Magnus Brevik & Lisa Willa

BOOLEAN data type in MySQL

Bachelor's project in Computer Engineering
Supervisor: Tore Mallaug
May 2021

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

ORACLE

Magnus Brevik & Lisa Willa

BOOLEAN data type in MySQL

Bachelor's project in Computer Engineering
Supervisor: Tore Mallaug
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Preamble

Preface

This report is the culmination of five months of work implementing a BOOLEAN data type in MySQL for Oracle Norge AS. The task has given us insight and experience with integrating into a large, preexisting system, with C++, MySQL, Agile methodology, and working with industry professionals.

We chose this project because we are familiar with MySQL, and are interested in backend, server and database programming. It was also enticing to get to work with professional developers on such a large and complex system.

The project has been filled with a lot of new challenges especially in terms of code comprehension. The lack of understanding and familiarity with the system and code forced us to work differently than we have in previous projects. We utilized pair programming, and learned to adapt to coding styles and otherwise conform to a preexisting codebase.

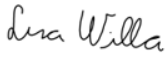
Thanks

First, we want to thank Oracle, for the task, equipment, and an interesting look into the employment and onboarding process into a large multinational company. It has been an experience to see how it is on the inside of such a large machinery. In Oracle we have been working with the MySQL Optimizer Team, who have been helpful and always happy to answer questions. We have especially enjoyed the weekly social meetings, where we have talked about anything from work to hobbies and politics. Our main contacts in the MySQL Optimizer Team have been Norvald Ryeng and Roy Lyseng. They know the relevant parts of the project well, and have been indispensable whenever we got stuck or needed guidance. We never would have gotten close to finishing without their regular feedback and system knowledge.

We want to thank our supervisor, Tore Mallaug, for proof reading, general advice for writing a bachelor's thesis, and being able to meet with us on short notice.

At last we want to thank our families, for proof-reading and giving feedback on the readability and general structure of the report. Without them, this report would have been riddled with small typos, wrong punctuation, and weird phrasings.

Magnus Brevik



Lisa Willa

Trondheim 20.05.2021

Task Description

The description in section and section is the original task description from Oracle.

Purpose of the Task

The purpose is to extend MySQL Server with SQL standard feature T031 BOOLEAN data type, enabling the MySQL Server to use BOOLEAN as a data type for columns in tables.

Description

While Boolean values, expressions and operators of course are part of the base SQL standard, the BOOLEAN data type is an optional extension, specified as feature T031. MySQL Server currently doesn't support this extension.

Today, MySQL users storing Boolean values in tables have to store them as INTEGER with false and true typically represented as 0 and 1, ENUM('FALSE','TRUE'), or CHAR(1) with false and true typically represented as 'T' and 'F', or 'Y' and 'N'. All these methods work and are functional replacements, but they all have their shortcomings

This task is to add SQL standard feature T031 to MySQL Server and involves the design, implementation and testing of the feature.

All code must be contributed under the Oracle Contributor Agreement (OCA): https://www.oracle.com/technetwork/community/oca-486395.html

Requirements

Requirements for the completed features can be found in the internal worklog WL#3554 (attachment 7.B) and in communication between the students and Oracle. The requirements are also specified in the vision document (attachment 7.A).

Summary

The boolean data type is a standard data type in most computer languages, taking the values `true` and `false`. This is represented in several ways, from Java using `true` and `false`, to C and most SQLs using 1 and 0, and some languages not having explicit booleans at all. The standard for BOOLEAN data type in SQL is directed by ISO/IEC 9075-2:2016, which also states that any standard compliant SQL should have a BOOLEAN data type [7].

In MySQL there is currently no BOOLEAN data type. Instead there is the word "BOOLEAN" which is an alias for TINYINT(1). This means that a column created with the word BOOLEAN is able to store any integer value that can be stored in one byte. This is a big ISO standard compliance issue, which leads to MySQL not being considered a "true" SQL. By not having completely standard compliant SQL Oracle risks being overlooked when the choice of database is made and therefore losing sales and reputation.

Our task has been to implement a BOOLEAN data type in the preexisting system of MySQL, so that it follows Oracle's vision for a BOOLEAN. The task came with a list of expected features and deviances from the ISO standard, some to be amended, and some to be left as is.

Throughout the development process we have used agile inspired development tactics such as pair programming, frequent customer collaboration, and standup meetings to build a working product that satisfies as many of the requirements as possible in the allotted time frame. To be able to implement a new feature in such a large preexisting code base, we had to spend many hours reading, discussing, and analyzing the code to understand which changes were needed and which were unnecessary.

The resulting feature is a BOOLEAN data type integrated in MySQL, along with new and updated tests for the MySQL Test Suites. The feature is not fully standard compliant, and misses a couple of the requirements from Oracle, but the final product works, and is close to a finished product that can be implemented in the official version of MySQL.

Further work would include implementing a way to explicitly cast between BOOLEAN and other data types with the CAST-function, updating the NDB Cluster storage engine to allow for the BOOLEAN data type, and implementing the word UNKNOWN as an alias for NULL in BOOLEAN context. These features have been omitted due to time constraints.

To get access to internal documents we had to be employed and onboarded in the MySQL Optimizer Team at Oracle Norge AS. This included several "new hire"-courses on anything from office safety to insider trading and workplace harassment.

Contents

List of Figures

Chapter 1

Introduction and Relevance

The open-source relational database management system MySQL is maintained by Oracle Corporation. MySQL databases are widely used and popular [1], ranking as the most popular database system according to Stack Overflows survey in 2020 [13], and hosting is provided by several services. Despite MySQL's wide popularity it also has its critics. One of the biggest criticisms is that it fails to follow the ISO standard for SQL. MySQL, like all other databases using SQL, is not entirely compliant with this standard. The BOOLEAN datatype is implemented in nothing but name, and is the source of many compliance issues for MySQL. These are listed in section 2.1.1.

This project seeks to remedy these compliance issues by implementing the BOOLEAN data type, and trying to make it more standard compliant. However, Oracle has to make sure it is backwards compatible to some extent, and wants it implemented as an integer type. This means that the implementation of the BOOLEAN datatype, as Oracle wants it, will not be completely standard compliant. This report will study the level of standard compliance that can be achieved while complying with Oracle's vision.

1.1 Thesis/Problem statement

To what extent will a BOOLEAN datatype implemented into MySQL, developed in accordance with Oracle's wishes, comply with the ISO standard?

1.2 Structure

This report will first go through relevant literature and theory in chapter 2. Choices of technology and methodology will be discussed in chapter 3. Results can be found in chapter 4, while the discussion of these can be found in chapter 5. Chapter 6 gives a conclusion, and recommendations for further work.

1.3 Glossary

- **Code comprehension:** Level of code understanding [20].
- **Falsy:** a value is *falsy* if it is evaluated as *false* in a boolean context. The set of falsy values is different between languages, but often includes `false`, 0, empty string, null.
- **ISO:** International Organization for Standardisation. An organization responsible for creating international standards also for SQL [6].
- **MySQL:** An open-source RDBMS, that uses SQL.
- **MySQL Optimizer team:** The team within Oracle that works with developing and optimizing MySQL.
- **Oracle:** refers to Oracle Corporation, a multinational company specializing in storage technology.
- **Oracle Norge AS:** refers to Oracle Corporation's branch in Norway.
- **RDBMS:** Relational DataBase Management System. Systems that handle relational databases.
- **Relational database:** Database that can store data in relation to each other. Data is usually organized in tables.
- **SQL:** Structured Query Language. Scripting language used to communicate with SQL databases.
- **Storage engine:** The underlying software handling the low-level allocation of memory and storage of data in a database
- **Truthy:** a value is *truthy* if it is evaluated as *true* in a boolean context. The set of truthy values is different between languages, but often includes `true`, any non-zero number, any non-empty string, any non-null object.

Chapter 2

Theory

2.1 Technical theory

2.1.1 ISO/IEC 9075-2:2016 Information technology — Database languages — SQL — Part 2: Foundation (SQL/-Foundation)

ISO/IEC

The International Organization for Standardisation (ISO) is an international organization responsible for creating and developing a myriad of international standards [6], and the International Electrotechnical Commission (IEC), provides standards for electrical, electronic and related technologies. Their joint efforts in "ISO and IEC Joint Technical Committee (JTC 1) for information technology" are responsible for several important standards [8], among them is the ISO/IEC JTC 1/SC 32 standard "ISO/IEC 9075-2:2016 Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)". This is the standard that details several important aspects of the SQL standard, including how the BOOLEAN data type should behave to comply with the SQL standard [7].

BOOLEAN data type in SQL according to ISO/IEC standard

- The data type is named BOOLEAN
- The BOOLEAN data type can have the values TRUE, FALSE and UNKNOWN/NULL
- BOOLEAN values are comparable and TRUE is greater than FALSE, comparisons with UNKNOWN/NULL will give UNKNOWN/NULL
- Boolean operators NOT, AND and OR can take BOOLEAN data type as operand

- BOOLEAN data type has the same values as expressions that return TRUE, FALSE, UNKNOWN/NULL
- UNKNOWN and NULL can be used interchangeably

[7]

These features are not allowed according to the ISO/IEC standard unless the BOOLEAN data type is implemented [7]:

- The reserved words TRUE, FALSE and UNKNOWN
- A data type called BOOLEAN
- A boolean value expression
- The boolean operators IS, NOT, OR or AND being used on value expressions that are not boolean value expression
- A value expression with no parenthesis interpreted as a boolean value
- A boolean column
- Having the operators EVERY, ANY or SOME

2.1.2 Boolean data type

The boolean data type, often shortened to bool, is a binary data type that can only have two values: true, and false. These values are usually represented as the numbers 1 and 0, or just the words "true" and "false". In this report they are mostly referred to as 1 and 0. Some form of boolean data type is present in every language made for a computer to understand, as computers work exclusively with binary data. With the boolean data type follows a set of rules for working with booleans, based on the rules of boolean algebra.

