Build your own Database

Week 4
Agenda

• Q&A Sprint 2
• Review Sprint 1
• Experiments with Chunks
• std::optional
• Lambdas
Sprint 2

Questions?
Formatting / Linting

```
dyod_sprint git:(sprint1_group) gs
On branch: sprint1_group | No changes (working directory clean)
dyod_sprint git:(sprint1_group) ./scripts/format.sh
```
```
dyod_sprint git:(sprint1_group) x gs
On branch: sprint1_group | [*] => $e$

Changes not staged for commit

```
 modified:  1 src/test/storage/chunk_test.cpp
```
```
dyod_sprint git:(sprint1_group) x
```
```
→ dyod_sprint git:(sprint1_group) x ./scripts/lint.sh
```
```
src/test/storage/chunk_test.cpp:29: Add #include <memory> for Category 'build/include_what_you_use' errors found: 1
Total errors found: 1
src/test/storage/storage_manager_test.cpp:19: Add #include ...
] [4]
Category 'build/include_what_you_use' errors found: 1
Total errors found: 1
→ dyod_sprint git:(sprint1_group) x
```
std::vector<std::string> m_columnNames;
std::vector<std::string> m_columnTypes;
std::vector<Chunk> m_chunks;
unsigned int m_chunkSize;

this->chunk_size()
Clean Commits

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
<th>@@ -1,4 +1,4 @@</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>-#pragma once</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>+#pragma once</td>
</tr>
</tbody>
</table>

- **Dockerfile**
  - Dockerfile
  - +13 -0

- **playground.cpp**
  - src/bin/playground.cpp
  - +22 -0
For now, this is not incorrect, but in the future, it will not work any more.
std::vector<std::string> StorageManager::table_names() const {
    std::vector<std::string> names;
    auto get_name = [](const auto& entry) { return entry.first; };
    std::transform(m_tables.begin(), m_tables.end(), std::back_inserter(names), get_name);
    return names;
}
C++ things

Let’s play a different game – what did we like about this?

