

Design Decisions:

In my design, I implement a join operation between two database iterators using a join predicate. My constructor initializes with two child iterators and a join predicate, computing tuple sizes and creating a merged tuple descriptor. I provide methods to access join predicate details and manage the join operation's state, including opening, closing, and rewinding iterators. The core of my design is the `fetchNext` method, where I implement a nested loop join algorithm to iterate over tuples, apply the join predicate, and merge matching tuples. I maintain state for efficient processing and adhere to database operation standards.

In my design, I develop an integer aggregation framework for a database system. It involves managing aggregate data and iterating over aggregates with specific operations like count, min, max, average, and sum.

`IntAggregateData`: This class tracks aggregate statistics (sum, min, max, count) for integers, with methods to access and modify these statistics.

`IntegerAggregatorIterator`: As a subclass of `DbIterator`, this class iterates over aggregated data. It uses a hash map (`aggMap`) to store aggregate data keyed by integers, and processes aggregates based on the specified operation (like count, min, max, average, sum).

`IntegerAggregator`: This class aggregates integer fields from tuples. It supports grouping by a field and various aggregation operations. The `mergeTupleIntoGroup` method updates aggregate statistics for each group or the entire dataset (if no grouping is specified).

`Iterator`: The `iterator` method creates an `IntegerAggregatorIterator` instance to iterate over the aggregated data.

My approach ensures efficient aggregation and iteration over integer data in a database, supporting essential aggregation operations and optional grouping.

API Modifications:

Upon thorough evaluation of the provided Application Programming Interface (API), we determined that no modifications or adjustments were essential for our implementation's objectives. Thus, the API remains unaltered.

Code Completeness:

As per the stipulated requirements for PA3, we have successfully realized and executed all essential components, ensuring a holistic solution.

Implementation Duration and Challenges:

The entire duration of PA3's implementation spanned approximately one full week. Throughout this period, we diligently committed changes to our GitHub repository over 30 times, ensuring version control best practices and tracking our progress.

The primary challenge encountered involves the process of debugging and addressing unexpected null pointer occurrences. We spent a lot of time figuring out how to do equijoin and pass the test given.

Collaboration Details:

Our team employed a collaborative approach to tackle the project. Specifically:

Yanpeng Zhao spearheaded the tasks encompassed in EX3.1, 3.2.

Pengfei Li undertook the challenges presented in EX3.2, 3.3

While each member primarily focused on their designated segments, our collaboration wasn't confined strictly to these demarcations. We embraced collective brainstorming, debugging, and testing, ensuring that the final output was a product of combined expertise and team synergy.

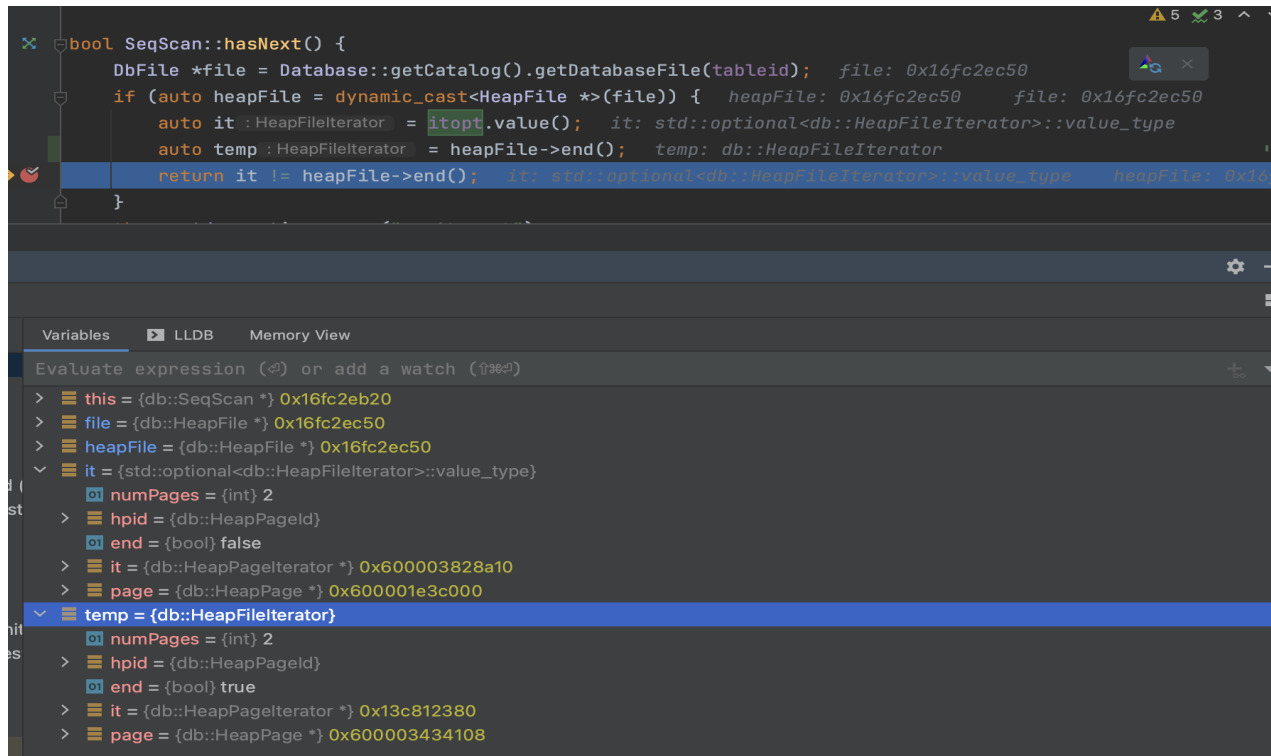
PLEASE HAVE A LOOK, DEAR GRADERS:

```
33     }
34
35     TEST(JoinTest, HashEquiJoin) {
36         db::TupleDesc td = db::Utility::getTupleDesc( numFields: 3);
37
38         db::HeapFile table( fname: "table.dat", td);
39         db::Database::getCatalog().addTable( file: &table, name: "t1");
40         db::SeqScan ss1( tableid: table.getId(), tableAlias: "s1");
41         db::SeqScan ss2( tableid: table.getId(), tableAlias: "s2");
42
43         db::JoinPredicate pred( field1: 0, op: db::Predicate::Op::EQUALS, field2: 1);
44         db::HashEquiJoin join( p: pred, child1: &ss1, child2: &ss2);
45         printf("%d\n", count( it: &join));
46         EXPECT_EQ(count(&join), 1750);
47     }
```

