Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Build your own Database

Week 9

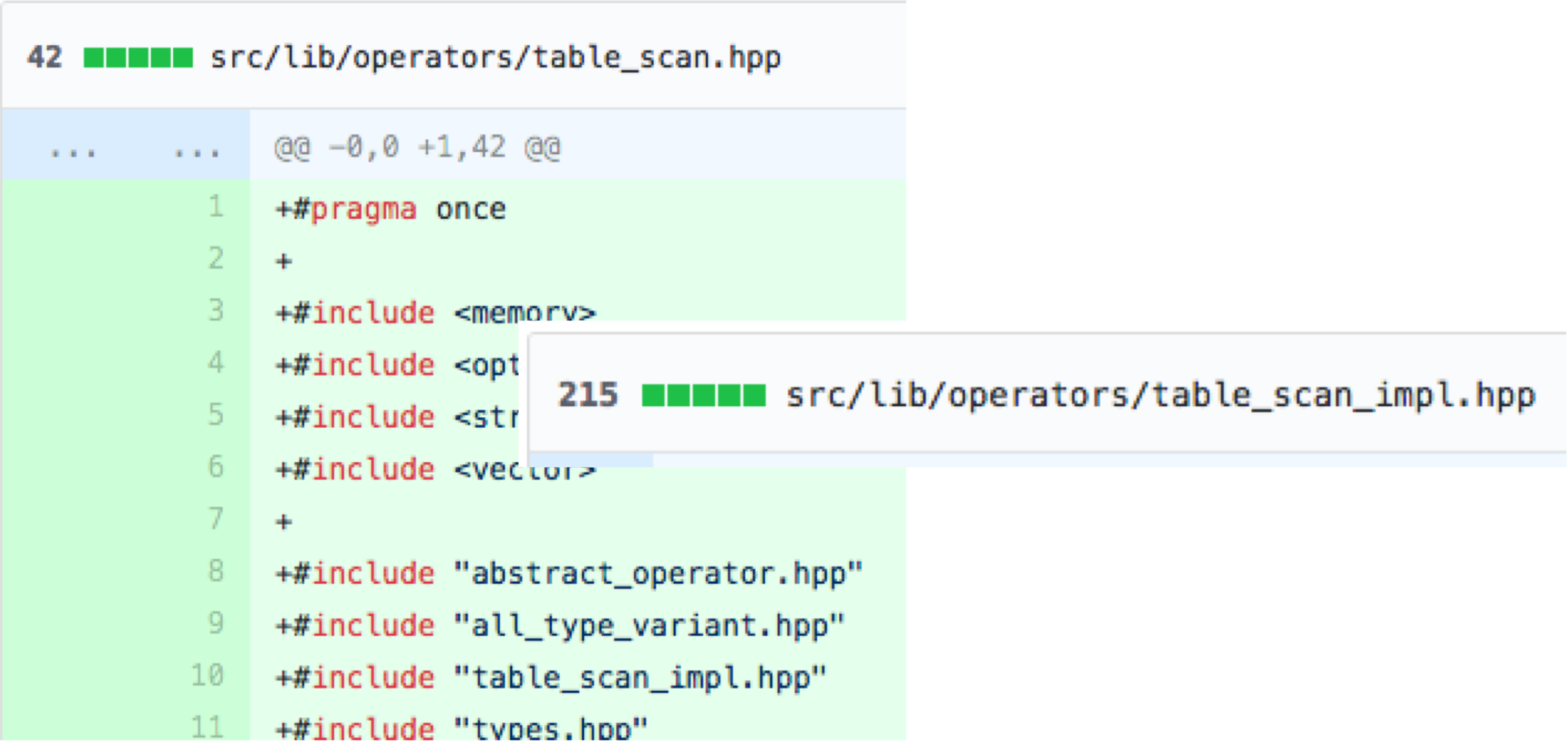
Agenda

- Review Sprint 3
- Logistics
- Midterm Presentation
- Group Meetings

Review Sprint 3

- Good Things First ☺
  - All implementations work
  - All are (with variations) easy to understand
  - Many placed code for internal classes in .cpp files
  - All groups have a shortcut if no matches exist
  - In total >100 new tests have been written

Review Sprint 3



```
42 ■■■■■ src/lib/operators/table_scan.hpp

...    ...    @@ -0,0 +1,42 @@
+#pragma once
+
+#include <memory>
+#include <opt
+#include <str
+#include <vector>
+
+#include "abstract_operator.hpp"
+#include "all_type_variant.hpp"
+#include "table_scan_impl.hpp"
+#include "types.hpp"
```

```
215 ■■■■■ src/lib/operators/table_scan_impl.hpp
```

Review Sprint 3

```
if (table->chunk_count() == 1) {
 // [...]
   if (reference_column) {
     // [...]
     return output_table;
   }
 }

 // [deal with value / dict column...]
```

