. 日志中常见报错



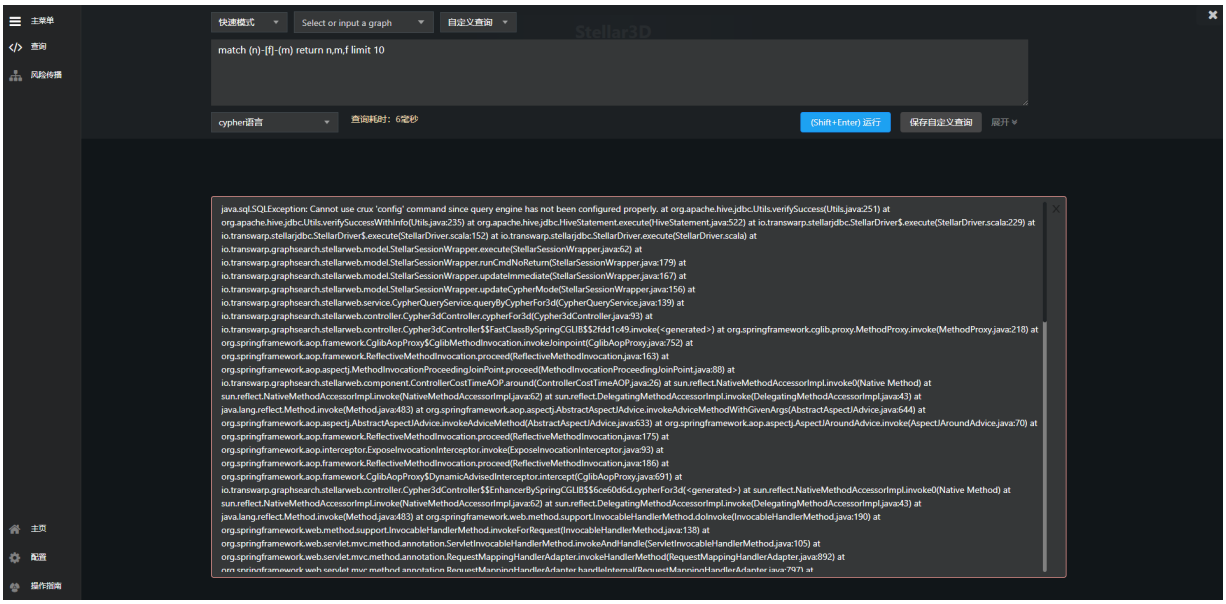
4. 原因及解决方式：

日志中提示端口号被占用，可在Transwarp Manager页面的基础参数页面修改被占用的端口号。

12.3. 常见使用报错及解决方案

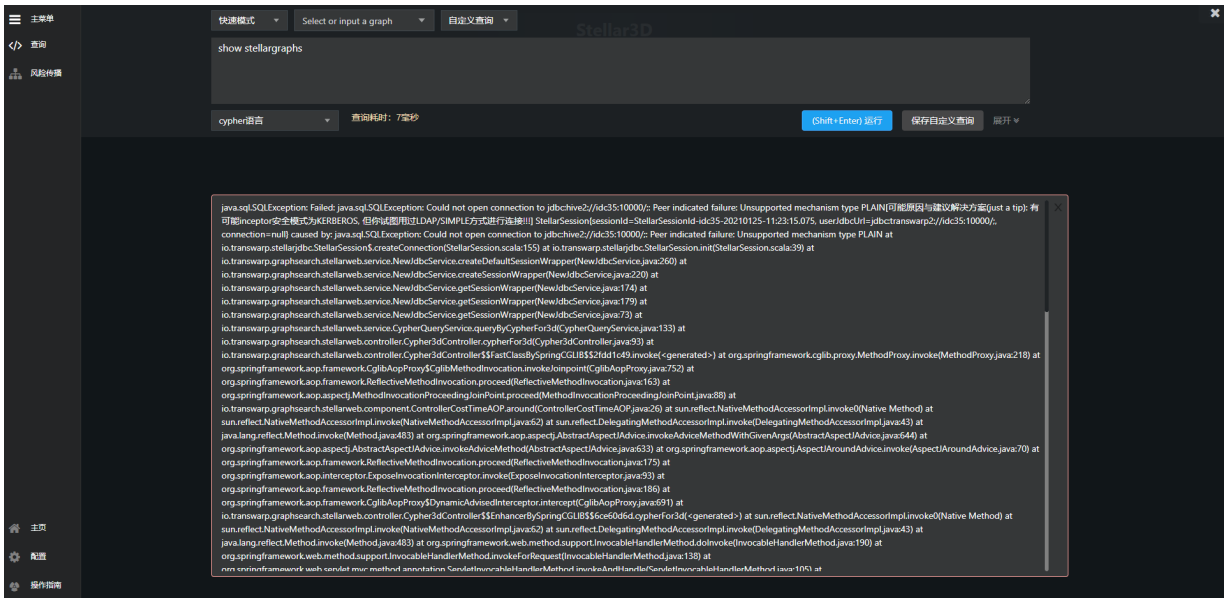
12.3.1. KG Explorer执行语句报错

