MariaDB

Extra features that make it a better branch of MySQL

Michael “Monty” Widenius
MariaDB hacker
monty@askmonty.org
http://mariadb.com/

(Very) Brief MySQL history

Unireg (base of MySQL code) was started 1981.

MySQL code was released December 1995 under dual licensing.

MySQL Finland Ab took in investment and hired a new CEO, Mårten Mickos, in 2001.

MySQL Ab was sold to Sun in March 2008 for 1 billion $.

Monty & others left Sun in Feb 2009 to work on Maria engine.

Oracle started to acquire Sun (including MySQL) in April 2009.

Oracle got a “statement of objection” from the European. Commission regarding the MySQL part of the Sun deal in November, 2009.

EU Commission approved the deal in March 2010 without any commitments from Oracle. **Russia** got some firm commitments.

MySQL's future is still unsure. MariaDB's future isn't.

Why MariaDB was created

"Save the People, Save the Product"

- To keep the MySQL talent together
- To ensure that a free version of MySQL always exists
- To get one community developed and maintained branch
- Work with Drizzle (and other MySQL forks/branches) to share knowhow and code

After Oracle announced it wanting to buy Sun & MySQL this got to be even more important.

Monty Program Ab

- Started in February 2009 after Monty's exit from MySQL/SUN
- Shifted focus from Aria (Maria) storage engine to MariaDB (A branch of MySQL) after Oracle acquired Sun.
- Drives (but doesn't own) the MariaDB development
- Founding member of the Open Database Alliance (ODBA)
- “Virtual company” (no offices) with 20+ employees all over the world. All original 'core' developers of MySQL are employed. (Full optimizer team, 3 of 4 MySQL architects, etc)
- Very technical company (only development, open source consulting and L3 (bug fix & advanced) support)
- Uses the Hacking Business Model ('Company is owned by the employees')

About MariaDB

MariaDB is a branch of MySQL with extra features

- User level compatible with MySQL (unlike Drizzle)
  - Binary drop in replacement for MySQL 5.1
  - More plugins, more features, better code quality.
- GPL-only license
- More and more open development
  - Source in public repository on launchpad
  - Open build system (buildbot)
  - Open Knowledgebase (Replaces manual, form, FAQ...)
  - Several active external contributors
- Current state
  - MariaDB 5.1.42 was released as stable in February. Current release is 5.1.50. (~One release a month)
  - MariaDB 5.2-RC out this week

MariaDB 5.1 vs MySQL

Extra features

- XtraDB storage engine
- Slow query log extended statistics
- Microsecond precision in processlist
- Table elimination optimization
- PBXT storage engine
- Aria storage engine
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

MariaDB 5.1 vs MySQL

Extra features

- XtraDB storage engine
- Slow query log extended statistics
- Microsecond precision in processlist
- Table elimination optimization
- PBXT storage engine
- Aria storage engine
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

Periodic pull from upstream

3rd-party patch for MySQL

Developed at MariaDB from scratch

Out-of-tree Sun/MySQL's development

MariaDB 5.1 vs MySQL

Extra features

- **XtraDB storage engine**
- Slow query log extended statistics
- Microsecond precision in processlist
- Table elimination optimization
- PBXT storage engine
- Aria storage engine
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

XtraDB storage engine

- This is a patched version of InnoDB plugin.
- Produced by Percona, Inc (the people behind http://mysqlperformanceblog.com)
- It provides
  - Performance improvements for multi-cpu systems similar to MySQL 5.4 and 5.5
  - More diagnostic information
  - Ability to save/pre-load buffer pool (cache) contents for faster server restart or slave bring up
  - Index statistics collection fixes
  - etc, etc...