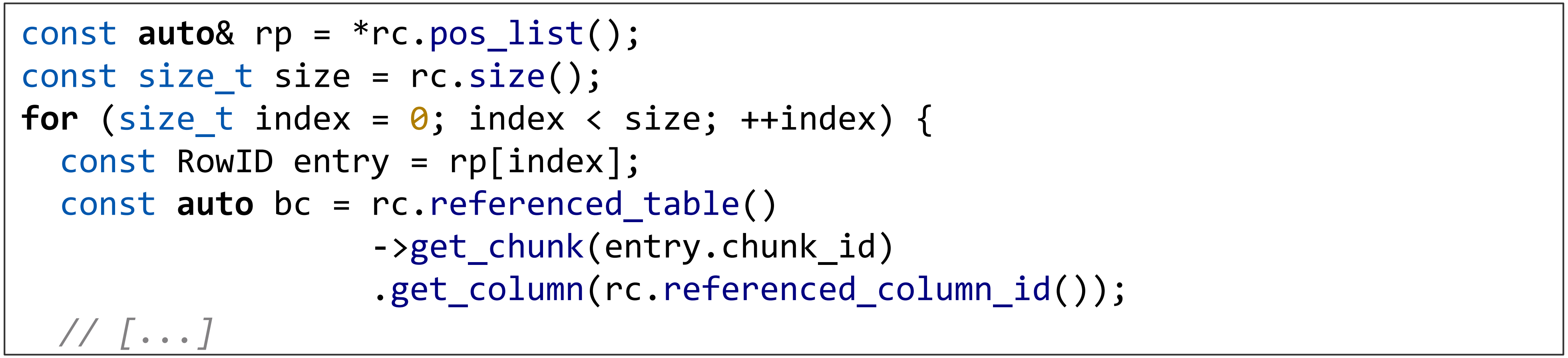
```
// We assume that the chunk_ids of positions are always in order.
```

Review Sprint 3

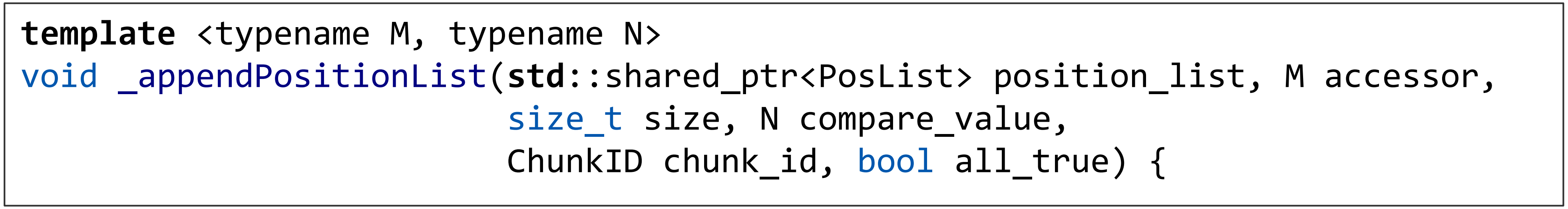
```
const std::vector<T>* ref_val_vector = nullptr;
// [...]
  ref_val_vector = ref_val_col ? &ref_val_col->values() : nullptr;
```

```
template <typename T>
void TableScanImpl<T>::_scanColumn(std::shared_ptr<PosList> position_list,
                                   std::shared_ptr<DictionaryColumn<T>> dc,
                                   const ChunkID chunk_id) {
```