1. 错误为"since query engine has not been configured properly"



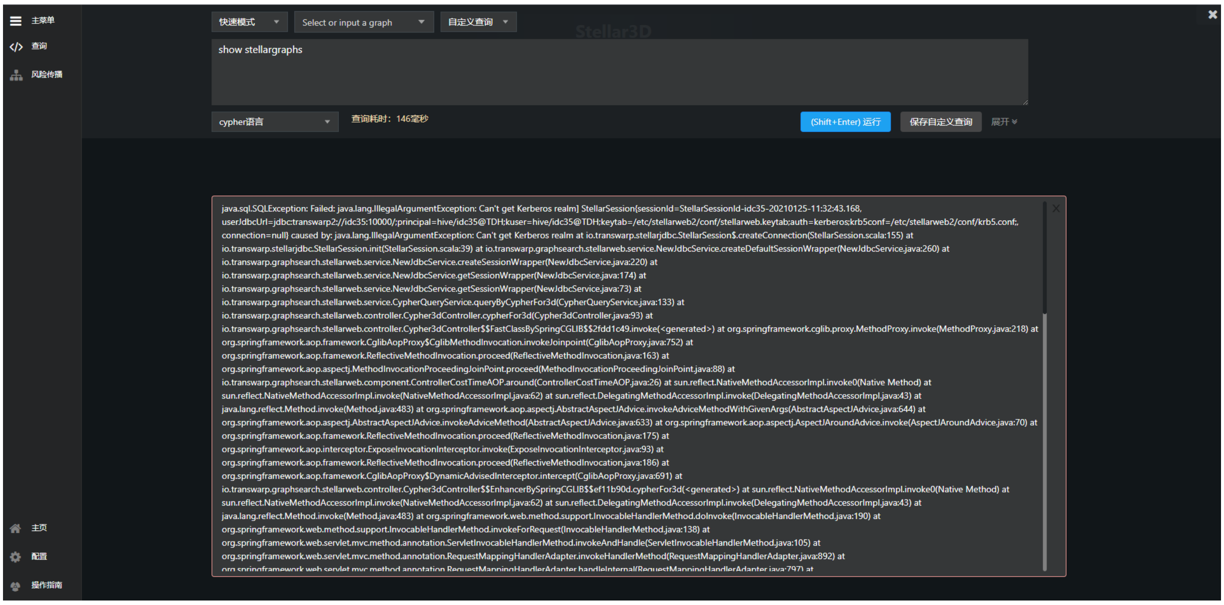
原因	解决方案
Quark没有依赖StellarDB	Transwarp Manager界面配置Quark依赖StellarDB，配置服务并重启Quark
Quark版本和StellarDB不适配	Quark需要使用StellarDB安装包中的版本，非TDH自带版本，重新安装和StellarDB适配的Quark