MariaDB 5.1 vs MySQL

Extra features

- XtraDB storage engine
- **Slow query log extended statistics**
- **Microsecond precision in processlist**
- Table elimination optimization
- PBXT storage engine
- Aria storage engine
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

Extended statistics in slow query log

```
# my.cnf
slow_query_log=/path/to/slow.log
+log_slow_verbosity=Query_plan
+log_slow_filter=name,name,...
+log_slow_rate_limit=n
```

'name's:

- admin
- filesort, filesort_on_disk,
- full_join,
- full_scan
- query_cache,query_cache_miss,
- tmp_table tmp_table_on_disk

Extended statistics in slow query log (2)

MySQL

```
# Time: ...
# User@Host: root[root] @ localhost []
# Query_time: 3.480293  Lock_time: 0.000754 Rows_sent: 1  Rows_examined: 10
use test;
SET timestamp=...;
select count(*) from one_k A, one_k B, ten C where A.a < B.a;
```

MariaDB

```
# Time: ...
# User@Host: root[root] @ localhost []
# Thread_id: 1  Schema: test  QC_hit: No
# Query_time: 4.605642  Lock_time: 0.000964  Rows_sent: 1  Rows_examined: 10
# Full_scan: Yes  Full_join: Yes  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
SET timestamp=...;
select count(*) from one_k A, one_k B, ten C where A.a < B.a;
```

Microsecond precision in processlist

- Based on microsec_process.patch by Percona
- Displays milliseconds with fractions in processlist
  - Useful for analyzing load of small queries

MySQL:

```
MySQL [(test)]> select * from information_schema.processlist;
+----+------+-----------+------+---------+------+--------------+-------...-+
| ID | USER | HOST      | DB   | COMMAND | TIME | STATE        | INFO      |
+----+------+-----------+------+---------+------+--------------+-------...-+
|  2 | root | localhost | test | Query   |    2 | Sending data | select    |
```

MariaDB:

```
MariaDB [(test)]> select * from information_schema.processlist;
+----+------+-----------+------+---------+------+-----------+-------...-+----------+
| ID | USER | HOST      | DB   | COMMAND | TIME | STATE     | INFO      | TIME_MS  |
+----+------+-----------+------+---------+------+-----------+-------...-+----------+
|  2 | root | localhost | test | Query   |    0 | executing | select    |    1.363 |
```

Table elimination

- Optimization for queries over highly-normalized data
- Basic idea
  Detect outer joins that have "unused" inner sides and delete those inner sides

```
SELECT tbl1.*
FROM
  tbl1 LEFT JOIN tbl2 ON tbl2.primary_key=tbl1.id
WHERE
  condition(tbl1.*)
```

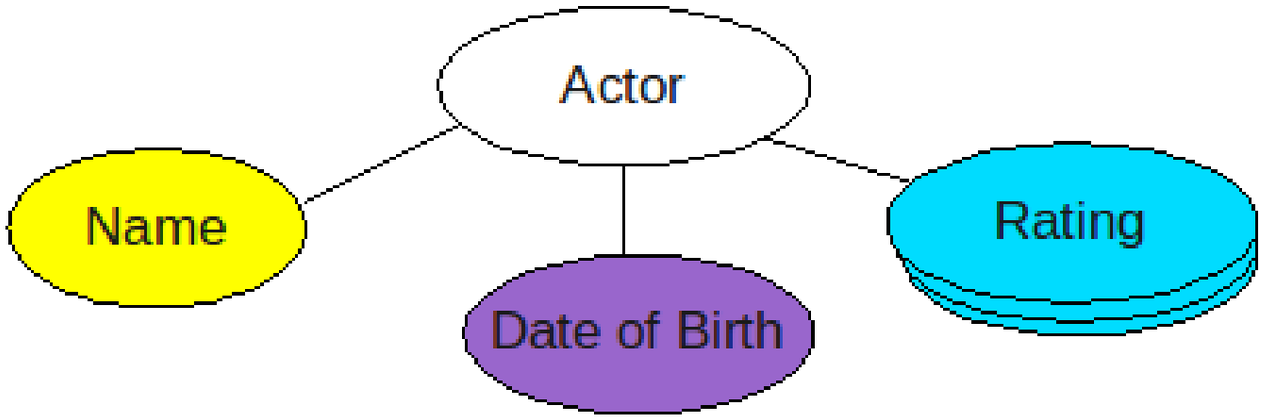
- It is guaranteed that for each record of tbl1
  - tbl2 will have not more than one match (tbl2.primary_key=..)
  - If tbl2 has no match, LEFT JOIN will generate a NULL-record