Review Sprint 3



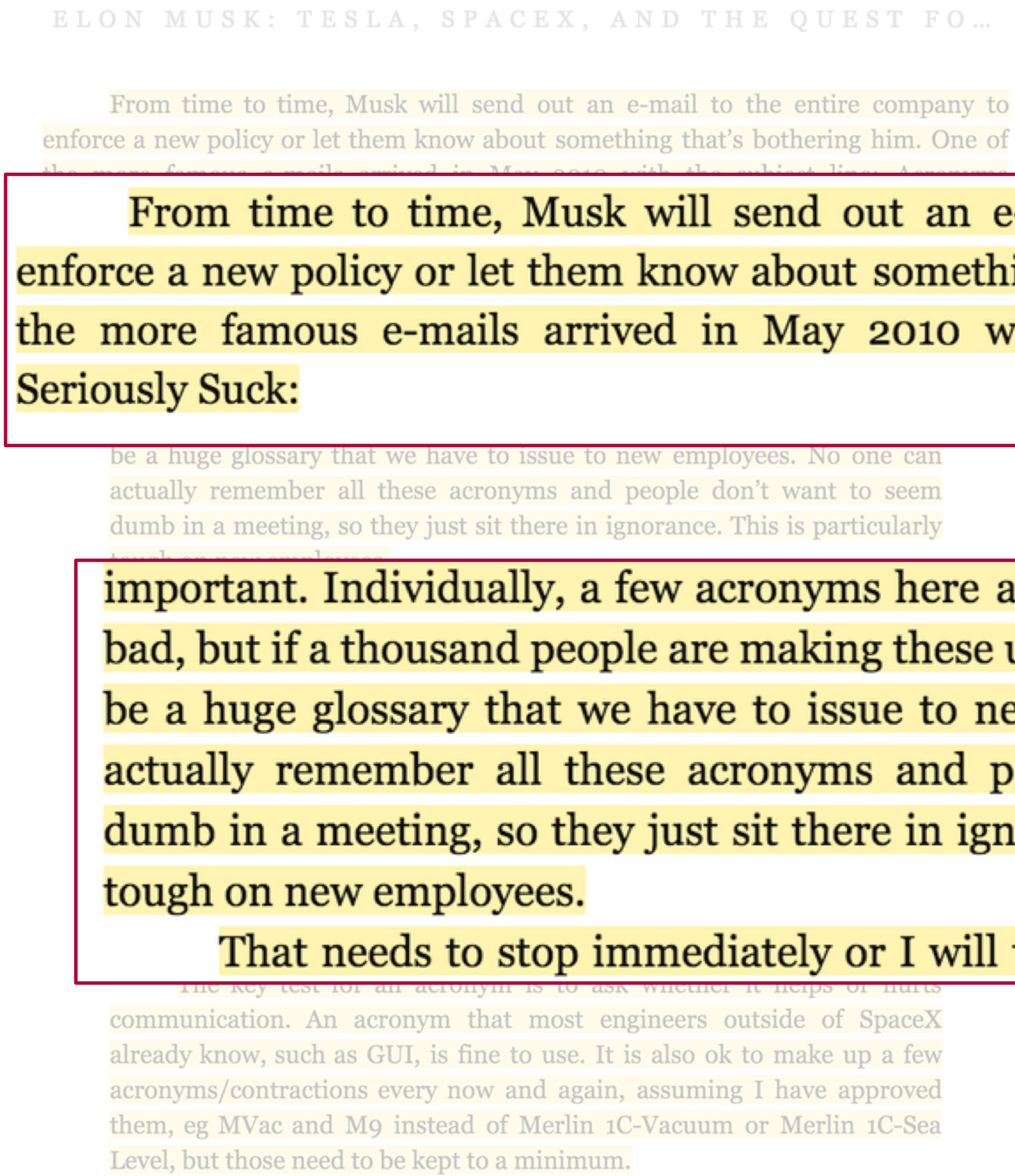
```
const auto& rp = *rc.pos_list();
const size_t size = rc.size();
for (size_t index = 0; index < size; ++index) {
  const RowID entry = rp[index];
  const auto bc = rc.referenced_table()
                  ->get_chunk(entry.chunk_id)
                  .get_column(rc.referenced_column_id());
  // [...]
```



```
template <typename M, typename N>
void _appendPositionList(std::shared_ptr<PosList> position_list, M accessor,
                         size_t size, N compare_value,
                         ChunkID chunk_id, bool all_true) {
```

From time to time, Musk will send out an e-mail to the entire company to enforce a new policy or let them know about something that's bothering him. One of the more famous e-mails arrived in May 2010 with the subject line: Acronyms Seriously Suck:

> There is a creeping tendency to use made up acronyms at SpaceX. Excessive use of made up acronyms is a significant impediment to communication and keeping communication good as we grow is incredibly important. Individually, a few acronyms here and there may not seem so bad, but if a thousand people are making these up, over time the result will be a huge glossary that we have to issue to new employees. No one can actually remember all these acronyms and people don't want to seem dumb in a meeting, so they just sit there in ignorance. This is particularly tough on new employees.
>
> That needs to stop immediately or I will take drastic action—I have given enough warnings over the years. Unless an acronym is approved by me, it should not enter the SpaceX glossary. If there is an existing acronym that cannot reasonably be justified, it should be eliminated, as I have requested in the past.
>
> For example, there should be no "HTS" [horizontal test stand] or "VTS" [vertical test stand] designations for test stands. Those are particularly dumb, as they contain unnecessary words. A "stand" at our test site is obviously a *test* stand. VTS-3 is four syllables compared with "Tripod," which is two, so the bloody acronym version actually takes longer to say than the name!
>
> The key test for an acronym is to ask whether it helps or hurts communication. An acronym that most engineers outside of SpaceX already know, such as GUI, is fine to use. It is also ok to make up a few acronyms/contractions every now and again, assuming I have approved them, eg MVac and M9 instead of Merlin 1C-Vacuum or Merlin 1C-Sea Level, but those need to be kept to a minimum.



