更改记录

vertex加字段classInfo

graphd层

- 为了将classInfo插进去+查出来，需要从插入语句形式开始定义。目前支持如下语法插入类信息：

```
CREATE TAG player(name string, age int);
// 在RDFVERTEX后，tag信息前，写入(...)，需要有左右括号
// (123，234)为类信息，类型为int64，类的数量与插入点的个数匹配
INSERT RDFVERTEX (123，234) player(name, age) VALUES "p1":("Tim Duncan",
42),"p2":("Tom", 21);
```

- 下面是总的代码修改记录
- parser.yy
  - 为RDFINSERT加入新的格式，第三个的$3是类信息。
    - vertex_class_info_item是(123,234)，使用nebula已经包装好的value_list来包装。

```
2397  /* RDF */
2398  insert_RDFvertex_sentence
2399      : KW_INSERT KW_RDFVERTEX vertex_tag_list KW_VALUES vertex_row_list {
2400          $$ = new InsertRDFVerticesSentence($3, $5);
2401      }
2402      | KW_INSERT KW_RDFVERTEX KW_NO KW_OVERWRITE vertex_tag_list KW_VALUES vertex_row_list {
2403          $$ = new InsertRDFVerticesSentence($5, $7, false /* not overwritable */);
2404      }
2405      | KW_INSERT KW_RDFVERTEX vertex_class_info_item vertex_tag_list KW_VALUES vertex_row_list {
2406          $$ = new InsertRDFVerticesSentence($4, $6, $3, true /* overwritable */);
2407      }
2408      ;
2409
2410  vertex_class_info_item
2411      : L_PAREN value_list R_PAREN {
2412          $$ = new VertexClassInfoItem($2);
2413      }
2414      ;
```

  - 为了满足yacc语法，添加一些边角料

```
85      nebula::VertexTagList                *vertex_tag_list;
86      nebula::VertexClassInfoItem          *vertex_class_info_item;
87      nebula::VertexTagItem                *vertex_tag_item;
```

```
247   %type <vertex_tag_list> vertex_tag_list
248   %type <vertex_class_info_item> vertex_class_info_item
249   %type <vertex_tag_item> vertex_tag_item
```

  - yacc定义后，需要c++代码中有对应的类，即实现vertex_class_info_item的VertexClassInfoItem类
- VertexClassInfoItem类（src/parser/MutateSentences.h ）
  - 这里的Expression是仿照nebula使用Value类的方式弄的

```
105
106 class VertexClassInfoItem final{
107 public:
108     VertexClassInfoItem(ValueList *values){
109         values_.reset(values);
110     }
111
112     std::vector<Expression*> values() const {
113         return values_->values();
114     }
115 private:
116     std::unique_ptr<ValueList>                  values_;
117 };
118
```

- vertex_class_info_item处理好了后，需要修改负责收集全部信息的InsertRDFVerticesSentence类

- InsertRDFVerticesSentence类（src/parser/MutateSentences.h ）
  - 增加一个构造函数

```
210 // 在 src/validator/MutateValidator.cpp 中使用
211 class InsertRDFVerticesSentence final : public Sentence {
212 public:
213     InsertRDFVerticesSentence(VertexTagList *tagList,
214                               VertexRowList *rows,
215                               bool overwritable = true) {
216         tagList_.reset(tagList);
217         rows_.reset(rows);
218         overwritable_ = overwritable;
219         kind_ = Kind::kInsertRDFVertices;
220     }
221
222     InsertRDFVerticesSentence(VertexTagList *tagList,
223                               VertexRowList *rows,
224                               VertexClassInfoItem *classInfos,
225                               bool overwritable = true) {
226         tagList_.reset(tagList);
227         rows_.reset(rows);
228         overwritable_ = overwritable;
229         kind_ = Kind::kInsertRDFVertices;
230         classInfos_.reset(classInfos);
231     }
232
233     bool overwritable() const {
234         return overwritable_;
```

  - 加入相应的get函数

```
243     }
244
245     std::vector<Expression*> classInfos() const {
246         return classInfos_->values();
247     }
```

  - 加入classInfo_的字段

```
250
251 private:
252     bool                                    overwritable_{true};
253     std::unique_ptr<VertexTagList>          tagList_;
254     std::unique_ptr<VertexRowList>          rows_;
255     std::unique_ptr<VertexClassInfoItem>    classInfos_;
256 };
```