Boolean algebra

Boolean algebra is a field of mathematics with a small set of values and operations. The allowed values are 0 and 1, and the three standard operations are AND, OR and NOT. NOT inverts the value, i.e. 0 becomes 1 and 1 becomes 0. The truth tables of AND and OR are displayed in Figure 2.1 [19].

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

Figure 2.1: Truth tables for binary AND and OR [19]

AND	0	NULL	1
0	0	0	0
NULL	0	NULL	NULL
1	0	NULL	1

OR	0	NULL	1
0	0	NULL	1
NULL	NULL	NULL	1
1	1	1	1

Figure 2.2: Truth tables for three-valued AND and OR [19]

Three-Value logic

Three-value logic is an extension of boolean logic that also allows the value UNKNOWN, or NULL. NULL can be treated as either both 0 and 1 or neither 0 nor 1. The relevant version here is where NULL is neither value. This gives the extended truth tables displayed in Figure 2.2 [19].

2.1.3 TINYINT(1)

In MySQL the word BOOLEAN is currently an alias for TINYINT(1). TINYINT is an 8 bit integer type, storing values between -128 and 127 (signed) or 0 and 255 (unsigned), both inclusive. The (1) means a display width of 1, which affects the padding of the output when getting TINYINT(1) values from a table [14].

Currently, if you create a table with `CREATE TABLE t (b BOOLEAN);` and show table information with `SHOW CREATE TABLE t;` you get Figure 2.3 showing that BOOLEAN is an alias for TINYINT(1) in MySQL.

```
CREATE TABLE `t` (
  `b` tinyint(1) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 2.3: Result from `SHOW CREATE TABLE` on a column created with the `BOOLEAN` word

2.1.4 Boolean literals, words and expressions in MySQL

Boolean literals, words and expressions already implemented in MySQL include

- MySQL contains the boolean literals TRUE and FALSE as aliases for 1 and 0, see Figure 2.4.
- MySQL contains the word BOOLEAN as an alias for TINYINT(1). This is not a boolean data type.
- MySQL contains boolean value expressions, e.g. `IS [NOT] <boolean literal>` and similar. These evaluate to 0, 1, or NULL. See Figure 2.5.
- In a boolean context, i.e. when a boolean value is expected, MySQL treats any non-zero number as TRUE, and anything else as FALSE. See Figure 2.5
- MySQL contains the word UNKNOWN in the context of `IS [NOT] UNKNOWN`. UNKNOWN is not an alias for NULL and has to be used in the context of IS. See Figure 2.6.
- MySQL contains the words NOT, AND and OR, treating operands as truthy or falsy before comparing them. The result from the *main.bool* test contains a showcase of the these words. See Figure 2.7
- MySQL contains the words ALL, ANY, SOME, treating the operands as truthy or falsy before checking them.

```
mysql> SELECT TRUE, FALSE;
+------+-------+
| TRUE | FALSE |
+------+-------+
|    1 |     0 |
+------+-------+
```

Figure 2.4: TRUE and FALSE are aliases for 1 and 0

```
mysql> SELECT (1 IS FALSE) AS a, (2 IS TRUE) AS b, (0 IS NOT TRUE) as c,
    (0.1 IS FALSE) as d;
+---+---+---+---+
| a | b | c | d |
+---+---+---+---+
| 0 | 1 | 1 | 0 |
+---+---+---+---+
```

Figure 2.5: Shows boolean value expressions

```
mysql> SELECT UNKNOWN;
ERROR 1054 (42S22): Unknown column 'UNKNOWN' in 'field list'
mysql> SELECT NULL IS UNKNOWN;
+-----------------+
| NULL IS UNKNOWN |
+-----------------+
|               1 |
+-----------------+
```

Figure 2.6: Shows function of `UNKNOWN` in MySQL

2.2 Methodology

2.2.1 Enhancement projects and code comprehension

An enhancement project is a project that adds a feature to an already existing system, while a maintenance project is a project that fixes bugs or improves pre-existing functionality in a system. The terms are however used interchangeably [12]. Both enhancement projects and maintenance projects differ from development projects that create a new system. One of the most significant differences between the different types of projects is that maintenance/enhancement projects spend significantly more time on code comprehension than development projects do [20].

Code comprehension is the activity of trying to understand code, and includes reading documentation, reading source code and asking colleagues among other activities. It is shown that the average developer uses around 58% of their time on code comprehension activities [20]. For novice developers, the percentage of time spent on code comprehension activities is significantly higher and lies between 55% and 80% of their working hours. Even experienced developers, who are new to the codebase, will spend more than the 58% on code comprehension [20].

Since newcomers to codebases spend so much of their time on code compre-

```
mysql> CREATE TABLE t (a INT, b INT);
mysql> INSERT INTO t VALUES(NULL, NULL), (0, NULL), (1, NULL), (NULL,
    0), (NULL, 1), (0, 0), (0, 1), (1, 0), (1, 1);
mysql> SELECT IFNULL(a, 'N') AS A, IFNULL(b, 'N') AS B, IFNULL(NOT a,
    'N') AS nA, IFNULL(NOT b, 'N') AS nB, IFNULL(a AND b, 'N') AS AB,
    IFNULL(NOT (a AND b), 'N') AS `n(AB)`, IFNULL((NOT a OR NOT b),
    'N') AS nAonB, IFNULL(a OR b, 'N') AS AoB, IFNULL(NOT(a OR b), 'N')
    AS `n(AoB)`, IFNULL(NOT a AND NOT b, 'N') AS nAnB FROM t;
+---+---+----+----+----+-------+-------+-----+--------+------+
| A | B | nA | nB | AB | n(AB) | nAonB | AoB | n(AoB) | nAnB |
+---+---+----+----+----+-------+-------+-----+--------+------+
| N | N | N  | N  | N  | N     | N     | N   | N      | N    |
| 0 | N | 1  | N  | 0  | 1     | 1     | N   | N      | N    |
| 1 | N | 0  | N  | N  | N     | N     | 1   | 0      | 0    |
| N | 0 | N  | 1  | 0  | 1     | 1     | N   | N      | N    |
| N | 1 | N  | 0  | N  | N     | N     | 1   | 0      | 0    |
| 0 | 0 | 1  | 1  | 0  | 1     | 1     | 0   | 1      | 1    |
| 0 | 1 | 1  | 0  | 0  | 1     | 1     | 1   | 0      | 0    |
| 1 | 0 | 0  | 1  | 0  | 1     | 1     | 1   | 0      | 0    |
| 1 | 1 | 0  | 0  | 1  | 0     | 0     | 1   | 0      | 0    |
+---+---+----+----+----+-------+-------+-----+--------+------+
```

Figure 2.7: Example from the *main.bool* test [17]

hension it is common to have an onboarding process in which the newcomer is introduced to the unfamiliar codebase by an expert[21]. This reduces the time spent on code comprehension for the newcomers and makes them effective contributors faster[21]. It is also common practice for all developers to ask team members for help with parts of code, as no single developer has full code comprehension in a large system. It is stated that most system knowledge lies within the heads of the team members rather than in the documentation [11].

2.2.2 Agile development

Agile development is a philosophy about development revolving around communication, working products, and customer and developer satisfaction. It is based on the *Agile Manifesto*, a short list of priorities in agile development written in 2001. The main tenets of the *Agile Manifesto* are

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

This is also extended into the *Twelve Principles of Agile Development* [3]. The main gist of Agile development is making something that works and is sufficient for the product owner, removing unnecessary overhead and focusing on the product and the people.

2.2.3 Pair programming

Pair programming is a technique used in agile software development, where two developers work together on one computer. The developer in control of the keyboard and mouse is referred to as the driver, while the other who observes and comments is called navigator or observer. The pair will switch roles often, and discuss the code as it is being written.

The pair programming technique is proven to increase code quality, especially in pairs consisting of novice programmers [9]. Novice pairs also see increase in code correctness when the task is complicated [2]. And the method by nature facilitates code comprehension and distributes knowledge of the codebase in both programmers [4].

Chapter 3

Choice of Technology and Method

3.1 Choice of technology

3.1.1 Git

Git is the most popular Version Control System (VCS) among developers [18]. Git uses a commit and branch based system to keep version and file history, and allows for parallel feature development. Git was selected as our VCS because it is familiar and it is used by the product owner.

As a service for a central git repository Github was chosen, as this is where the open source version of the base project is already uploaded, and it is a familiar service.

3.1.2 MySQL Test Framework

MySQL has a built-in test framework that comes with source and binary distributions. It uses SQL queries to test features of the system by comparing actual and expected output after the queries.

The list of queries are stored in *.test*-files, and the expected output is stored in *.result*-files. After the *.test*-file is run, the output is compared to the corresponding *.result*-file, and the test only passes if the contents are exactly the same.

Most of the MySQL system is written without unit testing, so this test framework is the best way to test functionality in MySQL. Unit testing would require mocking other parts of the system and is difficult to do. We have been adviced to not try unit testing [10].

This test framework is also very intuitive and easy to use as long as you know a bit of SQL. We have learned MySQL in our studies, so this is a good way to test functionality [15].

A hypothetical test file could look like in Figure 3.1 and would have the corresponding result file in Figure 3.2.