ELON MUSK: TESLA, SPACEX, AND THE QUEST FO…

From time to time, Musk will send out an e-mail to the entire company to enforce a new policy or let them know about something that's bothering him. One of the more famous e-mails arrived in May 2010 with the subject line: Acronyms Seriously Suck:

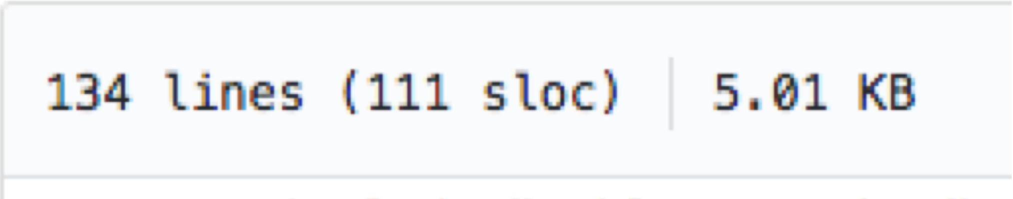
be a huge glossary that we have to issue to new employees. No one can actually remember all these acronyms and people don't want to seem dumb in a meeting, so they just sit there in ignorance. This is particularly

important. Individually, a few acronyms here and there may not seem so bad, but if a thousand people are making these up, over time the result will be a huge glossary that we have to issue to new employees. No one can actually remember all these acronyms and people don't want to seem dumb in a meeting, so they just sit there in ignorance. This is particularly tough on new employees.

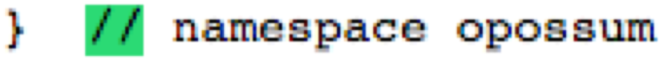
That needs to stop immediately or I will take drastic action—I have

The key test for an acronym is to ask whether it helps or hurts communication. An acronym that most engineers outside of SpaceX already know, such as GUI, is fine to use. It is also ok to make up a few acronyms/contractions every now and again, assuming I have approved them, eg MVac and M9 instead of Merlin 1C-Vacuum or Merlin 1C-Sea Level, but those need to be kept to a minimum.