The case for table elimination

- Highly-normalized data:

  actor(name,
  date_of_birth,
  rating)

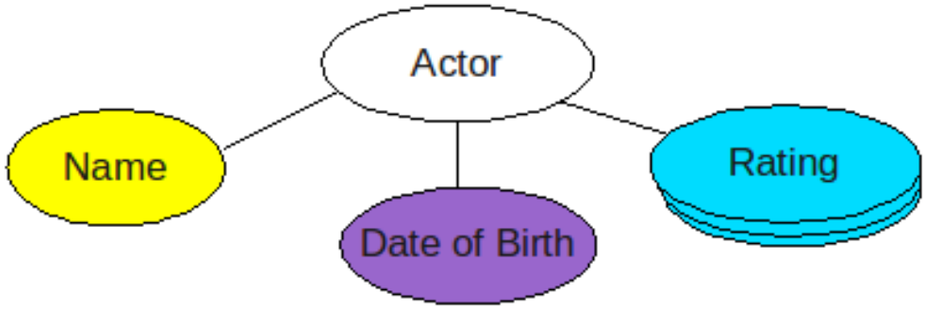
  is stored as:




```
create table  ac_anchor(AC_ID int primary key);

create table ac_name(AC_ID int, ACNAM_name char(N),
                     primary key(AC_ID));

create table ac_dob(AC_ID int, ACDOB_birthdate date,
                    primary key(AC_ID));

create table ac_rating(AC_ID int,
                       ACRAT_rating int,  ACRAT_fromdate date,
                       primary key(AC_ID, ACRAT_fromdate));
```

The case for table elimination (2)

- Then select back:




```
create view actors as select * from

select
  ac_anchor.AC_ID, ACNAM_Name, ACDOB_birthdate, ACRAT_rating
from
  ac_anchor
  left join ac_name on ac_anchor.AC_ID=ac_name.AC_ID
  left join ac_dob on ac_anchor.AC_ID=ac_dob.AC_ID
  left join ac_rating on (ac_anchor.AC_ID=ac_rating.AC_ID and
                          ac_rating.ACRAT_fromdate =
                            (select max(sub.ACRAT_fromdate)
                             from ac_rating sub
                             where sub.AC_I=ac_rating.AC_ID))

select ACRAT_rating from actors where ACNAM_name='Gary Oldman';
```