```
CREATE TABLE t (i1 INT, i2 INT);
SHOW CREATE TABLE t;
INSERT INTO t VALUES (1, 2), (3, 5), (-5, 7);
SELECT * FROM t;
DROP TABLE t;
```

Figure 3.1: A hypothetical test file in the MySQL test framework

```
CREATE TABLE t (i1 INT, i2 INT);
SHOW CREATE TABLE t;
Table  Create Table
t      CREATE TABLE `t` (
  `i1` int DEFAULT NULL,
  `i2` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
INSERT INTO t VALUES (1, 2), (3, 5), (-5, 7);
SELECT * FROM t;
i1     i2
1      2
3      5
-5     7
DROP TABLE t;
```

Figure 3.2: A hypothetical result file corresponding to the test file in Figure 3.1

3.1.3 GDB

GDB, or the GNU Project Debugger, is a debugger for several programming languages. It lets you break on different points in the program, step through it line by line and read data in memory to find the reason for bugs and unexpected behaviour in your program. GDB is the default debugger used for debugging C++, and this is what was suggested and is also used by Oracle.

3.2 Choice of method

3.2.1 Pair-programming

Pair-programming was utilized in parts of the project. It was used when the part in question had other relevant code dependent on it or when an extra pair of eyes was needed to complete the task. A shared understanding was essential for further development dependent on that part of the program, and for our ability to make correct decisions where one programmer was unsure. As the team consisted of only two individuals, it also ensured that the entire team had a shared understanding and participated in decision making. It also ensured maximized code correctness and quality on pair-programmed code. Pair-programming was avoided when the code written was simple to write and had little code dependent on it within our project. This was because it was more relevant to save programmer time when the advantages of pair-programming were less important than fast progress and dynamic schedules for the team. Pair-programming was utilized when the team wanted to ensure a shared understanding of the code written, or needed to discuss a piece of code.

3.2.2 Agile inspired development

During our studies we have learned of several ways to do agile development in larger teams, and even though these methods are largely designed for bigger teams, most of the ideas are applicable to pairs or single developers as well. Therefore we have chosen some aspects of these implementations to use, and some that were not fit for our team.

The Agile Manifesto

Individuals and interactions
Good development requires motivated and interested team members. We have made sure to solve any conflicts and share knowledge and discoveries whenever needed.
With direct access to experts within the MySQL development team we have also been able to access expertise about the system whenever needed.

Working software
The main focus of our project has been to deliver a working product that filled as many of the feature specifications as possible. As MySQL is already a well documented system, and our feature implements many abstract (called 'virtual') methods inherited from abstract parent classes that are already documented, most of our time could be used for making a feature complete product.

Customer collaboration
As a product of being integrated in the MySQL Optimizer Team at Oracle, we have had direct access to the product owner throughout the project. We have

been able to ask questions about implementation and code whenever we needed to, and we had regular progress meetings with the product owner.

Responding to change
MySQL has a large code base, and no single person knows how everything is put together. Therefore it is difficult to make a plan that does not need to be changed frequently during the project. We started with a simple plan of making the smallest possible changes that would satisfy the specifications, and changed the plan whenever we came over anything unexpected.

Specific points of agile development

- Core hours
  Having core hours helps interactions between the individuals in the team. During these hours we have both been available through voice and text
- Pair programming
  The team consists of two people, and the project requires high code comprehension of both team members. Pair programming is a good way to get better code comprehension
- Team building
  Outside of working hours we have been talking and discussing other matters to function better as a team. Other team building exercises have been difficult to perform during the pandemic.
- Frequent contact with customer
  Giving the product owner frequent updates and being in direct contact with experts through meetings and text helped responding to change and keeping the customer satisfied.
- Minimizing unnecessary work
  We chose to do as few and small changes as possible to make a working product. This is a good way to keep existing behaviour and make the code easily readable and debuggable.

3.3 Choice of structures and architectures

3.3.1 Google code style

Oracle uses Google C++ Code Style[5], so this project had to comply with that.

3.4 Role and work distribution

There has been no formal role distribution, as with a team of two it might not be advantageous to designate someone a leader or idea maker. Both team members

have contributed about equally when it comes to planning, brainstorming, and implementation. During the pair-programming Magnus was the primary driver and Lisa was the primary navigator. This was mainly due to Magnus' higher writing speed.

The team had no formal work distribution. This is due to the unpredictable nature of the project and the fact that the tasks in the project demanded similar skills. It was never necessary to have one team member specialise in one particular field like in full stack development projects. As the team was unable to predict what tasks came next it was rarely simple to distribute tasks in a way that would distinguish the work types.

When there was a semi-functioning version of the product available, some tasks were divided between the team members. The task of updating old tests with the new feature was mainly done by Lisa, while finding and fixing unexpected behaviour was mainly done by Magnus. Most other work was done in collaboration.

3.5 Employment and onboarding with Oracle

Oracle has a database of intended features (worklogs) and bugs for all their software products. Any non-implemented worklogs are internal documents and not accessible to the general public. To get access to these one has to be an Oracle employee, or get special permission from Oracle.

To get an employment at Oracle one first has to apply in the standard way with CV, resume and cover letter. There are usually interviews with prospective employees, but those were skipped in this case. After being accepted the new employees have to go through a few courses on insider trading, workplace safety, workplace harassment, and internal procedures.

Chapter 4

Results

4.1 Scientific results

4.1.1 ISO Standard Compliance

The data type is named BOOLEAN

The implemented data type is called BOOLEAN. This is compliant with the ISO Standard.

The BOOLEAN data type can have the values TRUE, FALSE and UNKNOWN/NULL

The implemented BOOLEAN data type can have the values 1, 0 and NULL. This is not compliant with the ISO Standard.

BOOLEAN values are comparable and TRUE is greater than FALSE, comparisons with UNKNOWN/NULL will give UNKNOWN/NULL

As the implemented BOOLEAN data type is an integer, it can be compared to other values in the same way as an integer, i.e. 1 (TRUE) is greater than 0 (FALSE), and comparison with NULL returns NULL. This is compliant with the ISO Standard.

Boolean operators NOT, AND and OR can take BOOLEAN data type as operand

Boolean operators NOT, AND and OR are previously implemented in MySQL to work with integer values, and work with the new BOOLEAN data type as well. This is compliant with the ISO Standard.

BOOLEAN data type has the same values as expressions that return TRUE, FALSE, UNKNOWN/NULL

Boolean expressions in MySQL return the values 0, 1, and NULL, which are the only values stored in columns of the new BOOLEAN data type. This is compliant with the ISO Standard.

4.1.2 Implementation

When inserting a floating or fixed point number into an integer column in MySQL it is rounded using the regular rules for rounding, i.e. [.0, .5⟩ are rounded down and [.5, .0⟩ are rounded up. In the implemented feature all non-zero numbers inserted into a BOOLEAN column are stored as 1, regardless of round-off rules. This also applies to strings that can be parsed to a number.

4.2 Engineering results

4.2.1 Feature specification from engineers

Feature specifications are outlined in the vision document, chapter 5 (Attachment A).

Create column with BOOLEAN data type

Writing `CREATE TABLE t (b BOOLEAN);` creates a table with a boolean column. This extends to creating a boolean column in a table with other column types.

Allow BOOL as synonym for BOOLEAN

BOOL and BOOLEAN are both parsed the same.

Internally, let BOOLEAN by an integer value with only two possible values: 0 (FALSE) and 1 (TRUE).

BOOLEAN is internally represented as the integers 0 and 1, as well as being nullable.

SHOW CREATE TABLE must show BOOLEAN data types.

Creating a table like in section 4.2.1 and querying `SHOW CREATE TABLE t;` prints the output shown in Figure 4.1.

DESCRIBE must show BOOLEAN data types.

Creating a table like in section 4.2.1 and querying `DESCRIBE t;` prints the output shown in Figure 4.2.

```
CREATE TABLE `t` (
    `b` boolean DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 4.1: SHOW CREATE TABLE output

```
Field  Type    Null   Key Default Extra
b      boolean YES       NULL
```

Figure 4.2: DESCRIBE output

DUMP command must show BOOLEAN data types for backup purposes.

