Aufgabe 3: Optimierung des Querymanagers mithilfe eines index-basierten Joins

Gruppe A

Bisheriger Algorithmus

- **Aufgabe: Optimierung des Querymanagers mithilfe eines index-basierten Joins**
- **Bisher: Nested Loop Join ohne Index**
  - SelectTuple() holt jeweils Relation aus Tabelle und selectJoinTuple() fügt Relationen zusammen
  - SelectTuple() wendet WHERE-Predicate an und berechnet Schnittmenge aus Ergebnissen
  - Nutzt Index auf Predicate, sonst kompletter Scan

Ideen zur Optimierung

- **Optimierung: Indexed Nested Loop Join nutzt Index auf Attribut, auf dem vereinigt wird statt innere Relation immer komplett zu durchlaufen**
- **Idee 1: selectTuple() wie vorher, aber mit zusätzlichem Index → Problem: Aufbauen zu teuer**
- **Idee 2: In innerer Relation nur Objekten mit passender ID auf Join-Attribut suchen→ Problem: Eintrag müsste aus TID geladen werden für andere Attribute**

Ideen zur Optimierung

- **Idee 3: selectTuple() prinzipiell wie vorher, aber zuerst find-Anfrage of Join-Attribut**

  → Bilden von weiterer Schnittmenge

- **void DBMyQueryManager::selectIndexedTuple(DBTable *table, DBListPredicate &where, DBListTuple &tuple, uint attrIndex, const DBAttrType &attrValue)**

Neue Funktion selectIndexedTuple()

```
DBListTID tidList;
const DBRelDef &def = table->getRelDef();
QualifiedName qname;
strcpy(qname.relationName, def.relationName().c_str());

DBIndex *index = NULL;
DBAttrDef adef = def.attrDef(attrIndex);
strcpy(qname.attributeName, adef.attrName().c_str());
try {
  index = sysCatMgr.openIndex(connectDB, qname, READ);
  index->find(attrValue, tidList);
  tidList.sort();
  delete index;
} catch (DBException e) {
  if (index != NULL)
    delete index;
  throw e;
}

DBListPredicate::iterator u = where.begin();
while (u != where.end()) {
```

Ideen zur Optimierung

Neue Join-Funktion:

void DBMyQueryManager::selectJoinTupleIndexedNested(DBTable *table[2], uint attrJoinPos[2], DBListPredicate where[2], DBListJoinTuple &tuples, uint outer)

selectJoinTupleIndexedNested()

```
uint inner = 1 - outer;

DBListTuple outerlist;
DBListTuple innerlist;

selectTuple(table[outer], where[outer], outerlist);

DBListTuple::iterator i = outerlist.begin();
while (i != outerlist.end()) {
  DBTuple &left = (*i);

  selectIndexedTuple(table[inner], where[inner], innerlist,
                     attrJoinPos[inner], left.getAttrVal(attrJoinPos[outer]));

  DBListTuple::iterator u = innerlist.begin();
  while (u != innerlist.end()) {
    DBTuple &right = (*u);
    LOG4CXX_DEBUG(logger, "left:\n" + left.toString("\t"));
    LOG4CXX_DEBUG(logger, "right:\n" + right.toString("\t"));
    pair<DBTuple, DBTuple> p;
    p.first = left;
    p.second = right;
    tuples.push_back(p);
    ++u;
  }
  ++i;
}
```

Finden von innerer/äußerer Relation

- **Finden von innerer und äußerer Relation:**

```
// Determine outer-inner relation
const DBRelDef &def0 = table[0]->getRelDef();
const DBRelDef &def1 = table[1]->getRelDef();
DBAttrDef adef0 = def0.attrDef(attrJoinPos[0]);
DBAttrDef adef1 = def0.attrDef(attrJoinPos[1]);
uint smaller = 0;
if (table[0]->getPageCnt() > table[1]->getPageCnt()) {
  smaller = 1;
}

bool indexed = false;
uint outer = 0;
if (adef0.isIndexed()) {
  indexed = true;
  if (adef1.isIndexed()) {
    if (smaller == 1) {
      outer = 1;
    }
  } else {
      outer = 1;
  }
} else if (adef1.isIndexed()) {
  indexed = true;
} else {
  if (smaller == 1) {
    outer = 1;
  }
}
```

Auswahl des Join-Algorithmus

- **Auswahl des Join-Algorithmus:**
  - Bisheriger Non-Indexed Nested Loop Join wenn keines der Join-Attribute indiziert ist
  - Indexed Nested Loop Join wenn indiziert

```
if (!indexed) {
  selectJoinTupleNested(table, attrJoinPos, where, tuples);
} else {
  selectJoinTupleIndexedNested(table, attrJoinPos, where, tuples, outer);
}
```