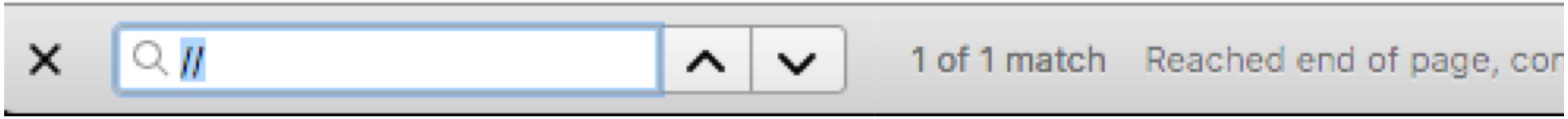
Review Sprint 3



134 lines (111 sloc) | 5.01 KB



```
} // namespace opossum
```

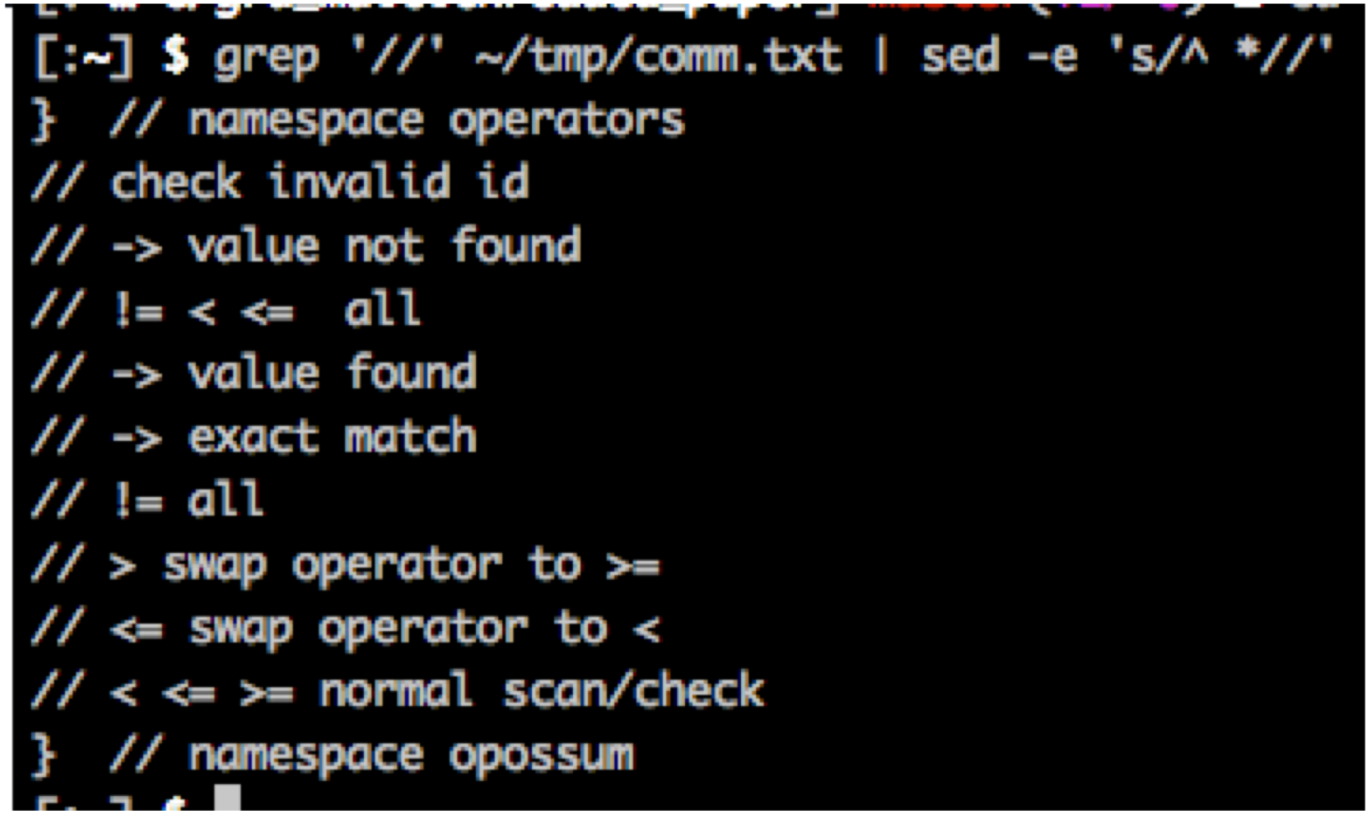


//

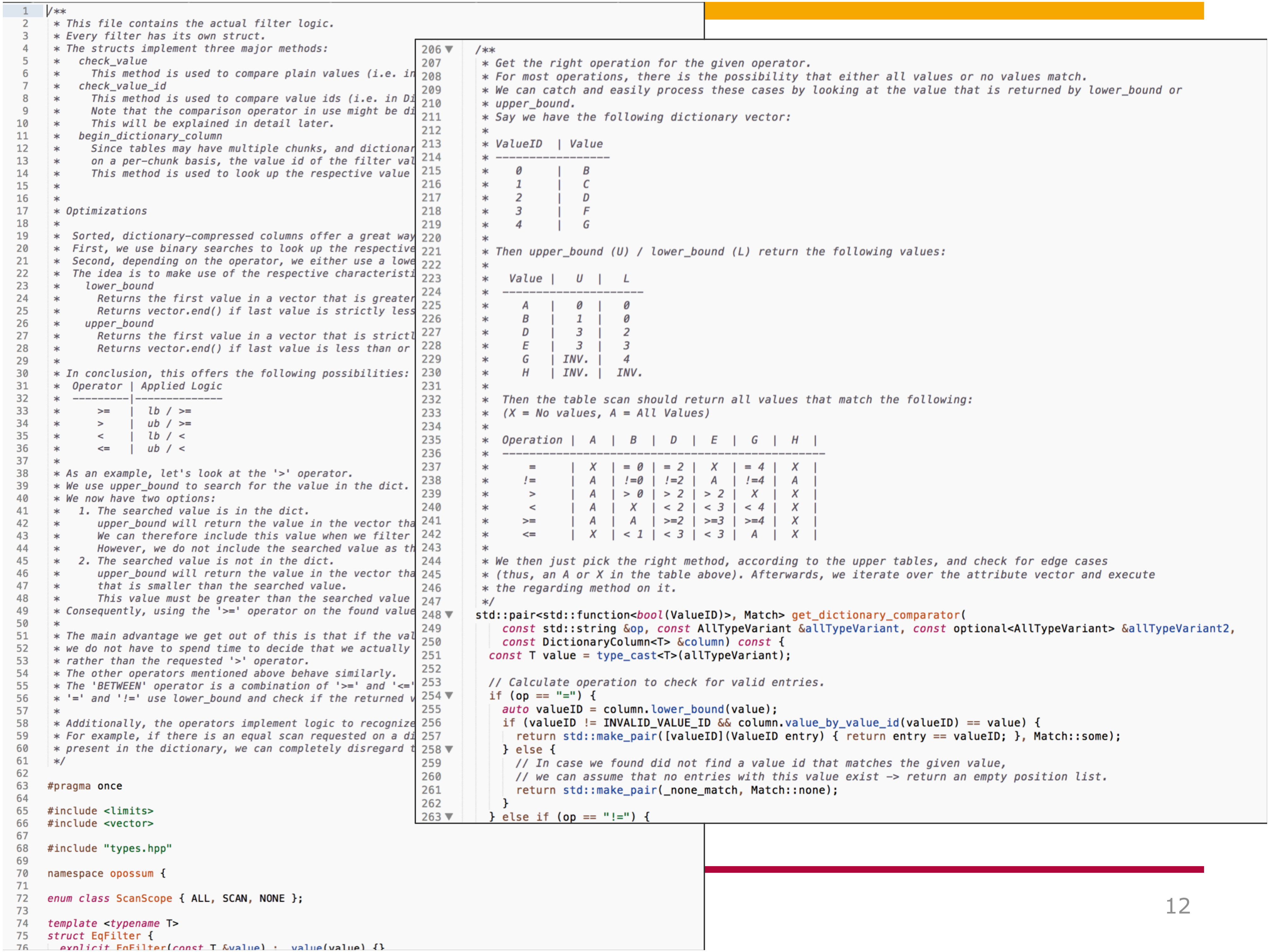
1 of 1 match   Reached end of page, cor

```
DebugAssert(dict_column, "Unknown Column Type in Table Scan");
// make sure the above is the case; only in debug mode
```