Creating a table in a database `DB` like in 4.2.1 Create column with BOOLEAN data type and running `mysqldump DB` in a regular terminal prints the output shown in Figure 4.3.

```
CREATE TABLE `t` (
    `b` boolean DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 4.3: mysqldump output

CAST functionality

Explicit cast functionality from other data types to BOOLEAN is not implemented. Trying to query `CAST(1 as BOOLEAN)` returns an error message.
Cast from the character strings 'TRUE' and 'FALSE' to BOOLEAN is not implemented. Trying to query `CAST('TRUE' as BOOLEAN)` returns an error message.
Cast from BOOLEAN to the character strings 'TRUE' or 'FALSE' is not implemented. Casting a BOOLEAN to a character string with `CAST(b AS CHAR)` returns 0 or 1.
Cast from BOOLEAN to other scalar data types is implemented. Querying `CAST(b AS SIGNED)` or `CAST(b AS DECIMAL)` returns 0 or 1.

Implicit CAST functionality

BOOLEAN values are implicitly casted to other data types where needed.

Conversion between SQL BOOLEAN type and JSON boolean

Conversion between SQL BOOLEAN and JSON boolean is not implemented.

ALTER TABLE allows to convert between data types, according to the rules for CAST

ALTER TABLE allows changing the column type from other data types to BOOLEAN as long as all values in the original column can be inserted into a BOOLEAN column.
ALTER TABLE allows changing the column type from BOOLEAN to other data types with the same restrictions as any other integer data type.

Compatibility with existing MySQL functionality

BOOLEAN is no longer converted to a TINYINT type. Any value that is valid for insertion into a TINYINT column is also valid for insertion into a BOOLEAN column. No compatibility toggle has been implemented to allow for the BOOLEAN word to be used as an alias for TINYINT(1).

Internal issues

1. The field type MYSQL_TYPE_BOOL has been added to switch statements where necessary, allowing for the use of the new data type.
2. Class Field has a new subclass named Field_boolean. It is similar to Field_tiny, but with some different implementation.
3. No changes have been made to comparison predicates. BOOLEAN is considered an integer data type and is treated as such.

4.2.2 Feature specification from Worklog WL#3554

Where the worklog overlaps with other feature specifications, see relevant section above.

Possible values

Only the values 0, 1 and NULL can be stored in a BOOLEAN column, but any number or string representing a number can be inserted into a BOOLEAN column. When a non-zero number is inserted into a BOOLEAN column it is converted to 1 before being stored. 0 is stored as 0.

Operators

Any operators that are allowed with other integer data types are allowed with the BOOLEAN data type.

Reserved words

BOOL, BOOLEAN and UNKNOWN are already reserved words. BOOL and BOOLEAN have changed from being aliases for TINYINT(1) and now both refer to the BOOLEAN data type. UNKNOWN has not changed.

Connectors

It is unknown if this change affects connectors. There has been no indication of this.

Shown values

When you SELECT a BOOLEAN value you see { 0 — 1 — NULL} rather than { FALSE — TRUE — UNKNOWN }.

4.2.3 Tests

In addition to the implemented feature, the MySQL test framework has been extended with tests on the implemented feature. This includes a new test case (type_boolean.test), and modification to several preexisting tests. Two of the test cases relating to JSON fail. All tests using BOOLEAN data type in NDB fails, but the storage engines InnoDB, MyISAM, MEMORY, CSV, BLACKHOLE, MERGE, ARCHIVE and FEDERATED pass. All other test cases pass. This includes test cases testing already existing functionality before implementing the new feature.

In the new test case *type_boolean.test* there are several tests to check that a table is created and values stored correctly. The test in Figure 4.4 and corresponding result in Figure 4.5 show that BOOL and BOOLEAN now refer to the BOOLEAN data type. The test in Figure 4.6 and corresponding result in Figure 4.7 show that non-zero values are inserted as 1, while FALSE/0 and NULL are inserted as 0 and NULL.

type_boolean.test and *type_boolean.result* can be found in section 7.D or the codebase [17].

```
CREATE TABLE t1 (b1 BOOLEAN, b2 BOOL);
SHOW CREATE TABLE t1;
```

Figure 4.4: Sample from *type_boolean.test* showing a test for CREATE TABLE [17]

```
CREATE TABLE t1 (b1 BOOLEAN, b2 BOOL);
SHOW CREATE TABLE t1;
Table Create Table
t1 CREATE TABLE `t1` (
  `b1` boolean DEFAULT NULL,
  `b2` boolean DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 4.5: Sample from *type_boolean.result* showing the result of the test in Figure 4.4 [17]

```
CREATE TABLE t1 (b BOOLEAN);
INSERT INTO t1 VALUES (FALSE), (TRUE), (NULL), (0.2), (3), (-1), (-1.5),
    ("0.1");
SELECT * FROM t1;
```

Figure 4.6: Sample from *type_boolean.test* showing a test for INSERT and SELECT [17]

4.2.4 Implementation in code

Internally BOOLEAN is represented by the class *Field_boolean*, which extends *Field_num*, the parent class of all scalar fields. The *Field_boolean* class implements the virtual methods from *Field_num*. Most methods are implemented the same way as in *Field_tiny*, the class representing the TINYINT data type, except when different behaviour is needed.

Field_boolean::store

There are four store methods, taking either a `char *`, a `double`, a `longlong`, or a `my_decimal`, a fixed-point number class internally in MySQL. The "main" store method, which the other store methods call, takes a longlong and stores the result of the test `value != 0`. The store method that takes a `char *` parses the string to a floating point number before storing the number. If the text cannot be parsed, an error is thrown.

The store method taking a `double` stores the result of the test `value != 0.0e0` using the first store method. The store method taking a `my_decimal` stores the result of the test `!my_decimal_is_zero(value)` using the first store method. The main store method is shown in Figure 4.8, where `ptr` is an object variable in the *Field* class pointing to the stored value.

```
CREATE TABLE t1 (b BOOLEAN);
INSERT INTO t1 VALUES (FALSE), (TRUE), (NULL), (0.2), (3), (-1), (-1.5),
    ("0.1");
SELECT * FROM t1;
b
0
1
NULL
1
1
1
1
1
```

Figure 4.7: Sample from *type_boolean.result* showing the result of the test in Figure 4.6 [17]

```
type_conversion_status Field_boolean::store(longlong nr, bool) {
  ASSERT_COLUMN_MARKED_FOR_WRITE;
  type_conversion_status error = TYPE_OK;
  // Maybe change to *ptr = nr != 0
  if(nr != 0) {
    *ptr = 1;
  } else {
    *ptr = 0;
  }
  return error;
}
```

Figure 4.8: The main Field_boolean::store() method [17]

MYSQL_TYPE_BOOL

MySQL has an internal enum with a list of valid data types. MYSQL_TYPE_BOOL is the value for the BOOLEAN data type. Several places in the code there are switch statements that are based on this enum. These switch statements have different functions, and MYSQL_TYPE_BOOL was added to these when necessary.

An example of where MYSQL_TYPE_BOOL is implemented into a switch statement is in the method `mysql_query_attributes::query_parameter_val_str` in the file `sql/mysql_query_attributes_imp.cc`, shown in Figure 4.9.

```
static String *query_parameter_val_str(const PS_PARAM *param,
                                       const CHARSET_INFO *cs) {
  String *str = nullptr;
  switch (param->type) {
    // the expected data types listed in the manual
    case MYSQL_TYPE_BOOL:
      if(param->length == 1) {
        bool value = static_cast<bool>(*param->value);
        str = new String[1];
        str->set_int(value, param->unsigned_type != 0, cs);
      }
      break;
    case MYSQL_TYPE_TINY:
      if (param->length == 1) {
        int8 value = (int8)*param->value;
        str = new String[1];
        str->set_int(value, param->unsigned_type != 0, cs);
      }
      break;
.
.
.
    }
}
```

Figure 4.9: The head and first part of method query_parameter_val_str [17]

4.2.5 Code examples

There is too much code to show it all in the report, but here are some examples.

Field_boolean::cmp

The *Field_boolean::cmp* method as seen in Figure 4.10 is implemented in the standard way for a compare method. If the first parameter is smaller than the second, return `-1`. If the first parameter is larger than the second, return `1`. If they are the same, return `0`.

```
/**
 * @brief Compare two uchars as booleans
 *
 * @param a_ptr pointer to first value to compare
 * @param b_ptr pointer to second value to compare
 *
 * @return -1 if *a_ptr < *b_ptr, 1 if *a_ptr > *b_ptr, 0 if *a_ptr ==
     *b_ptr
 */
int Field_boolean::cmp(const uchar *a_ptr, const uchar *b_ptr) const {
  bool a, b;
  /*  The overridden cmp method takes uchar, so they are converted to
      bools before comparing */
  a = static_cast<bool>(a_ptr[0]);
  b = static_cast<bool>(b_ptr[0]);
  return (a < b) ? -1 : (a > b) ? 1 : 0;
}
```

Figure 4.10: The Field_boolean::cmp method [17]

Item::tmp_table_field_from_field_type

In *Item::tmp_table_field_from_field_type* in Figure 4.11 a case for MYSQL_TYPE_BOOL has been added. It is similar to the cases for other integer types. As length and signedness are meaningless to a BOOLEAN, these are omitted in the constructor.

```
/**
  Create a field based on field_type of argument.

  For now, this is only used to create a field for
  IFNULL(x,something) and time functions

  @return Created field
  @retval NULL error
*/
Field *Item::tmp_table_field_from_field_type(TABLE *table,
                                            bool fixed_length) const {
  /*
    The field functions defines a field to be not null if null_ptr is
        not 0
  */
  Field *field;

  switch (data_type()) {
    case MYSQL_TYPE_DECIMAL:
    case MYSQL_TYPE_NEWDECIMAL:
      field = Field_new_decimal::create_from_item(this);
      break;
    case MYSQL_TYPE_TINY:
      field = new (*THR_MALLOC)
          Field_tiny(max_length, m_nullable, item_name.ptr(),
              unsigned_flag);
      break;
    case MYSQL_TYPE_BOOL:
      field = new (*THR_MALLOC)
          Field_boolean(m_nullable, item_name.ptr());
      break;
.
.
.
  }
  if (field) field->init(table);
  return field;
}
```

Figure 4.11: The head and MYSQL_TYPE_BOOL part of the *Item::tmp_table_field_from_field_type* function [17]

4.3 Administrative results

The administrative results are further documented in the project handbook(attachment 7.C).

