Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Develop your own Database

Week 7

Agenda

- How was Sprint 3?
- NULL Values
- Logistics
- How to work with the Code Base
- Topic Assignments
- First Meeting for Group Project

Sprint 3

26.11.19, 21:53
26.11.19, 23:36
26.11.19, 23:38
26.11.19, 23:40
26.11.19, 23:59
00:00
01:15
01:16
01:22
02:25

- What issues did you encounter?
- Please remember to start your reviews
- Not necessary to include review comments in your code base

NULL values in SQL

- `NULL` is used to represent absent values
- It is a state/marker not a value
- DBMS dependent behavior in many cases
  - Arithmetic operations involving `NULL`, usually result in `NULL`: `NULL * 17` → `NULL`, but what about `NULL / 0`?
  - String concatenations involving `NULL`, result in `NULL`

NULL values in SQL

- `SELECT 17 = NULL` → ?
- SQL offers three logical results: `True, False, Unknown` → Three-valued Logic (3VL)

a	b	a AND b	a OR b	a = b
True	True	True	True	True
True	False	False	True	False
True	Unknown	Unknown	True	Unknown
False	True	False	True	False
False	False	False	False	True
False	Unknown	False	Unknown	Unknown
Unknown	True	Unknown	True	Unknown
Unknown	False	False	Unknown	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

NULL values in SQL

Customer

c_id	firstname	lastname
0	NULL	Zayer
1	Alex	Geier
2	Frank	Meier

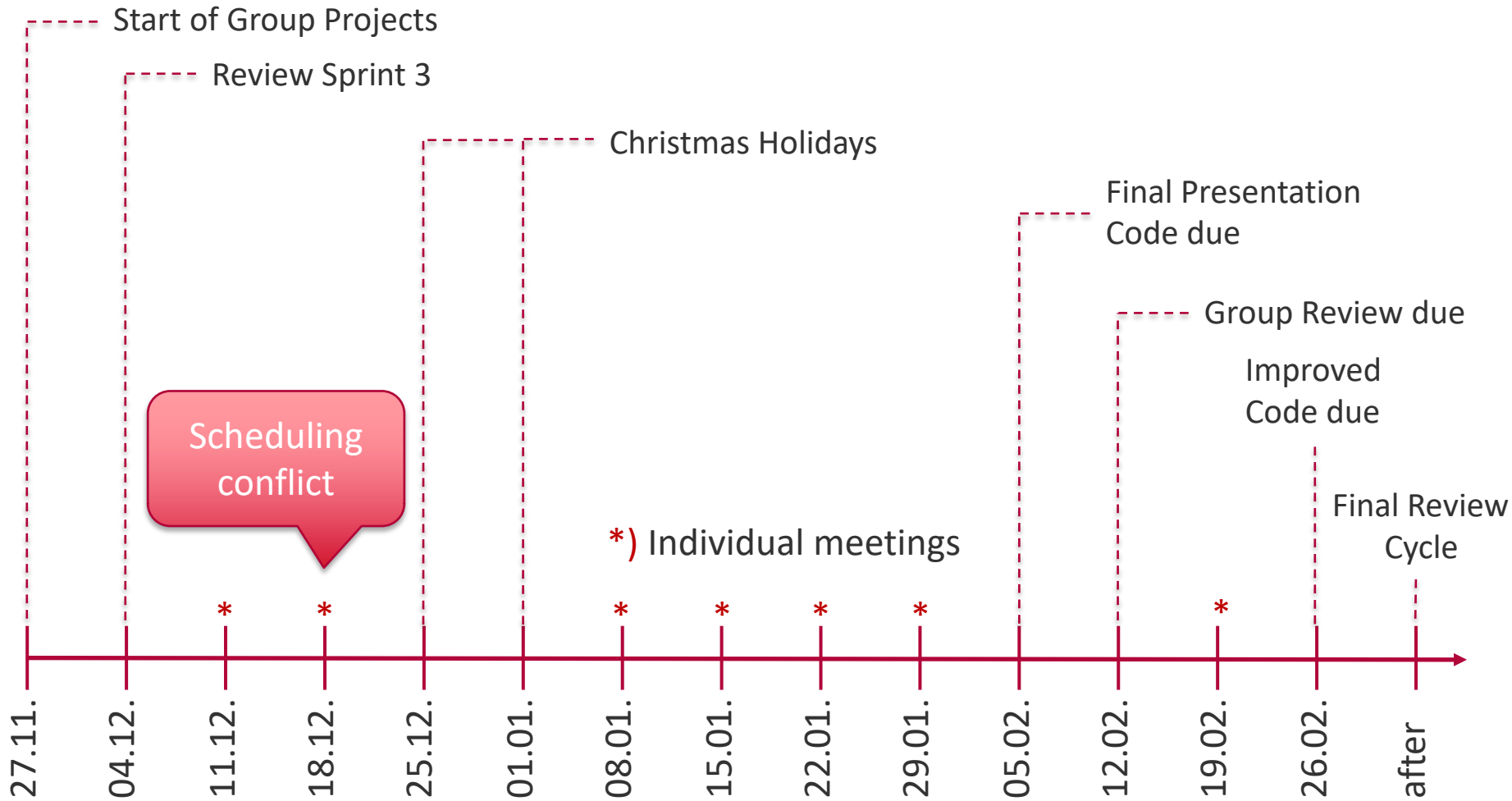
- `SELECT * FROM customer WHERE firstname = NULL;` → ?
- `SELECT * FROM customer WHERE firstname <> Alex;` → ?
- Rows for which predicates evaluate to `Unknown` are treated like rows that evaluate to `False`
- All standard comparison operators return `Unknown` when comparing `NULL`
- Thus, SQL offers `IS NULL` and `IS NOT NULL`