- 至此，信息已经完全从parser收集至类中。InsertRDFVerticesSentence类之后被src/validator/MutateValidator.cpp调用，继续修改。

- InsertRDFVerticesValidator类（src/validator/MutateValidator.cpp）
  - 先加入字段，把信息存住
    - classInfo_

```
60 private:
61     using TagSchema = std::shared_ptr<const meta::SchemaProviderIf>;
62     GraphSpaceID                                                spaceId_{-1};
63     std::vector<VertexRowItem*>                                 rows_;
64     std::unordered_map<TagID, std::vector<std::string>>         tagPropNames_;
65     std::vector<std::pair<TagID, TagSchema>>                    schemas_;
66     uint16_t                                                    propSize_{0};
67     bool                                                        overwritable_{false};
68     std::vector<Expression*>                                    classInfos_;
69     std::vector<storage::cpp2::NewRDFVertex>                    vertices_;
70 };
71
```

  - InsertRDFVerticesValidator类中共有4个相关函数，其执行顺序与调用关系如下
    - validateImpl()：处理上述收集的信息，主要是调用下面2个函数
      - check()：将数据拿出，进行数据是否合理的检查
      - prepareVertices()：准备点信息
    - toPlan()：所有信息处理好了，从validate进入下一步
  - check()
    - classInfo信息从sentence中拿出

```
176     }
177
178     classInfos_ = sentence->classInfos();
179     // 判断vertexClassInfo是否合理,此处需要判断class个数与点个数是否相等
180     if (false) {
181         // 随便写了一句
182         LOG(ERROR) << "No schema found for ";
183     }
184
```

  - prepareVertices()
    - 从vector<Expression*>转为vector<Value>，保存在classes中

```
221
222 Status InsertRDFVerticesValidator::prepareVertices() {
223     // Todo:把每个classinfo弄出来，给每个点加进去
224
225     // check value expr,貌似是判断一下value可否visit
226     for (auto &value : classInfos_) {
227         if (!evaluableExpr(value)) {
228             LOG(ERROR) << "Insert wrong value: `" << value->toString() << "'.";
229             return Status::SemanticError("Insert wrong value: `%s'.",
230                                          value->toString().c_str());
231         }
232     }
233     auto classesRet = SchemaUtil::toValueVec(classInfos_);
234     NG_RETURN_IF_ERROR(classesRet);
235     auto classes = std::move(classesRet).value();
236     for(auto &i: classes){
237         LOG(ERROR) << "MutateValidator.cpp classes:"<<i.toString()<<std::endl;
238     }
239
```

- 给每个点加入class信息

```
285
286         storage::cpp2::NewRDFVertex vertex;
287         vertex.set_id(vertexId);
288         vertex.set_classInfo(classes[i]);
289         vertex.set_tags(std::move(tags));
290         vertices_.emplace_back(std::move(vertex));
291     }
```

  - 可以看到，这里已经是NewRDFVertex了，下面插入一下如何修改的NewRDFVertex

- 修改storage.thrift接口（modules/common/src/common/interface/storage.thrift）
  - 加入新数据结构NewRDFVertex

```
438
439 struct NewRDFVertex {
440     1: common.Value id,
441     2: list<NewTag> tags,
442     3: common.Value classInfo,
443 }
444
```

  - 在Request中使用NewRDFVertex

```
462 struct AddRDFVerticesRequest {
463     1: common.GraphSpaceID                      space_id,
464     // partId => vertices
465     2: map<common.PartitionID, list<NewRDFVertex>>
466         (cpp.template = "std::unordered_map")   parts,
467     // A map from TagID -> list of prop_names
468     // The order of the property names should match the data order specifi
469     //   in the NewVertex.NewTag.props
470     3: map<common.TagID, list<binary>>
471         (cpp.template = "std::unordered_map")   prop_names,
472     // If true, it equals an (up)sert operation.
473     4: bool                                     overwritable = true,
474 }
475
```