4.3.1 Project goals

As the Gannt-diagram in Figure 4.12 shows, the coding lasted longer than planned while testing took less time than estimated. And the delivery time of the video moved the delivery date. Otherwise the plan was somewhat optimistic.

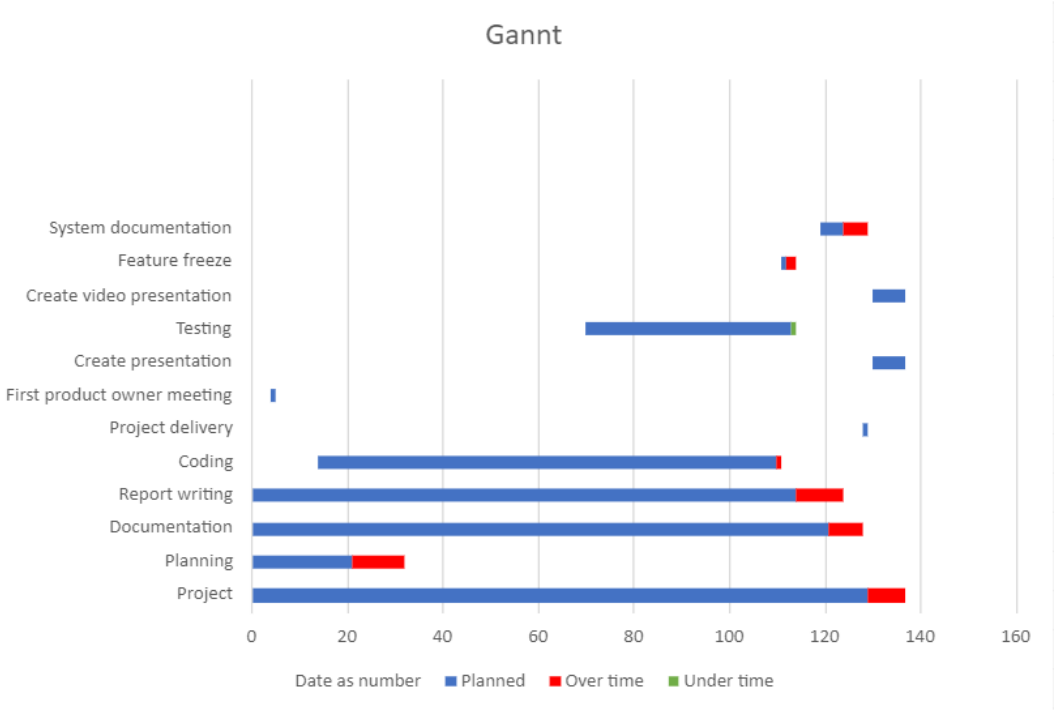


Figure 4.12: Gannt-diagram

4.3.2 Worked hours and activity distribution

The pie charts in Figure 4.13, Figure 4.14 and Figure 4.15 represent the distribution of worked hours per person and in total.

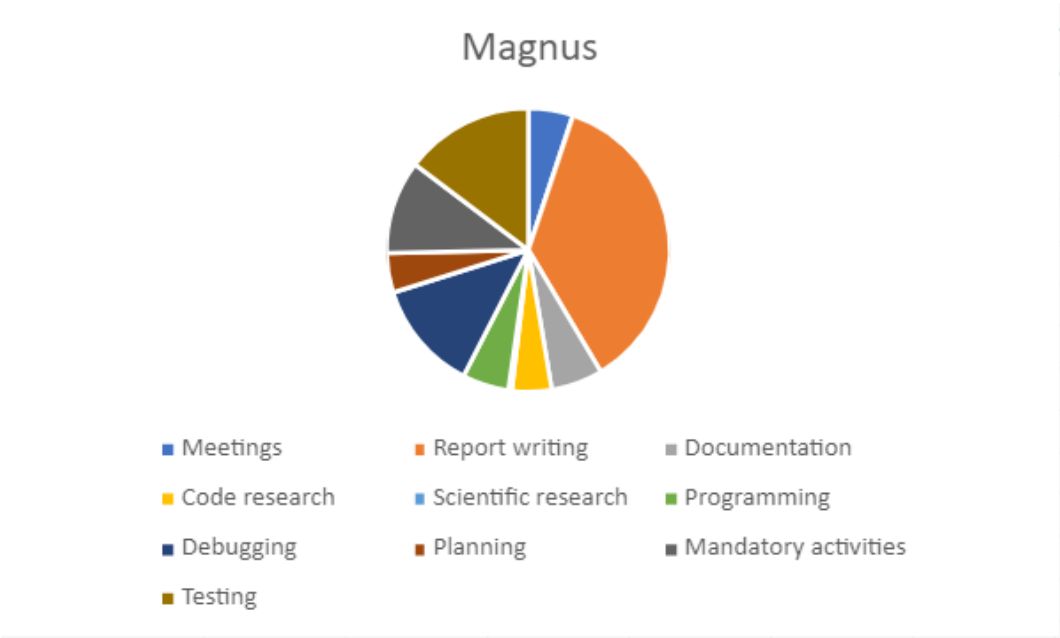


Figure 4.13: **Magnus Brevik:** 509 total hours

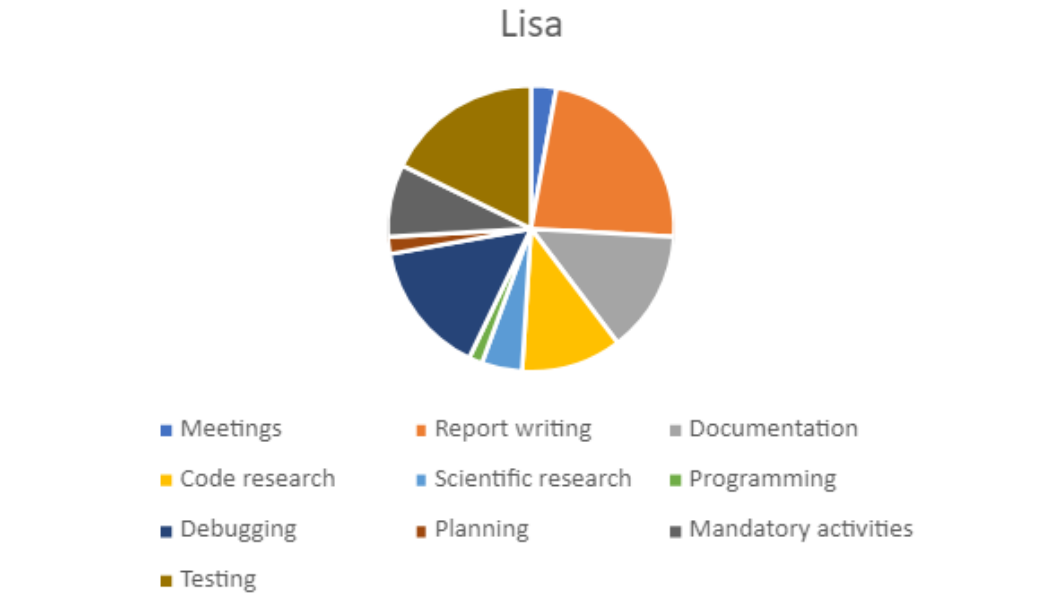


Figure 4.14: **Lisa Willa:** 525 total hours

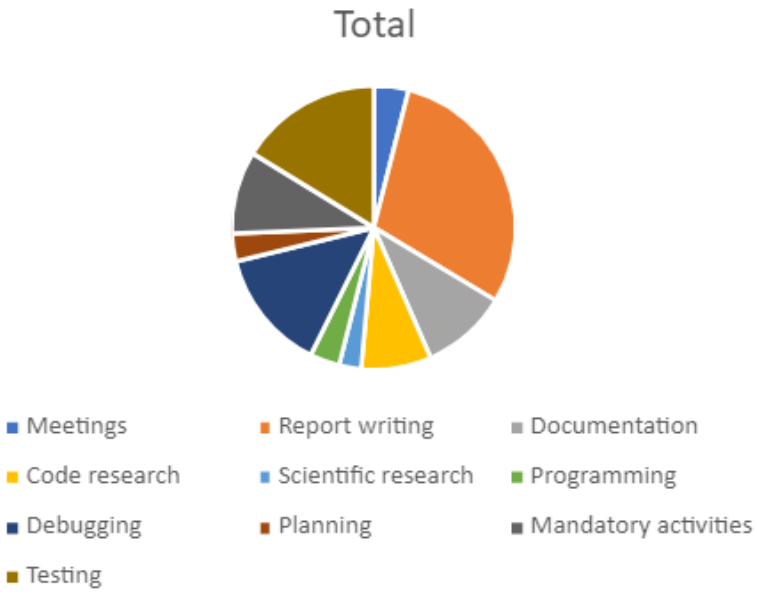


Figure 4.15: **Sum total:** 1034 total hours

Chapter 5

Discussion

5.1 Scientific

5.1.1 SQL Standard Compliance

The non-compliance of TINYINT(1)

In the lack of a BOOLEAN data type, MySQL uses TINYINT(1) instead. As TINYINT stores an entire byte, a "BOOLEAN" in MySQL can store 256 different values, as well as being NULL. This can lead to confusion, and it does not conform with the ISO standard.
Having an integer instead of a boolean value means any mathematical operators work with the "boolean" data type. It also entails that any boolean operator has to work with regular integers. This leads to undefined behaviour, and is not allowed in conforming SQL languages.

Whenever a boolean expression is made with integer operators, they are first treated as truthy or falsy before being compared.