NULL values in Hyrise

We need a representation for NULL for Value, Reference and Dictionary columns:

```
// Value Segments

static const auto NULL_VALUE = AllTypeVariant{};

// Reference Segments

constexpr ChunkOffset INVALID_CHUNK_OFFSET{std::numeric_limits<ChunkOffset>::max()};

const RowID NULL_ROW_ID = RowID{ChunkID{0u}, INVALID_CHUNK_OFFSET};

// NULL ValueIDs for Dictionary Segments

constexpr ValueID NULL_VALUE_ID{std::numeric_limits<ValueID::base_type>::max()};

segment.unique_values_count()
```

Logistics


Start of Group Projects
Review Sprint 3
Christmas Holidays
Final Presentation
Code due
Group Review due
Improved
Code due
Final Review
Cycle
Scheduling
conflict
*) Individual meetings
27.11.
04.12.
11.12.
18.12.
25.12.
01.01.
08.01.
15.01.
22.01.
29.01.
05.02.
12.02.
19.02.
26.02.
after

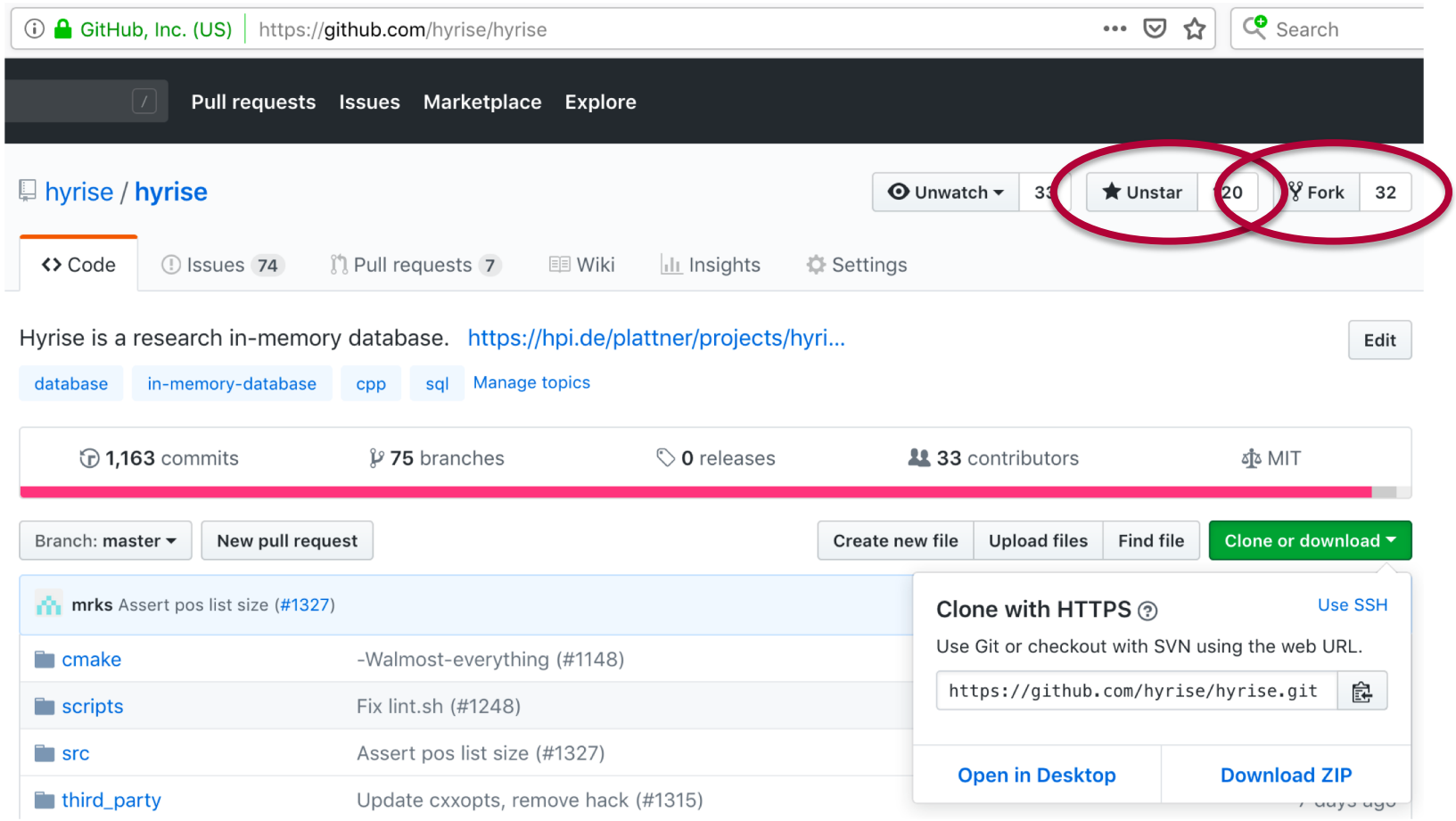
Date of Final Presentation

- Final day of instruction is on 05.02.
- We would like to have the final presentation on that day, but ~11 minutes per group is not enough
- Three options
  - Super Wednesday (05.02., 09:15 – 12:30)
  - Find a second slot in that week
  - Take the Wednesday before (our least favorite)

Setting up Hyrise

1. **Fork** the Hyrise Repo
2. Run ./install.sh
3. Setup a build folder
   - `mkdir build; cd build; cmake ..; make -jX`
   - Important cmake flags:
     - `-DCMAKE_CXX_COMPILER_LAUNCHER=ccache`
     - `-DCMAKE_BUILD_TYPE=Release (or RelWithDebInfo)`
     - `-DENABLE_NUMA_SUPPORT=Off`

Hyrise Code Base



Demo

Run Console

- `cd ..; ./build-release/hyriseConsole`
- `generate_tpch 0.1`
- `SELECT * FROM customer JOIN orders ON c_custkey = o_custkey WHERE o_orderpriority = '5-LOW' LIMIT 1`
- `visualize`

Run TPCH Benchmark

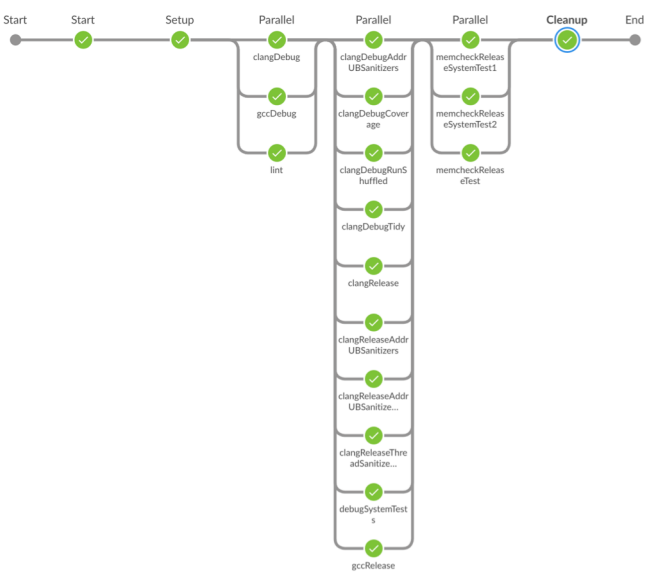