Review Sprint 3



```
[:~] $ grep '//' ~/tmp/comm.txt | sed -e 's/^ *//'
}  // namespace operators
// check invalid id
// -> value not found
// != < <=  all
// -> value found
// -> exact match
// != all
// > swap operator to >=
// <= swap operator to <
// < <= >= normal scan/check
}  // namespace opossum
```



```
/**
 * This file contains the actual filter logic.
 * Every filter has its own struct.
 * The structs implement three major methods:
 *   check_value
 *     This method is used to compare plain values (i.e. in
 *   check_value_id
 *     This method is used to compare value ids (i.e. in Di
 *     Note that the comparison operator in use might be di
 *     This will be explained in detail later.
 *   begin_dictionary_column
 *     Since tables may have multiple chunks, and dictionar
 *     on a per-chunk basis, the value id of the filter val
 *     This method is used to look up the respective value
 *
 *
 * Optimizations
 *
 *  Sorted, dictionary-compressed columns offer a great way
 *  First, we use binary searches to look up the respective
 *  Second, depending on the operator, we either use a lowe
 *  The idea is to make use of the respective characteristi
 *    lower_bound
 *      Returns the first value in a vector that is greater
 *      Returns vector.end() if last value is strictly less
 *    upper_bound
 *      Returns the first value in a vector that is strictl
 *      Returns vector.end() if last value is less than or
 *
 * In conclusion, this offers the following possibilities:
 *  Operator | Applied Logic
 *  ---------|--------------
 *      >=   |  lb / >=
 *      >    |  ub / >=
 *      <    |  lb / <
 *      <=   |  ub / <
 *
 * As an example, let's look at the '>' operator.
 * We use upper_bound to search for the value in the dict.
 * We now have two options:
 *   1. The searched value is in the dict.
 *      upper_bound will return the value in the vector tha
 *      We can therefore include this value when we filter
 *      However, we do not include the searched value as th
 *   2. The searched value is not in the dict.
 *      upper_bound will return the value in the vector tha
 *      that is smaller than the searched value.
 *      This value must be greater than the searched value
 * Consequently, using the '>=' operator on the found value
 *
 * The main advantage we get out of this is that if the val
 * we do not have to spend time to decide that we actually
 * rather than the requested '>' operator.
 * The other operators mentioned above behave similarly.
 * The 'BETWEEN' operator is a combination of '>=' and '<='
 * '=' and '!=' use lower_bound and check if the returned v
 *
 * Additionally, the operators implement logic to recognize
 * For example, if there is an equal scan requested on a di
 * present in the dictionary, we can completely disregard t
 */

#pragma once

#include <limits>
#include <vector>

#include "types.hpp"

namespace opossum {

enum class ScanScope { ALL, SCAN, NONE };

template <typename T>
struct EqFilter {
  explicit EqFilter(const T &value) :  value(value) {}
```

```
/**
 * Get the right operation for the given operator.
 * For most operations, there is the possibility that either all values or no values match.
 * We can catch and easily process these cases by looking at the value that is returned by lower_bound or
 * upper_bound.
 * Say we have the following dictionary vector:
 *
 * ValueID   | Value
 * -----------------
 *     0     |    B
 *     1     |    C
 *     2     |    D
 *     3     |    F
 *     4     |    G
 *
 * Then upper_bound (U) / lower_bound (L) return the following values:
 *
 *   Value |   U   |   L
 *  ---------------------
 *     A   |   0   |   0
 *     B   |   1   |   0
 *     D   |   3   |   2
 *     E   |   3   |   3
 *     G   | INV.  |   4
 *     H   | INV.  |  INV.
 *
 *  Then the table scan should return all values that match the following:
 *  (X = No values, A = All Values)
 *
 *  Operation |  A  |  B  |  D  |  E  |  G  |  H  |
 *  -----------------------------------------------
 *      =     |  X  | = 0 | = 2 |  X  | = 4 |  X  |
 *     !=     |  A  | !=0 | !=2 |  A  | !=4 |  A  |
 *      >     |  A  | > 0 | > 2 | > 2 |  X  |  X  |
 *      <     |  A  |  X  | < 2 | < 3 | < 4 |  X  |
 *     >=     |  A  |  A  | >=2 | >=3 | >=4 |  X  |
 *     <=     |  X  | < 1 | < 3 | < 3 |  A  |  X  |
 *
 * We then just pick the right method, according to the upper tables, and check for edge cases
 * (thus, an A or X in the table above). Afterwards, we iterate over the attribute vector and execute
 * the regarding method on it.
 */
std::pair<std::function<bool(ValueID)>, Match> get_dictionary_comparator(
    const std::string &op, const AllTypeVariant &allTypeVariant, const optional<AllTypeVariant> &allTypeVariant2,
    const DictionaryColumn<T> &column) const {
  const T value = type_cast<T>(allTypeVariant);

  // Calculate operation to check for valid entries.
  if (op == "=") {
    auto valueID = column.lower_bound(value);
    if (valueID != INVALID_VALUE_ID && column.value_by_value_id(valueID) == value) {
      return std::make_pair([valueID](ValueID entry) { return entry == valueID; }, Match::some);
    } else {
      // In case we found did not find a value id that matches the given value,
      // we can assume that no entries with this value exist -> return an empty position list.
      return std::make_pair(_none_match, Match::none);
    }
  } else if (op == "!=") {
```