The BOOLEAN data type can have the values TRUE, FALSE and UNKNOWN/NULL

Oracle wanted the BOOLEAN data type to be an integer type with the values 1 and 0 representing TRUE and FALSE, to ensure backwards compatibility with the previous function of the BOOLEAN word. Since BOOLEAN was an alias for TINYINT(1) it had all the functionality of an integer value, which is behaviour Oracle wanted to keep.

Representing the BOOLEAN data type as 1 and 0 is the most visible breach with the ISO standard. According to the standard, the values TRUE and FALSE are separate from and not directly comparable to any other data types. Representing the result of boolean expressions as integers leads to behaviour like being able to add two BOOLEANs and get 2. As adding is already a defined

action with mathematical booleans, this is unexpected behaviour and in most cases wrong.

All fields in MySQL are nullable, and BOOLEAN fields are no exception. As long as `NOT NULL` or other constraints are not specified when creating a BOOLEAN column, the allowed values are 1, 0, and NULL, which is the closest we can get to being standard compliant while keeping BOOLEAN an integer type.

5.1.2 Implementation

Even though BOOLEAN is implemented as an integer type, the standard way of inserting a floating or fixed point number into an integer column, by rounding, is not followed. This is due to the already existing treatment of non-boolean values in a boolean context in MySQL. The result of the SQL query `SELECT 0.01 IS TRUE;` is `1`, which means 0.01 is treated as a truthy value whenever used in a boolean context. Any other non-zero numbers are treated the same way. Therefore it would be most consistent to treat any value as truthy or falsy when inserting it into a BOOLEAN column.

5.2 Engineering

5.2.1 Completed features

Most of the features desired by Oracle are implemented. A server based on our modification of MySQL works exactly like the current version MySQL as long as one does not use the BOOLEAN data type.

The BOOLEAN data type works as expected, with a few exceptions, outlined in subsection 5.2.2. As long as the storage engine NDB Cluster is not used most features expected of a BOOLEAN data type should be functional.

Being able to make the new datatype work so similarly to other features already implemented in MySQL is due to the fact that we utilized code from the project. Using polymorphism with existing classes, preexisting functions, and updating several switch statements instead of building the logic for BOOLEAN from scratch helped avoiding cluttering the codebase with unnecessary code and "hacky" solutions. Integrating into such a large codebase and implementing a feature to work with the already complex system required good code comprehension skills and many hours of reading code.

5.2.2 Incomplete features

The CAST method in MySQL has a list of target types that does not coincide with the list of data types in MySQL. While there are several integer data types, i.e. INTEGER, SMALLINT, TINYINT, MEDIUMINT, BIGINT, there are only

two integer types available as target types for CAST, SIGNED and UNSIGNED. As both of these convert the value to an integer type, and integer types can be treated like booleans, CAST functionality to BOOLEAN was down-prioritized. When other features were ready we did not have time to implement this feature.

The problems with JSON were discovered late in the project, right before feature freeze, and thus were given lower priority. Making sure tests and already implemented features were working was prioritized over JSON.

NDB or NDBCUSTER is a clustered database engine, and it does not currently work with our implementation of the BOOLEAN datatype. This in itself was considered acceptable by Oracle, but we also lack a temporary fix that ensures that an NDB database does not try to use our BOOLEAN, but instead uses the TINYINT(1) alias. This was something we worked hard on implementing, but we found that we were unable to do that without changing large quantities of code which we did not have the time to do. And big structural changes in code used for many things is not something novices should do without supervision. Oracle decided that we should not do it, since it would take too much of our time at that point [10].

5.2.3 Testing

We ended up producing our own test and modify several of the preexisting tests in the codebase. Some tests were changed to test BOOLEAN cases where there previously was none. Other tests were changed to accept the new correct behaviour in tests involving BOOLEAN. This took a lot of time, but also alerted us of lacking functionality and errors in the system. There is also a rule that says that MySQL optimizer team will only take in changes that pass all tests so this was important to work on, if we wanted our changes to be integrated into MySQL in the future.

5.2.4 Implementation in code

Even though the product owner had a clear vision for most of the features of the BOOLEAN data type, there were some choices that had to be made during the project. Should it be possible to insert values other than 0 and 1? How should small floating and fixed point numbers be treated? How should strings be treated?

The answer to the two first questions was to check the truthiness of the inserted value, and store that instead of the actual value. The version written in code could be simplified to `*ptr = nr != 0`, but this was a discovery made after feature freeze.

When it comes to strings there are two main approaches that can be made. The first one is to throw an error whenever a string is tried inserted into a

BOOLEAN column. This is not ideal, as strings are already silently converted to the correct data type when possible, and denying that would make MySQL inconsistent.

The other way is to attempt to parse the string to a number before trying to store the value. As our implementation allows for inserting any number into a BOOLEAN column, it only makes sense to be able to insert any string representing a number as well.

The strings "TRUE" and "FALSE"

When allowing for insertion of strings the question of the strings "TRUE" and "FALSE" comes up. Trying to insert these into any other scalar column would result in an error, but these words have special meaning in a boolean context. Currently, these words are treated as regular strings, and throw errors, as they cannot be parsed directly to numbers. There is a case to be made both for keeping it this way, and for allowing these to be inserted as TRUE and FALSE.

The simple case of letting it stay like it is, is consistent with other scalar data types, and conforms with the ISO standard. At this point, the ISO standard has already been ignored quite a bit by making the BOOLEAN data type an integer type, and allowing any number to be stored.

Allowing these strings to be inserted would require an extra check in the store-method, but that would not impact the performance of the system noticeably. Having support for inserting "TRUE" and "FALSE" into BOOLEAN columns could make the BOOLEAN data type more intuitive and easy to use, as the user would not have to convert from possible string representations of TRUE and FALSE before inserting into the database. This might be a niche problem, but it is worth considering.

The question about whether the strings "TRUE" and "FALSE" should be allowed as TRUE and FALSE was not directly addressed, as it appeared late in development and was prioritized below other features.

MYSQL_TYPE_BOOL

A change that was barely mentioned in chapter 4 was the moving or insertion of the enum MYSQL_TYPE_BOOL in several switch statements. These look like minor changes, but are essential for the feature to work. One change changes the word BOOLEAN from giving a *Field_tiny* to using the newly implemented class *Field_boolean*, while others let the system use the functionality of the new class. The changes were done by finding where the other data type enums were used, and adding MYSQL_TYPE_BOOL either to a list of scalar or integers, or adding another distinct case slightly different from the other cases.

The example case in Figure 4.9 shows MYSQL_TYPE_BOOL in a switch statement. The case for MYSQL_TYPE_BOOL is simple, and similar to the case for MYSQL_TYPE_TINY below. The difference is the use of `static_cast<>()`, as advised by Oracle, and that it is cast to `bool` instead of `int8`. Other, more complex data types have more complex code for their cases, for example needing some conversion or interpretation of data.

This is mostly the extent to which changes adding MYSQL_TYPE_BOOL to switch statements have been done. There have been several places similar changes were needed, but showing and explaining all of them would be redundant.

5.2.5 Code examples

Field_boolean::cmp

Field_boolean::cmp in Figure 4.10 is a comparison method used for sorting BOOLEANs. It is overridden from the *Field* class. The last line of the method is a standard way of comparing two numbers.

Seeing as this method compares two uchar (unsigned char), which are unsigned integers, the method could have been made simpler. Since sorting methods usually check whether the comparison results in a negative or positive number, or if it is 0, returning the value from `*a_ptr - *b_ptr` could have been a viable solution, but seeing as we have no control over the sorting algorithms of MySQL, it is safer to follow the style of other cmp methods.

Casting the values from `uchar*` to `bool` is mostly to ensure correct behaviour, but in theory makes no difference.

Item::tmp_table_field_from_field_type

Item::tmp_table_field_from_field_type in Figure 4.11 is one of the functions we had to change to make the feature work as intended. As the comment says, this method is only used for the `IFNULL` command and time functions. It creates a new instance of the relevant field and returns it.

This is just an example, as there are many similar changes made. These methods often contain a switch statement with every data type represented. Some times BOOLEAN needs it's own behaviour, and thus a separate case from the rest, like here, and in other places BOOLEAN can have the same behaviour as any other integer or number type. In those cases a MYSQL_TYPE_BOOL case could simply be added to the list of integer cases.

These are some of the choices and considerations we have had to make during development, which done correctly speaks of good code comprehension.

5.2.6 Adapting to Oracle's code

Adapting to the MySQL server codebase was very time consuming. Our experience is primarily from development projects where all code comprehension exists within the team, and most information can be found on the internet. However, in this enhancement project all information had to come from the code itself and it was more complicated to work with than explanations on stack overflow. Everything from error codes to components were internal and not something you would find with a google search. This is also by far the biggest codebase we have worked with, so our experience in dealing with large codebases was lacking as well.

We quickly realized that everything we did would have to fit in the existing system because everything was heavily dependent on other pieces of code. So our changes had to fit with the preexisting logic, or we would face many issues further along. Our strategy heavily emphasised making small insertions of code to include the BOOLEAN datatype into the preexisting logic. This approach allowed us to get high effect for few code lines and spared us a lot of trouble, but it was also more complicated than it seemed. We would have to understand the logic and functionality in the piece of code, and then determine if this was a place BOOLEAN should be. It saved us a lot of work, workarounds and code lines, but demanded a lot of code comprehension work.