2. 错误为"Peer indicated failure: Unsupported mechanism type XXXX"



原因	解决方案
Quark修改了认证方式，而KG Explorer没有重启	重启KG Explorer

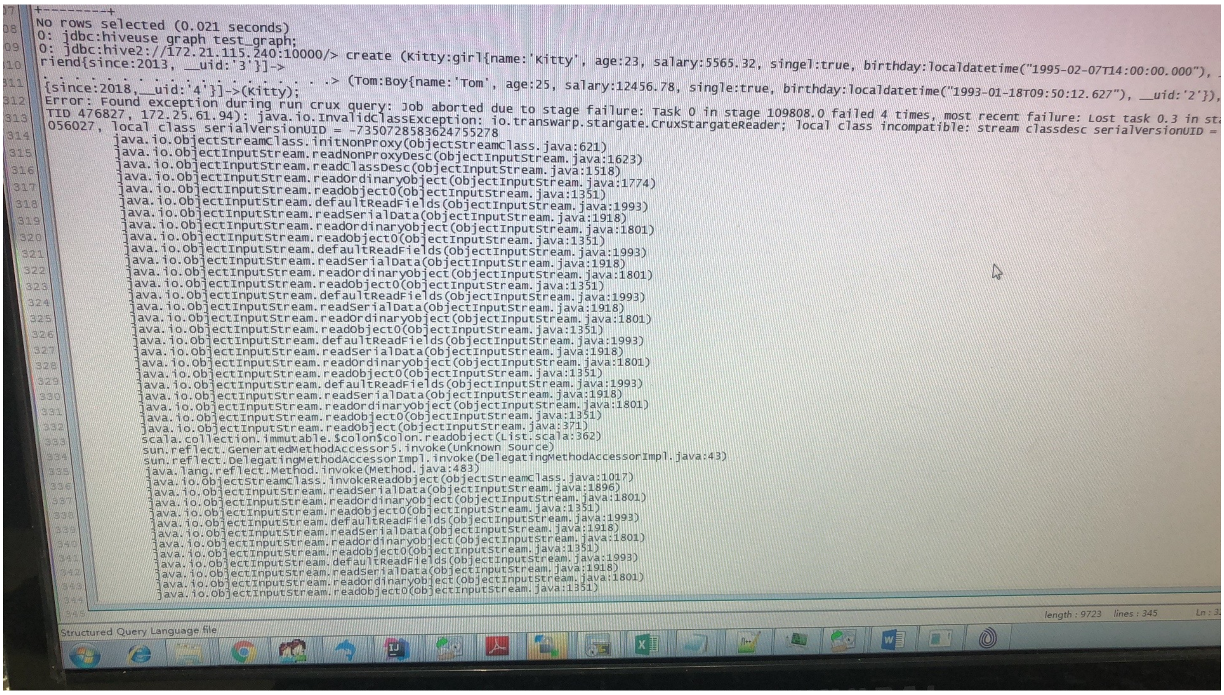
3. 报错为"Can’ t get Kerberos realm"



原因	解决方案
Quark开启了Kerberos认证，KG Explorer没有开启Kerberos认证	KG Explorer开启Kerberos认证

12.3.2. Quark执行语句报错

1. 报错为"local class incompatible"