  - 编译生成NewRDFVertex的数据结构后，就可以使用了

- 最后，需要修改一下GraphStorageClient.cpp和.h（modules/common/src/common/clients/storage/GraphStorageClient.cpp），其中有部分函数数据结构不支持NewRDFVertex
  - GraphStorageClient.h (modules/common/src/common/clients/storage/GraphStorageClient.h)

```
71      // rdf begin
72      folly::SemiFuture<StorageRpcResponse<cpp2::ExecResponse>> addRDFVertices(
73          GraphSpaceID space,
74          std::vector<cpp2::NewRDFVertex> vertices,
75          std::unordered_map<TagID, std::vector<std::string>> propNames,
76          bool overwritable,
77          folly::EventBase* evb = nullptr);
78      // rdf end
```

```
135   private:
136       StatusOr<std::function<const VertexID&(const Row&)>>
137           getIdFromRow(GraphSpaceID space, bool isEdgeProps) const;
138
139       StatusOr<std::function<const VertexID&(const cpp2::NewVertex&)>>
140           getIdFromNewVertex(GraphSpaceID space) const;
141   // rdf begin
142       StatusOr<std::function<const VertexID&(const cpp2::NewRDFVertex&)>>
143           getIdFromNewRDFVertex(GraphSpaceID space) const;
144   // rdf end
```

  - GraphStorageClient.cpp (modules/common/src/common/clients/storage/GraphStorageClient.cpp)

```
630   // rdf begin
631   StatusOr<std::function<const VertexID&(const cpp2::NewRDFVertex&)>>
632   GraphStorageClient::getIdFromNewRDFVertex(GraphSpaceID space) const {
633       auto vidTypeStatus = metaClient_->getSpaceVidType(space);
634       if (!vidTypeStatus) {
635           return vidTypeStatus.status();
636       }
637       auto vidType = std::move(vidTypeStatus).value();
638
639       std::function<const VertexID&(const cpp2::NewRDFVertex&)> cb;
640       if (vidType == meta::cpp2::PropertyType::INT64) {
641           cb = [](const cpp2::NewRDFVertex& v) -> const VertexID& {
642                   DCHECK_EQ(Value::Type::INT, v.id.type());
643                   auto& mutableV = const_cast<cpp2::NewRDFVertex&>(v);
644                   mutableV.id = Value(
645                           std::string(reinterpret_cast<const char*>(&v.id.getInt()), 8));
646                   return mutableV.id.getStr();
647               };
648       } else if (vidType == meta::cpp2::PropertyType::FIXED_STRING) {
649           cb = [](const cpp2::NewRDFVertex& v) -> const VertexID& {
650                   DCHECK_EQ(Value::Type::STRING, v.id.type());
651                   return v.id.getStr();
652               };
653       } else {
```

```
125   // rdf begin
126   folly::SemiFuture<StorageRpcResponse<cpp2::ExecResponse>>
127   GraphStorageClient::addRDFVertices(GraphSpaceID space,
128                                      std::vector<cpp2::NewRDFVertex> vertices,
129                                      std::unordered_map<TagID, std::vector<std::string>> propNames,
130                                      bool overwritable,
131                                      folly::EventBase* evb) {
132       auto cbStatus = getIdFromNewRDFVertex(space);
133       if (!cbStatus.ok()) {
```

- 至此，graphd层修改完毕

storage层

- storage层需要将class信息放入key中，修改较少
- AddRDFVerticesProcessor类
  （modules/storage/src/storage/mutate/AddRDFVerticesProcessor.cpp）
  - 把每个点的class信息拿出

```
75        for (auto& vertex : vertices) {
76            auto vid = vertex.get_id().getStr();
77            auto classInfo = vertex.get_classInfo().getInt();
78            const auto& newTags = vertex.get_tags();
```

  - 加入key中
    - 这里的LOG会报乱码，没法正常看到

```
96            }
97            // 改这里
98            auto key = NebulaKeyUtils::RDFvertexKey(spaceVidLen_, partId, vid, tagId, classInfo);
99            LOG(ERROR) << "The RDFvertexKey is: " << key << std::endl;
```

classInfo查出来

- 首先得先知道需要查出什么信息，看看各个推理相关的论文使用什么查询语句
  - inferray：有提到是“自己构建的benchmark”
  - 思卓姐：An Ontology-Aware Unified Storage Scheme for Knowledge Graphs
    - 插入数据的时间：原始版本vs语义版本
      - 语义版本 略慢一点，微乎其微，可以接受
    - 载入后数据大小：元数据vs原始版本vs语义版本vsNeo4j
      - 基本呈现这个趋势：原始版本<语义版本<Neo4j<元数据
    - LUBM给定查询的查询时间：原始版本vs语义版本
      - 语义版本比原始版本快