We were also using the programming language C++ and Google coding style. We both had a small subject teaching C++, but the google coding style was new to us. We found that code comprehension was just as important as our C++ knowledge in finding out how to do the coding. And we also found that mimicking the google coding style was doable. We also found that as we got used to reading code that uses google coding style it became easier to read the code. The consistency in the coding style allowed us to read code more effectively as we got used to it.

In addition to the coding style we also mimicked their code and comments. Making our code and comments similar to the what can be found in the codebase was a conscious choice to ensure readability and consistency in the codebase. A lot of our code resembles the equivalent code for TINYINT, so that it looks and behaves as similar as possible to the preexisting logic for similar data types. Similarly we have commented equivalent comments in doxygen to ensure a consistent documentation of the codebase. However, we have added more explanatory comments around the code on request from Oracle.

5.3 Administration

5.3.1 Project goals

The nature of the project was that of an enhancement project, meaning that we had to familiarise ourselves with a new codebase, and add to it. MySQL server is a large codebase, and we had a short onboarding into the MySQL optimizer team, consisting of a quick introduction to the MySQL server codebase during our first meeting [10]. We quickly found ourselves working on understanding code, far more than on writing it. Our experiences are consistent with research on code comprehension in enhancement projects executed by newcomer, to a codebase, novices like us. We also worked on a single feature. This meant that all the code relevant for the feature was connected and changes done in these places could impact the behaviour in unforeseen ways. This made it hard to work concurrently.

Detailed planning of the project quickly proved difficult because of the nature of the project. The large new codebase had us start with a very limited code comprehension, which made estimating time use on a task difficult. Working on the same feature meant that task distribution demanded more code comprehension than we had, so we often had to work atomically on programming to avoid issues with git. We found that we ended up progressing by resolving errors or adding lacking sub-features, as they were discovered, rather than by following a structured plan. These issues were unpredictable to us, and became top priority whenever they were uncovered, and it was natural to start working on them immediately. Making detailed planning very difficult and practically non beneficial for the programming part of the project.

Concurrent work became more attainable when the system was so far along that we had few errors and could effectively use the tests or our working server to detect lacking functionality. Since the system was a working minimum viable product and we had an improved code comprehension at this point, we knew better what would impact what parts of the system and thus not work on the same places or impact what the other one was doing. We could ensure that what we did worked, before pushing to version control. We also knew that each of us did not work in the same place as the other team member. At this point in the project we mostly ensured that the system worked as expected and tried to make tests pass in the test system.

5.3.2 Agile development methodology

Pair programming

The pair programming was very successful, and helped distribute and build code comprehension. The pair programming was primarily meant to distribute understanding of the code we wrote using the technique, but we found that

through discussions in our sessions that we built a common code comprehension of code connected to ours as well. We also discussed our interpretations of the specifications in these sessions when relevant, ensuring a common understanding of the goal.

Our reason for using pair programming also came from the fact that we often had situations in which only one task was available and it was essential for opening up new tasks. With these tasks we used pair programming to find the solution faster, and to give both team members code comprehension for the important part of the code.

We also avoided bugs and low quality solutions by having two developers involved, improving code correctness and quality. So pair programming worked very well for our project.

Stand up meetings and core hours

Our stand up meetings combined with our core hours allowed us to give each other updates every day. These meetings could also include some planning. Both were available for questions and comments on Slack throughout our core hours, which enabled good teamwork despite both working from home. We would also have Zoom meetings to discuss topics as they came up. This was as close as we came to a traditional open-plan office, as continuous Zoom calls would be somewhat draining over time.

Customer collaboration

Our meetings with the product owner combined with our contact with the expert on Slack allowed us to ask questions whenever they occurred to ensure we did as the product owner wanted, though we probably should have had more meetings to ensure their ability to comment on our code as it progressed. Since the representatives of the product owner we had contact with where both experts, their comments on our code would be very relevant, in contrast to projects in which the product owners are not developers. We also found that in some instances we should have asked for help from the experts earlier, as they proved to be very well equipped to solve our problems and give advice. If we had been better at asking for assistance we might have accomplished more.

We also managed to effectively adapt to change thanks to these meetings and our "solve as problems appear" strategy. For example, when we found that the storage engine NDB did not work with boolean, it was decided by mail that we should try to find a temporary workaround where the server would use TINYINT instead of BOOLEAN when NDB was used. By meeting 9 we had not been able to complete this, and in this meeting we came to an agreement with the Oracle experts that we should down-prioritize it [10].

Agile principle: Simplicity

We tried to use the principle of simplicity from agile. Specifically we tried to utilize functionality that existed in the system rather than creating new functions and objects that would create unnecessary clutter in the code. This required more code comprehension, but ensured our impact on the codebase was as noninvasive and elegant as possible. This is something we accomplished to a high degree.

Agile

Despite our use of agile techniques like pair programming and stand up meetings, and emphasis on agile principles our custom methodology does not necessarily qualify as agile. We lack iterations, and boards. We found ourselves unable to implement sprints because we could not set up a board that worked. This was because of our lack of understanding for the system and code structure.

Boards are an important way in agile projects to keep track of tasks and progress. It is common for a task on a scrum board or kanban board to be a single feature that does not affect other tasks on the board. We lacked the code comprehension to split our feature into sub-tasks that could work that way. We tried and found that the board did not work. Tasks would be resolved 5 at a time, or be impossible for us to work on without losing order. The closest we got to a board is our lists of tests to modify or work on passing that we had at the end. This was however just implemented as a list in a shared Teams document, as the culture of making a board was not there.

Sprints is one of the most important aspects of agile methods. We would have liked to implement sprints, and we actually tried. Despite our attempts the lack of code and system comprehension made sprint planning, which include making a board, unattainable. We were unable to predict time usage on the tasks we found, and we usually did not know where to look unless we knew the issue. Trying to split off in search of future programming tasks proved to be ineffective and possibly non beneficial.

We can not exactly make up for lacks of sprints, but we have had contact with our product owner and demonstrated our product and progress in semi-periodic meetings, like one would at the end of a sprint, and have had continuous contact with the product owner's representatives. Sprints have however not been executed as they would in agile methods, and can therefore not be documented as such.

5.3.3 Reflection Teamwork

The teamwork was very successful. The communication was good, which made the pair-programming smooth and effective, despite our lack of training in the

technique. The team members were very like-minded, so we were able to discuss ideas and problems very effectively. The members managed to be patient with each other and figure out core hours and meeting times dynamically.

5.3.4 Project in system perspective

This project might be able to help promoting the open source RDBMS MySQL. The purpose of this project is to make the MySQL RDBMS more intuitive and more standard compliant, and thus more accessible to new users. It would also keep existing users using MySQL, because of the more intuitive implementation. And draw in new and old users since ISO compliance would help improve MySQLs reputation. Seeing as MySQL is an open source RDBMS making MySQL more accessible and reputable would mean improving and promoting a free to use RDBMS with very cheap database server host services. This would make database server functionality more available for startups and developers with little money. Possibly improving the position of ambitious entrepreneurs as they can use a more intuitive and reputable database for little to no money.

5.3.5 Profession ethics

The ethical aspects of this system are the same as other RDBMSs. The system has the capacity to be used for crime, storing personal information illegally, and other misuse of its ability to store large quantities of information. MySQL also has the ability to enable charities, governments and hospitals to store lifesaving information. This is a project that has capabilities to cause good and harm, dependent on its use, and since MySQL is open source neither we nor Oracle has any control over how it is used. MySQL by nature enables safe and reliable storage of information, regardless of the intent or use of that information, making it a somewhat concerning, though helpful system.

Our modifications mostly help making MySQL more intuitive to users. And so our additions do not increase or decrease MySQL's ability to cause harm. This is because despite the BOOLEAN data type being a new and desired functionality, it does not change the capabilities of the system, since the old solution also functioned as a boolean if used correctly.

5.3.6 Employment and onboarding with Oracle

The employment process and the courses in the onboarding process were somewhat time consuming, but gave us insight into the lives of Oracle employees. The biggest problem about this was the time in which we could not really work on the project because we lacked the equipment and information necessary to proceed with our work effectively. This included lack of access to important documents, like the worklog [16] that served as feature specifications, the ISO standard [7], and hardware equipment powerful enough to work with the development effectively. Since it took some time before we got inside the Oracle

system and the equipment we needed, we had a period of time in which we did little work on the project, which somewhat disrupted our progression.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

6.1.1 Conclusion thesis

Developing this feature we found that there are many compliance issues with the ISO standard in MySQL. The new BOOLEAN data type is also in several ways non-compliant, the most important of which is probably having BOOLEAN be an integer type.

On the other hand, the new BOOLEAN data type has helped remedy several of the existing compliance problems. MySQL had several features implemented that were not allowed without a BOOLEAN data type. With the new BOOLEAN data type, MySQL is now allowed to have these features while being standard compliant.

The BOOLEAN data type in MySQL will never be completely standard compliant, but MySQL as a whole is more standard compliant now than before the implementation of the new feature.

6.1.2 Conclusion engineering

There are a few things that have not been implemented yet, including NDB support, CAST to BOOLEAN and JSON casting and conversions. Other than that it works as specified in the worklog and specifications from the Oracle engineers. Even though we have implemented a BOOLEAN data type that works for most usages and not changed the preexisting behaviour of the system, it can not be immediately implemented into the current version MySQL. All test cases must pass for a version of MySQL to be pushed to production, and

as some tests on NDB and JSON fail, this condition is not yet met by our system. This means that there is a bit of work left before our changes can go into production.