原因	解决方案
Quark版本和StellarDB版本不兼容	Quark和StellarDB使用相同版本

2. 报错为"IllegalArgumentException:Rowkey could not be null,the given data is null"

```
Driver stacktrace: (state=,code=0)
0: jdbc:hive2://vqa33:10000/default> create (:v1{__uid:"create_2",col_int:1}),(:v2{__uid:"create_3",col_int:1});
Error: Found exception during run crux query: Job aborted due to stage failure: Task 0 in stage 14859.0 failed 4 times, most recent failu
re: Lost task 0.3 in stage 14859.0 (TID 28296, 172.22.7.35): java.lang.IllegalArgumentException: Rowkey could not be null, the given data
 is null | {__uid=create_3, $$__$$__uid=create_3, $$__$$__isvertex=true, col_int=1, $$__$$__ulabel=[v2]}
        io.transwarp.idbc.serde.StellarDBSerDe.parseValues(StellarDBSerDe.scala:203)
        io.transwarp.idbc.serde.StellarDBSerDe.parseGeneral(StellarDBSerDe.scala:103)
        io.transwarp.idbc.serde.StellarDBSerDe.serialize(StellarDBSerDe.scala:248)
        io.transwarp.idbc.serde.StellarDBSerDe.serialize(StellarDBSerDe.scala:60)
        io.transwarp.inceptor.execution.crux.writer.CruxStellarDBWriter.write(CruxStellarDBWriter.scala:69)
        io.transwarp.inceptor.execution.crux.CruxFileSinkOperator$$anonfun$writeCompactRows$1$1.apply(CruxFileSinkOperator.scala:121)
        io.transwarp.inceptor.execution.crux.CruxFileSinkOperator$$anonfun$writeCompactRows$1$1.apply(CruxFileSinkOperator.scala:117)
        scala.collection.Iterator$class.foreach(Iterator.scala:727)
        io.transwarp.inceptor.execution.crux.CruxIteratorFactory$$anon$1.foreach(CruxIteratorFactory.scala:66)
        io.transwarp.inceptor.execution.crux.CruxFileSinkOperator.writeCompactRows$1(CruxFileSinkOperator.scala:117)
        io.transwarp.inceptor.execution.crux.CruxFileSinkOperator.processPartition(CruxFileSinkOperator.scala:146)
        io.transwarp.inceptor.execution.Operator$$anonfun$executeProcessPartition$1.apply(Operator.scala:707)
        io.transwarp.inceptor.execution.Operator$$anonfun$executeProcessPartition$1.apply(Operator.scala:700)
        io.transwarp.nucleon.rdd.RDD$$anonfun$mapPartitionsWithContext$1$$anonfun$15.apply(RDD.scala:761)
        io.transwarp.nucleon.rdd.RDD$$anonfun$mapPartitionsWithContext$1$$anonfun$15.apply(RDD.scala:761)
        io.transwarp.nucleon.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:35)
        io.transwarp.nucleon.rdd.RDD.computeOrReadCheckpoint(RDD.scala:360)
        io.transwarp.nucleon.rdd.RDD.iterator(RDD.scala:327)
        io.transwarp.nucleon.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:35)
        io.transwarp.nucleon.rdd.RDD.computeOrReadCheckpoint(RDD.scala:360)
        io.transwarp.nucleon.rdd.RDD.iterator(RDD.scala:327)
        io.transwarp.nucleon.scheduler.ResultTask.runTask(ResultTask.scala:80)
        io.transwarp.nucleon.scheduler.Task.run(Task.scala:85)
        io.transwarp.nucleon.executor.Executor$TaskRunner$$anonfun$run$1.apply$mcV$sp(Executor.scala:480)
        io.transwarp.nucleon.deploy.SparkHadoopUtil.runAsUser(SparkHadoopUtil.scala:61)
        io.transwarp.nucleon.executor.Executor$TaskRunner.run(Executor.scala:414)
        java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        java.lang.Thread.run(Thread.java:745)
Driver stacktrace: (state=,code=0)
0: jdbc:hive2://vqa33:10000/default>
```

原因	解决方案
若同时对多个点或者边进行操作，其中任何一个点或者边没有权限所有点或者边均不会操作成功	配置相关权限

3. 报错为"operator xxx process rows exceed limit 100000"

原因	解决方案
超出 **immediate** 模式默认每个operator最大输出行数为100000	1. 使用 **analysis** 模式： config crux.execution.mode analysis; 2. 调大该参数： set ngmr.local.fast.operator.io.limit=1000000;