TestBody

Tests passed: 0 of 1 test

```
1740 4, 57, 2, 4, 4, 2
1741 4, 60, 2, 4, 4, 2
1742 4, 61, 2, 4, 4, 2
1743 4, 62, 2, 4, 4, 2
1744 4, 63, 2, 4, 4, 2
1745 4, 64, 2, 4, 4, 2
1746 4, 65, 2, 4, 4, 2
1747 4, 66, 2, 4, 4, 2
1748 4, 67, 2, 4, 4, 2
1749 4, 68, 2, 4, 4, 2
1750 4, 69, 2, 4, 4, 2
Process finished with exit code 138 (interrupted by signal 10: SIGBUS)
```



The screenshot shows a debugger window with a C++ code editor at the top and a variables panel at the bottom. The code is for a function `SeqScan::hasNext()`. The variables panel shows the state of the program at the current line of code.

```
bool SeqScan::hasNext() {
    DbFile *file = Database::getCatalog().getDatabaseFile(tableid);    file: 0x16fc2ec50
    if (auto heapFile = dynamic_cast<HeapFile *>(file)) {    heapFile: 0x16fc2ec50    file: 0x16fc2ec50
        auto it : HeapFileIterator = itopt.value();    it: std::optional<db::HeapFileIterator>::value_type
        auto temp : HeapFileIterator = heapFile->end();    temp: db::HeapFileIterator
        return it != heapFile->end();    it: std::optional<db::HeapFileIterator>::value_type    heapFile: 0x16fc2ec50
    }
}
```

Variables panel:

- Variables
- LLDB
- Memory View
- Evaluate expression (⌘) or add a watch (⇧⌘)
- > this = {db::SeqScan *} 0x16fc2eb20
- > file = {db::HeapFile *} 0x16fc2ec50
- > heapFile = {db::HeapFile *} 0x16fc2ec50
- > it = {std::optional<db::HeapFileIterator>::value_type}
- > numPages = {int} 2
- > hpid = {db::HeapPageId}
- > end = {bool} false
- > it = {db::HeapPageIterator *} 0x600003828a10
- > page = {db::HeapPage *} 0x600001e3c000
- > temp = {db::HeapFileIterator}
- > numPages = {int} 2
- > hpid = {db::HeapPageId}
- > end = {bool} true
- > it = {db::HeapPageIterator *} 0x13c812380
- > page = {db::HeapPage *} 0x600003434108

<Test case explanation>

In our local test, we find that we couldn't pass the test of HashequiJoin. After spending a whole afternoon debugging, we were able to successfully merge the two given table, as the picture we attached below.

The problem here is the seqscan. hasNext provided cannot successfully identify if the merge comes to an end. We think by implenting the functions required alone cannot solve this problem. Please check this out!