6.1.3 Conclusion process

We find that our methodology is not exactly agile, but we do find our solution to be satisfying considering the issues we had planning sprints and making scrum boards because of our lacking code and system comprehension. We have implemented several strategies and techniques in our custom methodology to make up for our weaknesses. Sub optimal work environment was improved with stand up meetings and core hours. The project type being a complicated enhancement project was overcome by the use of pair programming and good frequent communication. Lack of ability to plan a sprint or use a scrum board was amended by communication within the team and with the project owner through demonstrations of product in meetings, giving us the ability to adapt to change. Though we did not succeed in implementing a truly agile methodology, we did manage to avoid pitfalls associated with non-agile methodologies thanks to the agile elements we implemented.

6.2 Further work

Before our project can be implemented into MySQL, our system has to pass all tests, including those that do not pass yet. The team actually intends to fix the JSON tests after delivery time. Implementing a unit test is also something we have not implemented yet, but should be made at some point in the future. Allowing NDB to use our feature would be a bigger fix that most likely will be left to MySQL optimizer team. The CAST functionality should also be addressed. There should be implemented a flag that makes BOOLEAN data type print TRUE, FALSE and UNKNOWN instead of 1, 0 and NULL to make it more compliant.

For those who might be asked to do a similar task for MySQL optimizer team in the future, we would like to come with some advice. You can look at our work, see where we have changed things and evaluate if that is relevant for your case. Look at other aspects of the system that has similar functionality to what you need, in case you can use it. And ask the experts when you have questions or are stuck, it might save a lot of time.

For those who are newcomers in an enhancement projects, here is some advice. Ask the experts before you waste time, they know the system and can point you in the right direction, or might even have the solution. Realize that when you do not know the codebase things take more time, but will seem easy afterwards, because then you have the necessary code comprehension for the task. Planning can prove difficult if it is not done for you by the experts who

know the system, and you work on features that affect each other. Ask experts for help here and maintain good and frequent communication.

Bibliography

[1] Md Kamaruzzaman Software Architect. *MySQL popularity*. 2021. URL: https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba.

[2] Erik Arisholm et al. "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise". In: *Software Engineering, IEEE Transactions on* 33 (Mar. 2007), pp. 65–86. DOI: 10.1109/TSE.2007.17.

[3] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: https://agilemanifesto.org/.

[4] Andrew Begel and Nachiappan Nagappan. "Pair Programming: What's in It for Me?" In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '08. Kaiserslautern, Germany: Association for Computing Machinery, 2008, pp. 120–128. ISBN: 9781595939715. DOI: 10.1145/1414004.1414026. URL: https://doi.org/10.1145/1414004.1414026.

[5] Google. *Google C++ Style Guide*. May 4, 2021. URL: https://google.github.io/styleguide/cppguide.html.

[6] *ISO info*. 2021. URL: https://www.iso.org/about-us.html.

[7] "Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)". In: 2000 (Dec. 2016).

[8] *JTC 1 info*. 2021. URL: https://jtc1info.org/about/.

[9] Kim Man Lui and Keith C.C. Chan. "Pair programming productivity: Novice–novice vs. expert–expert". In: *International Journal of Human-Computer Studies* 64.9 (2006), pp. 915–925. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2006.04.010. URL: https://www.sciencedirect.com/science/article/pii/S1071581906000644.

[10] Roy Lyseng and Norvald Ryeng. *Meetings with Oracle employees*. Personal communication. 2021.

[11] Walid Maalej et al. "On the Comprehension of Program Comprehension". In: *ACM Trans. Softw. Eng. Methodol.* 23.4 (Sept. 2014). ISSN: 1049-331X. DOI: 10.1145/2622669. URL: https://doi.org/10.1145/2622669.

[12] Anneliese Mayrhauser, A. Marie Vans, and Steve Lang. “Program Comprehension And Enhancement Of Software”. In: (June 1998).

[13] *MySQL popularity StackOverflow*. 2020. URL: https://insights.stackoverflow.com/survey/2020#technology-databases-all-respondents4.

[14] Oracle. *MySQL 8.0 Reference Manual*. May 9, 2021. URL: https://dev.mysql.com/doc/refman/8.0/en/.

[15] Oracle. *The MySQL Test Framework*. May 4, 2021. URL: https://dev.mysql.com/doc/dev/mysql-server/latest/PAGE_MYSQL_TEST_RUN.html.

[16] Oracle. “WL#3554 Boolean data type (internal document)”. Apr. 30, 2021.

[17] Oracle, Magnus Brevik, and Lisa Willa. *mysql-server:delivery*. May 20, 2021. URL: https://github.com/mbremyk/mysql-server/tree/delivery.

[18] *Version Control Systems Popularity in 2016*. 2016. URL: https://www.overleaf.com/learn/latex/Bibliography_management_in_LaTeX.

[19] Markus Winand. *The Three-Valued Logic of SQL*. May 8, 2021. URL: https://modern-sql.com/concept/three-valued-logic.

[20] Xin Xia et al. “Measuring Program Comprehension: A Large-Scale Field Study with Professionals”. In: *IEEE Transactions on Software Engineering* 44.10 (2018), pp. 951–976. DOI: 10.1109/TSE.2017.2734091.

[21] Rebecca Yates, Norah Power, and Jim Buckley. “Characterizing the transfer of program comprehension in onboarding: an information-push perspective”. In: *Empirical Software Engineering* 25 (Jan. 2020). DOI: 10.1007/s10664-019-09741-6.

Chapter 7

Attachments

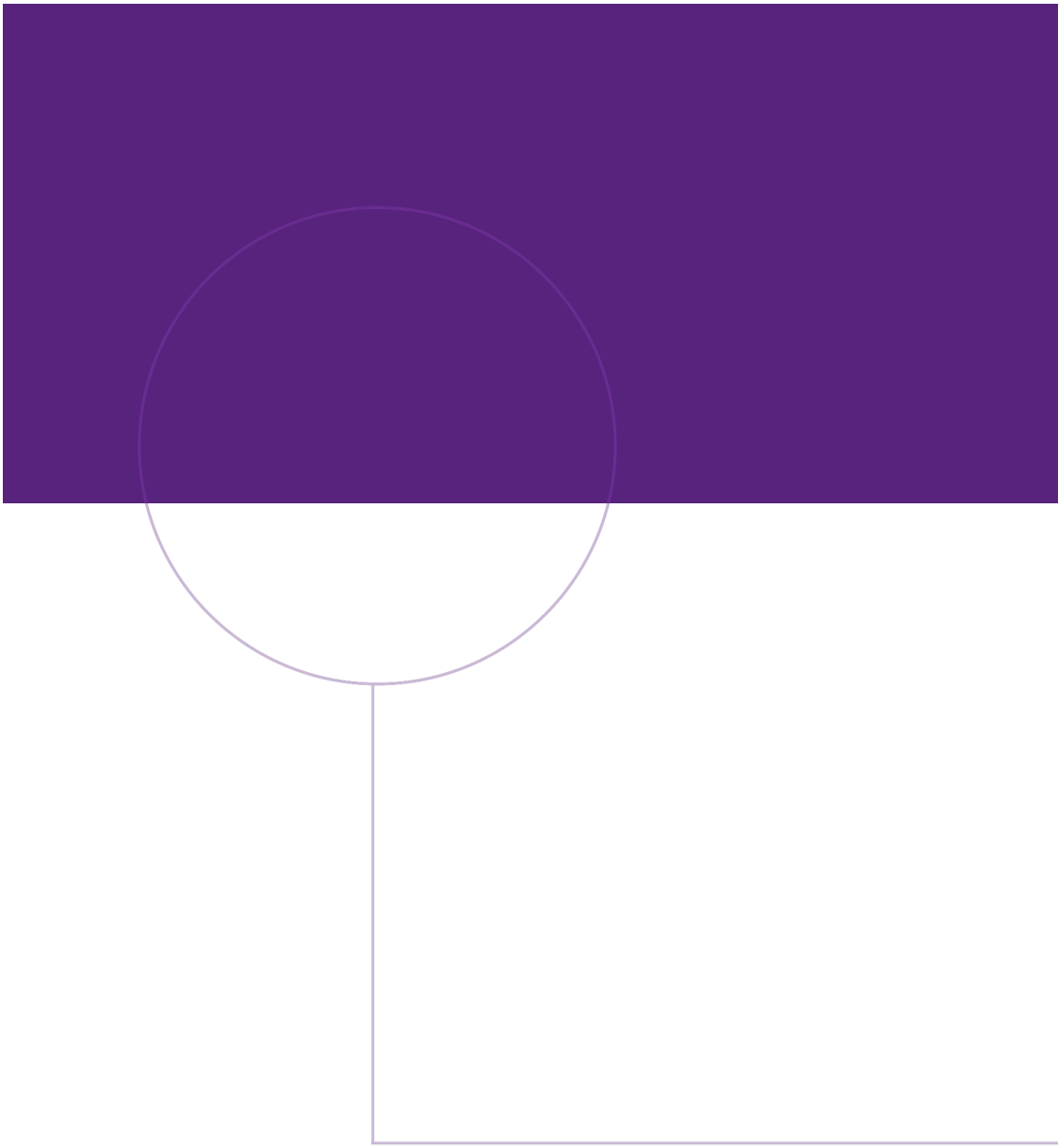
A Vision Document

B Worklog WL#3554 BOOLEAN data type

C Project handbook

D System documentation

E MySQL codebase with BOOLEAN data type



NTNU
Kunnskap for en bedre verden

ORACLE