4. 报错包含"Syntax Error"关键字

原因	解决方案
查询存在语法问题	检查查询语句是否有语法错误

5. 报错为STORAGE_CONF文件找不到

```
0: jdbc:hive2://mpp11:10000> load relationship into graph xw_amazon from snap.snap_amaon_e with json schema '{"loaders":[{"reverse.direction.needed":f
alse,"graph.load.type":"edge","desc":{"uslabel":{"type":"STRING_CONSTANT","value":"v"},"uelabel":{"type":"STRING_CONSTANT","value":"e"},"udlabel":{"ty
pe":"STRING_CONSTANT","value":"v"},"udid":"duid","ueid":{"type":"STRING_CONSTANT","value":""},"properties":{"duid":"duid","suid":"suid","attr":"attr"}
,"usid":"suid"}}],"loadSchemaVersion":"1.0"}';
Error: Found exception during run crux query: Job aborted due to stage failure: Task 3 in stage 120.0 failed 4 times, most recent failure: Lost task 3
.3 in stage 120.0 (TID 373, 172.18.43.48): java.io.FileNotFoundException: /vdir/mnt/disk3/stellardb/graph/stellardb_tmp4/10004/01/1_10013/STORAGE_CONF
 (No such file or directory)
        java.io.FileInputStream.open(Native Method)
        java.io.FileInputStream.<init>(FileInputStream.java:138)
        java.io.FileInputStream.<init>(FileInputStream.java:93)
        java.io.FileReader.<init>(FileReader.java:58)
        io.transwarp.graphsearch.storage.log.StorageConf.parseFrom(StorageConf.java:70)
        io.transwarp.idbc.serde.StellarDBBulkLoadWriter.initOnSlave(StellarDBBulkLoadWriter.scala:487)
        io.transwarp.inceptor.execution.crux.CruxBulkLoadOperator.processPartition(CruxBulkLoadOperator.scala:195)
        io.transwarp.inceptor.execution.Operator$$anonfun$executeProcessPartition$1.apply(Operator.scala:717)
        io.transwarp.inceptor.execution.Operator$$anonfun$executeProcessPartition$1.apply(Operator.scala:710)
        io.transwarp.nucleon.rdd.RDD$$anonfun$mapPartitionsWithContext$1$$anonfun$15.apply(RDD.scala:767)
        io.transwarp.nucleon.rdd.RDD$$anonfun$mapPartitionsWithContext$1$$anonfun$15.apply(RDD.scala:767)
        io.transwarp.nucleon.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:35)
        io.transwarp.nucleon.rdd.RDD.computeOrReadCheckpoint(RDD.scala:366)
        io.transwarp.nucleon.rdd.RDD.iterator(RDD.scala:333)
        io.transwarp.nucleon.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:35)
        io.transwarp.nucleon.rdd.RDD.computeOrReadCheckpoint(RDD.scala:366)
        io.transwarp.nucleon.rdd.RDD.iterator(RDD.scala:333)
        io.transwarp.nucleon.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:95)
        io.transwarp.nucleon.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:45)
        io.transwarp.nucleon.scheduler.Task.run(Task.scala:94)
        io.transwarp.nucleon.executor.Executor$TaskRunner$$anonfun$run$1.apply$mcV$sp(Executor.scala:490)
        io.transwarp.nucleon.deploy.SparkHadoopUtil.runAsUser(SparkHadoopUtil.scala:61)
        io.transwarp.nucleon.executor.Executor$TaskRunner.run(Executor.scala:419)
```

原因	解决方案
StellardbStorage的配置目录在Quark pod中挂盘失败	1. 检查Quark的pod中 `/vdir/mnt/disk/../STORAGE_CONF` 文件是否存在，如果目录存在配置文件不存在，则直接重装Quark。 2. 如果是目录不存在，则检查该目录对应存储，在pod中挂载是否正常，不正常则直接重装Quark。 3. 最后，如果pod挂盘正常，但指向的宿主机host path一般为 `/mnt/disk/..`(去除/vdir)是否挂载正常，若不正常则在宿主机中 `/etc/fstab` 中加入挂载信息，再重装Quark。