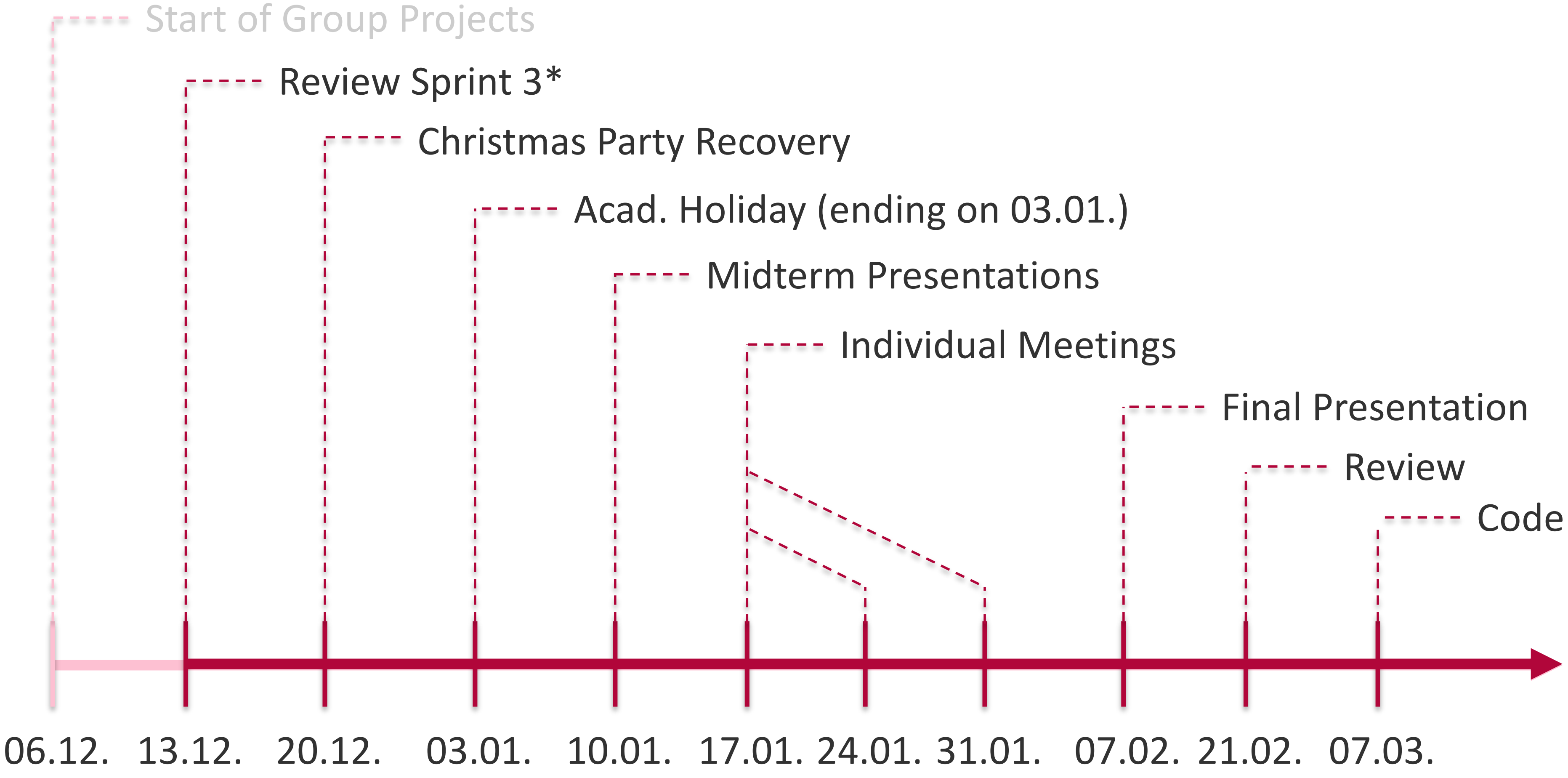
Review Sprint 3

```
auto value_column = std::dynamic_pointer_cast<ValueColumn<T>>(column);
if (value_column) {
  return _process_value_column(chunk_id, value_column);
}

auto dictionary_column =
std::dynamic_pointer_cast<DictionaryColumn<T>>(column);
if (dictionary_column) {
  return _process_dictionary_column(chunk_id, dictionary_column);
}

auto reference_column = std::dynamic_pointer_cast<ReferenceColumn>(column);
if (reference_column) {
  return _process_reference_column(chunk_id, reference_column);
}
```