```cpp
std::vector<std::string> StorageManager::table_names() const {
    std::vector<std::string> names;
    names.reserve(m_tables.size());
    // [...]
    for (const auto& chunk : m_chunks) {
        count += chunk.size();
    }
}
```
Const
Experiments with Chunks
Experiments with Chunks

**Benefits**

- Chunks are stable once they are compressed
- Simplified data placement in general and especially on NUMA systems
- Enhanced query execution

**Drawbacks**

- Potentially increased memory consumption by duplicated metadata structures
Chunks – Non-Uniform Memory Access
Chunks – Non-Uniform Memory Access
Experiments with Chunks

Benefits

• Chunks are stable once they are compressed
• Simplified data placement in general and especially on NUMA systems
• Enhanced query execution

Drawbacks

• Potentially increased memory consumption by duplicated meta data structures
Chunks – Enhanced Query Execution

Chunk #1
- John
- Mary
- Frank
- Peter

Chunk #2
- Peter
- Hasso
- Ann
- Lisa

Chunk #3
- Theresa
- Donald
- Angela
- Peter

Meta Data
Chunks – Enhanced Query Execution

SELECT * FROM customers WHERE firstname = 'Hasso'
Chunks – Enhanced Query Execution

SELECT * FROM customers WHERE firstname = ‘Hasso’
SELECT * FROM customers WHERE firstname = 'Hasso'
SELECT * FROM customers WHERE firstname = 'Peter'
### Chunks – Enhanced Query Execution

**SELECT * FROM customers WHERE firstname = ‘Peter’**

<table>
<thead>
<tr>
<th>Chunk #1</th>
<th>Chunk #2</th>
<th>Chunk #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Peter</td>
<td>Theresa</td>
</tr>
<tr>
<td>Mary</td>
<td>Hasso</td>
<td>Donald</td>
</tr>
<tr>
<td>Frank</td>
<td>Ann</td>
<td>Angela</td>
</tr>
<tr>
<td>Peter</td>
<td>Lisa</td>
<td>Peter</td>
</tr>
</tbody>
</table>
Experiments with Chunks

Benefits

• Chunks are stable once they are compressed
• Simplified data placement in general and especially on NUMA systems
• Enhanced query execution

Drawbacks

• Potentially increased memory consumption by duplicated meta data structures
Experiments with Chunks

1. How does chunking affect the memory footprint?

2. If the last chunk is uncompressed, what is its effect on the total memory consumption?

3. What are performance implications of chunks?
   1. Considering single-threaded and multi-threaded (NUMA) execution
Experiments with Chunks

• Methodology:
  – Utilize real-world data from a productive SAP system
  – Extract actual queries from system’s plan cache
  – Load 100M rows of data into Opossum/Hyrise
  – Measure memory footprint/query performance
  – Repeat experiments for different chunk sizes
Experiments with Chunks: Memory Footprint

Table Size for Main/Delta and Different Chunk Sizes

- Main: 32.2 GB
- 100 Chunks: 34.4 GB
- 50 Chunks: 34.1 GB
- 20 Chunks: 33.3 GB
- 10 Chunks: 32.8 GB
- 5 Chunks: 32.7 GB
Experiments with Chunks: Memory Footprint

Normalized Memory Consumption for Different Chunk Sizes Compared to Main/Delta

Memory Consumption in Percent of Main/Delta Memory Consumption

Column Name

100 Chunks
50 Chunks
20 Chunks
10 Chunks
5 Chunks
Experiments with Chunks: Memory Footprint

Dictionary Size for Main/Delta and Different Chunk Sizes
Experiments with Chunks: Memory Footprint
Experiments with Chunks: Memory Footprint

Attribute Vector Width for Main/Delta and Different Chunk Sizes

Column Name

| PRCTR | RFAREA | RBUS | KOKRS | SEGMENT | SCNTR | PRCTR | SFAREA | SBUS | RASSC | PSIGMENT | TSL | HSL | KSL | OSL | MSL | WSL | VMSL | KFSL | FSL | PFSL | VAFSL | VAQNT1 | VAQNT2 | VAQNT3 | DRCRK | POPER | GJAHR | BUDAT | BELNR | BWEI | BSCHL | BSTAT | LINETYPE | XPLMOD | USNAM | TIMESTAMP | EPRCTR | RHOART | KTOPL | REBVZ | REBZ | REBZT | REBEST | REBEN | EBELP | ZEKNN | SGXT | MATNR |
|-------|--------|------|-------|---------|-------|-------|--------|------|-------|----------|-----|-----|-----|-----|-----|-----|------|------|-----|------|-------|--------|--------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|

- Main
- 100 Chunks
- 50 Chunks
- 20 Chunks
- 10 Chunks
- 5 Chunks
Experiments with Chunks: Memory Footprint

Delta/Last Chunk Size for Different Chunk Configurations

- Delta: 0.02 GB
- 1M: 0.70 GB
- 2M: 1.40 GB
- 5M: 3.49 GB
- 10M: 6.98 GB
- 20M: 13.97 GB
Experiments with Chunks: Performance (single-threaded)

Execution Time for Main/Delta and Different Chunk Sizes

- Main
- 100 Chunks
- 50 Chunks
- 20 Chunks
- 10 Chunks
- 5 Chunks

Queries:
- Query 1
- Query 2
- Query 3
- Query 4
Experiments with Chunks: Performance (single-threaded)
Experiments with Chunks: Performance (NUMA)
Optionals

• „Manages an optional contained value, i.e. a value that may or may not be present."

• Example use case: A table scan that supports between and, therefore, needs two search value parameters

• Syntax:

```cpp
#include <optional>

// Templated object of type std::optional<T>
std::optional<AllTypeVariant> opt;
std::optional<AllTypeVariant> opt2 = std::nullopt;
std::optional<AllTypeVariant> opt3 = 17;
if (opt) {
    do_something(*opt);
}
```
Optionals

Any ideas how to implement that?

```cpp
std::pair<T, bool>

template <typename T>
class optional {
    bool _initialized;
    T _storage;
};
```

What is the result of `sizeof(std::optional<uint32_t>)`?
Lambda Expressions

A simplified table scan...