13. 附录 – 常用系统参数汇总

13.1. 常用服务端口

在系统配置过程中，若发生端口冲突，请在配置过程中参阅下表中常用端口信息，按需在Transwarp Manager中对应组件的配置列表中修改相对应的服务端口号。

参数名	默认值	所属组件
server.port	38282	KG Explorer
graph.file.server.port	43999	StellardbStorage
graph.id.server.port	44147	StellardbStorage
graph.loader.upload.file.server.port	30906	StellardbStorage
graph.master.port	30909	StellardbStorage
graph.loader.upload.file.server.port	30906	StellardbStorage
graph.master.ui.port	50088	StellardbStorage
graph.raft.server.port	41998	StellardbStorage
graph.worker.port	30815	StellardbStorage
graph.worker.ui.port	50089	StellardbStorage
hive.metastore.port	9083	Quark
hive.server2.thrift.port	10000	Quark
inceptor.ui.port	8888	Quark
spark.driver.port	51888	Quark
dbaservice.request.port	60607	DBAService
dbaservice.gateway.message.port	60608	DBAService
dbaservice.message.port	60606	DBAService
dbaservice.ui.port	4040	DBAService

13.2. KG Explorer常用系统参数

本节总结KG Explorer常用的系统参数，更多系统参数信息，请参阅章节《KG Explorer使用文档》。

13.2.1. Quark LDAP认证方式相关参数

若在Transwarp Manager中的Quark组件下配置认证方式为LDAP，即 `hive.server2.authentication` 配置项的值为LDAP时，请参阅下表在Transwarp Manager的KG Explorer组件的配置列表中配置相应的访问Quark的用户名和密码。

参数名	默认值	参数说明
stellar.dependency.inceptor.defaultUserName	hive	默认访问依赖Quark的用户名
stellar.dependency.inceptor.defaultUserPassword		默认访问依赖Quark的用户密码

13.2.2. KG Explorer页面权限和用户校验相关参数

若在使用KG Explorer过程中需要使用到下表中的功能，请确保Guardian插件已经安装，且为开启状态。下表所列参数需要在Transwarp Manager的KG Explorer组件的配置列表中进行修改。

参数名	默认值	参数说明
stellar.pagePermission.enable	false	是否开启KG Explorer页面权限功能
stellar.restApi.auth	true	对外查询接口是否需要进行用户校验

13.2.3. KG Explorer页面失效时间相关参数

以下参数控制浏览器cookie或者session失效的时间，需要在Transwarp Manager中进行配置。

参数名	默认值	参数说明	备注
server.servlet.session.cookie.max-age	1800（单位为秒）	浏览器cookie失效时间	若开启用户登录之后，cookie失效会跳转到登录页面重新登录，用户可以视情况调整失效时间
server.servlet.session.timeout	1800（单位为秒）	浏览器session失效时间，需要自定义添加该配置项	若开启用户登录之后，session失效也会跳转到登录页面重新登录，用户可以视情况调整失效时间

13.2.4. JDBC相关参数

KG Explorer使用JDBC连接StellarDB图数据库，以下为常用配置参数，请按需在Transwarp Manager中修改配置，或添加自定义参数配置。

参数名	默认值	是否必须	参数说明	备注
stellar.jdbc.sessionPool.maxCount	100	是	session数目上限	
stellar.jdbc.sessionPool.maxCountForUser	20	是	单个用户的session数目上限	
stellar.jdbc.sessionPool.createNewSessionEachTime	false	是	是否每次操作图数据库都开启一个新的StellarJDBC session	

参数名	默认值	是否必须	参数说明	备注
stellar.jdbc.sessionInvalidTime	30min	否	StellarJDBC session的空闲失效时间	可以设置的比依赖的Quark服务上实际的session的空闲失效时间短一点，让KG Explorer提前进行关闭