- `./build-release/hyriseBenchmarkTPCH –s 0.1 –visualize`
- Currently, Hyrise supports the TPC-H, -DS, -C, and JoinOrder Benchmark

Compare results

- `./scripts/compare_benchmarks.py old.json new.json`

Continuous Integration

- https://hyrise-ci.epic-hpi.de/
- Once you open your first PR, we will assign Hyrise team membership
- Only then will the CI start to build your code



- Once your code is stable and reviewed, you can add the *FullCI* tag, which enables additional CI verification

Pull Requests

- **Please open PRs from early on**
  - Early Feedback
  - Fewer Conflicts
  - Less Blocks
- The initial review will be done by *the other group*

Support

- **If you have any questions, do not wait for a week**
- You can use Piazza for questions or email us
- If anything looks odd or is hard to understand, please let us know

Hyrise Paper

- Our EDBT paper discusses some of the design decisions and the overall architecture
- Some parts have been covered in the lecture, others should be new
- https://tinyurl.com/HyrisePaper

Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management

Markus Dreseler, Jan Kossmann, Martin Boissier, Stefan Klauck,
Matthias Uflacker, Hasso Plattner
Hasso Plattner Institute
Potsdam, Germany
firstname.lastname@hpi.de

ABSTRACT

Research in data management profits when the performance evaluation is based not only on single components in isolation, but uses an actual DBMS end-to-end. Facilitating the integration and benchmarking of new concepts within a DBMS requires a simple setup process, well-documented code, and the possibility to execute both standard and custom benchmarks without tedious preparation. Fulfilling these requirements also makes it easy to reproduce the results later on.