```cpp
for (auto i = 0; i < value_column.size(); ++i) {
    switch (_scan_type) {
    case ScanType::OpEquals: {
        return value_column.get(i) == search_value;
        break;
    }
    case ScanType::OpNotEquals: {
        return value_column.get(i) != search_value;
        break;
    }
    case ScanType::OpLessThan: {
        return value_column.get(i) < search_value;
        break;
    }
    // [...]
```
Lambda Expressions

With lambda expressions

```cpp
auto comparator = get_comparator(_scan_type);
for (auto i = 0; i < value_column.size(); ++i) {
    return comparator(value_column.get(i), search_value);
}
```

```cpp
auto get_comparator(ScanType type) {
    switch (type) {
    case ScanType::OpEquals: {
        _return = [](auto left, auto right) { return left == right; };
        break;
    }
    case ScanType::OpNotEquals: {
        _return = [](auto left, auto right) { return left != right; };
        break;
    }
    // [...]
    }
}
```

+ separation of concerns
+ checks only once
+ reuse
Lambda Expressions

Syntax:

```cpp
auto f = [ captures ] ( params ) -> ret { body };
```

- **Parameters that are passed when the lambda is called**
- **Return value of the lambda** (if you leave it out, the compiler does it for you)
- **Code goes here**
- **Variables that you take from the current scope**
- **Can store lambdas in variables (and even members)**
- **You must use auto here**
- **Return value of the lambda** (if you leave it out, the compiler does it for you)
- **Code goes here**
- **Variables that you take from the current scope**
- **Can store lambdas in variables (and even members)**
- **You must use auto here**
Lambda Expressions

```cpp
auto f = [ captures ] ( params ) -> ret { body };

int main() {
    auto f = []() {
        std::cout << "Hallo Welt" << std::endl;
    };

    f();
}
```
Lambda Expressions

auto f = [ captures ] ( params ) -> ret { body };

int main() {
    auto f = [](const std::string& name) {
        std::cout << "Hallo " << name << std::endl;
    };

    f("Alexander");
}

auto f = [ captures ] ( params ) -> ret { body };

int main() {
    std::string my_name{"Larry"};

    auto f = [my_name](const std::string& name) {
        std::cout << "Hallo " << name << ", ich bin "
                   << my_name << std::endl;
    }

    f("Alexander");
}
Lambda Expressions

```cpp
auto f = [ captures ] ( params ) -> ret { body };

int main() {
    std::string my_name{"Larry"};

    auto f = [&my_name](const std::string& name) {
        std::cout << "Hallo " << name << ", ich bin "
                  << my_name << std::endl;
    };

    f("Alexander");
}
```
Lambda Expressions

```cpp
auto get_lambda() {
    std::string my_name{"Larry"};
    return [my_name]() {
        std::cout << "Ich bin " << my_name << std::endl;
    };
}

int main() {
    f = get_lambda();

    // my_name is undefined here

    f();
}
```
auto get_lambda() {
    std::string my_name="Larry";
    return [&my_name]() {
        std::cout << "Ich bin " << my_name << std::endl;
    };
}

int main() {
    f = get_lambda();
    // my_name is undefined here
    f();
}
Lambda Expressions

User code

```c++
int main()
{
    vector<X> v;

    // Add elements to the vector...
    int total = 0;
    int offset = 1;

    for_each(v.begin(), v.end(),
             [&total, offset](X& elem) { total += elem.getVal() + offset; });
    cout << total << endl;
}
```

From [https://blog.feabhas.com/2014/03/demystifying-c-lambdas/](https://blog.feabhas.com/2014/03/demystifying-c-lambdas/)
A great resource if you want to learn more about lambdas
Lambda Expressions

```cpp
for_each(v.begin(), v.end(),
    [&total, offset](X& elem) { total += elem.getVal() + offset; });

for_each(v.begin(), v.end(), _SomeCompilerGeneratedName_{total, offset});
```

Compiler generated (conceptual)

```cpp
class _SomeCompilerGeneratedName_
{
public:
    _SomeCompilerGeneratedName_(int& t, int o) : total_{t}, offset_{o} {} 
    void operator()(X& elem) const { total_ += elem.getVal() + offset_; }

private:
    int& total_;       // Context captured by reference
    int offset_;       // Context captured by value
};
```
Next Week

- Relational Algebra
- Operators
- Presentation of Sprint 3