13.2.5. 其他参数

以下参数可在Transwarp Manager中进行配置，或自定义配置。其他参数详细信息请参阅章节《KG Explorer使用文档》。

参数名	默认值	是否必须	参数说明	备注
server.ssl.enabled	false	否	是否开启HTTPS	
stellar.dependency.inceptor.hostPort	依赖的Quark服务的地址	是	依赖的Quark服务的地址：{Quark Server的host}:{Quark配置中的的hive.server2.thrift.port}	默认会自动获取，不需要手动添加该参数。当KG Explorer需要连接QuarkGateway时修改该参数为QuarkGateway服务的地址
security.oauth2.enable	false	否	是否开启跳转Guardian Federation跳转登录	开启KG Explorer跳转登录详细步骤见章节《KG Explorer使用文档》

13.3. StellarDB常用系统参数

以下参数可在Transwarp Manager中StellarDB组件中按需进行配置。

参数名	默认值	是否必须	参数说明
stellardb.master.memory	4096	是	Master角色内存大小
stellardb.worker.memory	31360	是	Worker角色内存大小
graph.drop.graph.policy	direct	否	配置回收站功能是否开启，默认不开启，可选值为 **direct** 和 **recycle**，该配置参数需要在Transwarp Manager中手动添加

13.4. 数据操作、图算法调用过程常用参数

13.4.1. 创建图时常用参数

以下参数通常在使用TEoC语句创建图的时候指定，使用方法请参考章节使用TEoC语法创建图。

参数名	默认值	是否必须	参数说明	备注
graph.shard.number	（图级别手动配置）	是	图分片数量，通常在创建图时配置	（点数+边数）小于等于1百万，推荐的值为1； （点数+边数）小于等于1千万，推荐的值为集群中节点数量； （点数+边数）小于等于1亿，推荐的值为集群中磁盘总数； （点数+边数）约等于N亿，推荐的值为max（N,集群中磁盘总数）
graph.replication.number	图级别手动配置	否	图副本数量，通常在创建图时配置	推荐配置为3，或者大于3的奇数
graph.encryption.type	NO_ENCRYPTION	否	配置是否需要对图数据进行加密，可选值为 NO_ENCRYPTION 和 SM4_CTR	使用 SM4_CTR 国标分组加密方式，会导致数据库读写性能下降

13.4.2. 切换查询语言、查询模式参数

以下参数通常在使用命令行操作的场景频繁使用，例如beeline、JDBC等。

参数名	默认值	参数说明	备注	使用方法
query.lang	sql	命令行客户端切换查询语言，查询语言有SQL和TEoC		config query.lang cypher;
crux.execution.mode	immediate	命令行客户端切换查询语句执行模式，执行模式有 immediate 和 analysis	immediate 模式多用于并发及端查询场景，查询结果和中间结果不超过百万；analysis 模式多用于图分析场景，创建图、插入数据以及图算法相关的语句必须在该模式下进行	config crux.execution.mode analysis;

13.4.3. bulkload数据校验常用参数

参数名	默认值	参数说明	使用方法
crux.bulkload.validate.enabled	false	使用bulkload进行数据导入时候，进行数据验证，需要手动开启。若开启，则必须配置 bulkload.partition.default.size 和 bulkload.failed.row.ratio	set crux.bulkload.validate.enabled=true;
crux.bulkload.partition.default.size	（手动设置）	需要在开启bulkload验证开关后进行手动配置，值乘以 bulkload.failed.row.ratio 的值为非法数据上限数量	set crux.blkload.partition.default.size=1000000;
crux.bulkload.failed.row.ratio	（手动设置）	需要在开启bulkload验证开关后进行手动配置，值乘以 bulkload.partition.default.size 的值为非法数据上限数量	set crux.bulkload.failed.row.ratio=0.01;
crux.stellardb.bulkload.err.output.enabled	false	将bulkload引擎的错误信息输出到HDFS目录中。启用后，当bulkload引擎遇到错误数据时，除了已有的输出到日志中，也会把错误数据输出到HDFS目录。该目录会在bulkload结束时通过语句显示。	set crux.stellardb.bulkload.err.output.enabled=false;