Review Sprint 3

```
const auto result_table = std::make_shared<opossum::Table>();
Chunk chunk;
```

Logistics



Start of Group Projects

Review Sprint 3*

Christmas Party Recovery

Acad. Holiday (ending on 03.01.)

Midterm Presentations

Individual Meetings

Final Presentation

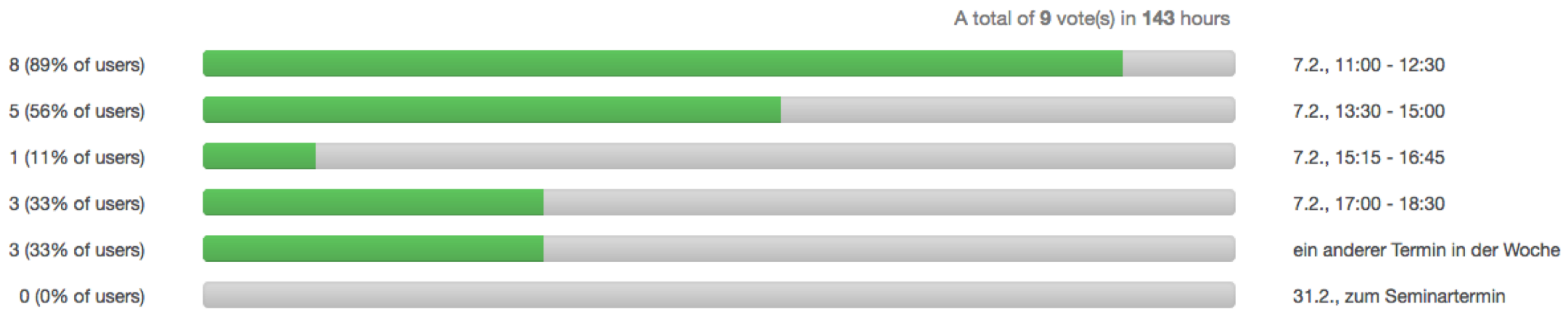
Review

Code

06.12. 13.12. 20.12. 03.01. 10.01. 17.01. 24.01. 31.01. 07.02. 21.02. 07.03.

*) For Sprint 3, we do not expect you to refactor your code

Logistics



A total of **9** vote(s) in **143** hours

8 (89% of users)	7.2., 11:00 - 12:30
5 (56% of users)	7.2., 13:30 - 15:00
1 (11% of users)	7.2., 15:15 - 16:45
3 (33% of users)	7.2., 17:00 - 18:30
3 (33% of users)	ein anderer Termin in der Woche
0 (0% of users)	31.2., zum Seminartermin

Midterm Presentations

- First meeting in the new year (10.01.)
- 5+2 minutes per group
  - What are you working on?
  - What design decisions did you make / will you have to make?
  - What is your biggest challenge?
- Please send us the slides before, so that we don’t have to switch laptops

Group Meetings

Group 1 (AH, AS, LW): Self-Driving	Glaskasten
Group 2 (LB, SD, RS): Networking	-
Group 3 (BF, MJ, TS): Data Types	Glaskasten
Group 4 (DH, PO, JW): Subqueries	V-2.16
Group 5 (AP, DS, ST): Pruning	here
Group 6 (JB, JN, FW): Joins	here
Group 7 (JC, NH, FM): Partitioning	-
Group 8 (FD, MF, TF): Optimizer Rules	V-2.16