The relational open-source database Hyrise (VLDB, 2010) was presented to make the case for hybrid row- and column-format data storage. Since then, it has evolved from a single-purpose research DBMS towards a platform for various projects, including research in the areas of indexing, data partitioning, and non-volatile memory. With a growing diversity of topics, we have found that the original code base grew to a point where new experimentation was made unnecessarily difficult. Over the last two years, we have rewritten Hyrise from scratch, focusing on building an extensible multi-purpose research DBMS that can serve as an easy-to-extend platform for a variety of experiments and prototyping in database research.

In this paper, we discuss how our learnings from the previous version of Hyrise have influenced our rewrite. We describe the new architecture of Hyrise and highlight the main components. We then show how our extensible plugin architecture facilitates research on diverse DBMS-related aspects without compromising the architectural tidiness of the code. In a first performance evaluation, we show that the execution time of most TPC-H queries is competitive to that of other research databases.

1 INTRODUCTION

Hyrise was first presented in 2010 [19] to introduce the concept of hybrid row- and column-based data layouts for in-memory databases. Since then, several other research efforts have used Hyrise as a basis for orthogonal research topics. This includes work on data tiering [7], secondary indexes [16], multi-version concurrency control [42], different replication schemes [43], and non-volatile memories for instant database recovery [44]. Over the years, the uncontrolled growth of code and functionality became an impediment for future experiments. We have identified four major factors leading to this situation:

- Data layout abstractions were resolved at runtime and incurred costs that sometimes had a disproportional overhead.
- Prototypical components have been implemented to work in isolation, but did not interact well with other components.
- The lack of SQL support required query plans to be written by hand and made executing standard benchmarks tedious.
- Accumulated technical debt made it difficult to understand the code base and to integrate new features.

For these reasons, we have rewritten Hyrise from scratch and incorporated the lessons learned. We redesigned the architecture to provide a stable and easy to use basis for holistic evaluations of new data management concepts. Hyrise now allows researchers to embed new concepts in a proper DBMS and evaluate performance end to end, instead of implementing and benchmarking them in isolation. At the same time, we allow most components to be selectively enabled or disabled. This way, researchers can exclude unrelated components and perform isolated measurements. For example, when developing a new join implementation, they can bypass the network layer or to disable concurrency control.

In this paper, we describe the new architecture of Hyrise and how our prior learnings have led to a maintainable and comprehensible database for researching concepts in relational in-memory data management (Section 2). Furthermore, we present a plugin concept that allows to test-drive different optimizations without having to modify the core DBMS (Section 3). We compare Hyrise to other database engines, show which approaches are similar, and highlight key differences (Section 4). Finally, we evaluate the new version and show that its performance is competitive (Section 5).

1.1 Motivation and Lessons Learned

The redesign of Hyrise reflects our past experiences in developing, maintaining, and using a DBMS for research purposes. We motivate three important design decisions.

Decoupling of Operators and Storage Layouts. The previous version of Hyrise was designed with a high level of flexibility in the storage layout model: each table could consist of an arbitrary number of containers, which could either hold data (in uncompressed or compressed, mutable or immutable forms) or other containers with varying horizontal and vertical spans. In consequence, each operator had to be implemented in a way where it could deal with all possible combinations of storage containers. This made the process of adding new operators cumbersome and led to a system where some operators made undocumented assumptions about the data layout (e.g., that all partitions used the same encoding type). Instead of relying on operators to properly process data structures with varying memory layouts, Hyrise now follows an iterator-based approach. By accessing data through iterators, the implementation of new operators is decoupled from the implementation of new data storage concepts without compromising the flexibility. Operators can implement custom specializations for specific iterators, but execution falls back to the default iterator if no implementation exists. The iterator abstraction is explained in Section 2.3.

Topic Assignments

• Gruppe 1 (LP, CT, JS):	Cache Parametrization (JK)	Glaskasten
• Gruppe 2 (LB, LR, FS):	Cost Model Calibration (MB)	V-1.16
• Gruppe 3 (LE, TL, TN):	Set Operations (MD)	Konfi
• Gruppe 4 (RE, LG, DM):	Smart Position Lists (JK)	Glaskasten
• Gruppe 5 (LF, BK, JN):	Sort Operator (MD)	Konfi
• Gruppe 6 (II, YK, TL):	Server (SH)	here
• Gruppe 7 (CG, HT, JT):	Exploit Sorting (MB)	V-1.16
• Gruppe 8 (MS, TZ):	Statistics Generation (TB)	here