13.4.4. StellarDB重建索引时常用参数

以下参数需要在Transwarp Manager中StellardbStorage组件的配置项中进行修改。

参数名	默认值	是否必须	参数说明	备注
graph.db.rebuild.index.threads	32	否	配置StellarDB重建索引时使用的线程数	按需修改
graph.rebuild.index.progress.track.enabled	true	否	配置StellarDB是否追踪索引重建阶段的进度	关闭进度追踪可以提升重建索引速度
graph.rebuild.index.wait.start.max.time	60000（60秒）	否	配置StellarDB在索引重建任务时，等待目前批量的导入任务完成的最长等待时间	索引重建不可与批量导入同时执行，但可以等待批量导入任务完成。如果系统频繁进行批量导入操作，且批量导入任务重要性较高，可以将本值设低，使重建索引任务更容易等待超时、快速自行取消

13.4.5. 设置return语句结果中的列名显示参数

以下两个参数通常一起使用，默认值为 **false**，即StellarDB默认不显示有效列名。需要时，需要同时设置为 **true**。

参数名	默认值	参数说明	使用方法
crux.rename.column.name	false	使用return子句中的返回列作为有效列名显示	set crux.rename.column.name=true;
crux.rename.column.name.like.cypher	false	使用return子句中的返回列作为有效列名显示	set crux.rename.column.name.like.cypher=true;

13.4.6. 其他参数

参数名	默认值	是否必须	参数说明	备注
graph.client.storage.limit.check.enabled	true	否	启用或者关闭StellarDB图存储配额限制	需要在Transwarp Manager中配置
crux.dedup.relationship.switch	true	否	切换 **match** 查询语义成路径点重复还是不重复。**true** 表示重复，**false** 表示不重复	set crux.dedup.relationship.switch=true;
crux.queryplan.cache	true	是	在使用 **session级while** 循环的时候，需要关闭该参数消除其对自定义参数的影响	set crux.queryplan.cache=false;
crux.escape.sequence.enabled	true	否	关闭TEoC语句自动转义功能	set crux.escape.sequence.enabled=true;
crux.auth.enable	false	否	配置图权限功能	需要在Transwarp Manager中手动添加，或使用 `set crux.auth.enable = true;` 进行配置
ngmr.local.fast.operator.io.limit	100000	否	配置 **immediate** 模式返回结果数量	set ngmr.local.fast.operator.io.limit=100000;

13.5. JDBC常用配置参数

13.5.1. JDBC功能性配置参数

参数名	默认值	参数说明	使用方法
splitSQLBySemicolon	true	配置JDBC是否支持查询语句按分号切分。通常用于支持 **简单session级while循环** 查询，若不支持请更新客户端	使用时需要将 ";splitSQLBySemicolon=false" 添加到JDBC连接串后

13.5.2. JDBC优化参数

在使用优化参数时，可以通过将 -D参数名=参数值 添加到jvm的启动参数中。jvm启动参数 server.opts 可在Transwarp Manager的KG Explorer组件的配置列表中进行配置。

参数名	默认值	参数说明
stellarjdbc.log.off	true	是否将每一条执行语句输出到日志
stellarjdbc.concurrency	false	适合高并发场景。该值为 **true** 时，要求图的schema不能频繁变化. 为 **false** 时，用户执行 use graph graphName 时，JDBC内部会执行 describe graph graphName
stellarjdbc.optimize.for.gateway	false	是否在QuarkGateway场景为 **match** 语句添加 **hint**

14. 客户服务

技术支持

感谢您使用星环信息科技（上海）有限公司的产品和服务。如您在产品使用或服务中有任何技术问题，可以通过以下途径找到我们的技术人员给予解答。

email： support@transwarp.io

技术支持热线电话：4007-676-098

官方网址： http://www.transwarp.cn/

论坛支持： community.transwarp.cn

意见反馈

如果您在系统安装，配置和使用中发现任何产品问题，可以通过以下方式反馈：

email： support@transwarp.io

感谢您的支持和反馈，我们一直在努力！