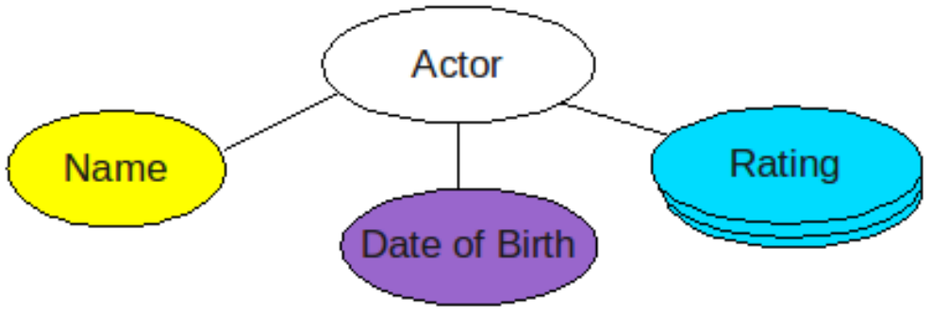
Table elimination – examples

```
explain select ACRAT_rating, ACDOB_birthdate from actors where ACNAM_name='Gary Oldman';
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
| id | select_type        | table     | type   | possible_keys | key     | key_len | ref             |-..
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
|  1 | PRIMARY            | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL            |
|  1 | PRIMARY            | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  1 | PRIMARY            | ac_dob    | eq_ref | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  1 | PRIMARY            | ac_rating | ref    | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  3 | DEPENDENT SUBQUERY | sub       | ref    | PRIMARY       | PRIMARY | 4       | ac_rating.AC_ID |
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
```

```
explain select ACRAT_rating from actors where ACNAM_name='Gary Oldman';
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
| id | select_type        | table     | type   | possible_keys | key     | key_len | ref             |-..
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
|  1 | PRIMARY            | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL            |
|  1 | PRIMARY            | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  1 | PRIMARY            | ac_rating | ref    | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  3 | DEPENDENT SUBQUERY | sub       | ref    | PRIMARY       | PRIMARY | 4       | ac_rating.AC_ID |
+----+--------------------+-----------+--------+---------------+---------+---------+-----------------+-..
```

```
explain select ACDOB_birthdate from actors where ACNAM_name='Gary Oldman';
+----+-------------+-----------+--------+---------------+---------+---------+-----------------+-..
| id | select_type | table     | type   | possible_keys | key     | key_len | ref             |-..
+----+-------------+-----------+--------+---------------+---------+---------+-----------------+-..
|  1 | PRIMARY     | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL            |
|  1 | PRIMARY     | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
|  1 | PRIMARY     | ac_dob    | eq_ref | PRIMARY       | PRIMARY | 4       | ac_anchor.AC_ID |
+----+-------------+-----------+--------+---------------+---------+---------+-----------------+-..
```

Table elimination: conclusion



- With table elimination, you can:
  - Do normalization on optional/historic data
  - Create a denormalized view with LEFT JOINs
  - Use this view and get something like “index only” scans.
- Good for 'Anchor model applications'
  - Very simple queries (just one view)
  - Allow you to instantly add/drop/change columns or tables without having to change applications.

MariaDB 5.1 vs MySQL

Extra features

- XtraDB storage engine
- Slow query log extended statistics
- Microsecond precision in processlist
- Table elimination optimization
- **PBXT storage engine**
- Aria storage engine
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

PBXT storage engine

- A transactional, ACID-compliant, MVCC-based storage engine
- Developed by PrimeBase Technologies (http://www.primebase.org)
- Design choices
  - Write-once (stores data in its “d-log” permanently)
  - Disk-based MVCC
    - => Fast commits
  - No in-place updates
  - No undo
  - File-per-table

PBXT storage engine design

- Index entry recovery
  - Indexes need not be flushed on transaction commit
  - Indexes are updated in background
- Operation IDs
  - Modifications normally require simultaneous update of cache and transaction log
  - Writer Thread uses the operations ID to sort changes
- Update clustering
  - New records are grouped so that they can be written together by the Writer
- Update consolidation
  - The Writer sorts updates from the log

Why use PBXT

- Emerging competitor to InnoDB in general high-performance OLTP
  - User-friendly developer model
- Takes advantage of SSD drives' properties
- Upcoming features
  - In-memory tables
    (already in PBXT source, not included in MariaDB yet)
  - Engine-level replication for HA setups

MariaDB 5.1 vs MySQL

Extra features

- XtraDB storage engine
- Slow query log extended statistics
- Microsecond precision in processlist
- Table elimination optimization
- PBXT storage engine
- **Aria storage engine**
- Thread pool support
- utf8_croatian_ci, ucs2_croatian_ci collations
- FederatedX storage engine.

Aria storage engine

- Originally called Maria
- Based on MyISAM code
- The goal is “transactional and nontransactional MySQL”
- Currently it is “Crash-safe MyISAM”
- Group commit in 5.2
- Used in MariaDB for SQL runtime temporary tables (instead of MyISAM)
  - Queries that need to use temporary tables and return blobs do not have to store temporary data on disk.
- Monty Program Ab's main focus is now on MariaDB, but we still plan to work on Aria in the future.

Thread pooling support

- Traditional MySQL model: one process, each client connection is served with a dedicated thread.
- Thread pooling model: serve queries from N clients with M threads.

```
# my.cnf
thread-handling=  one-thread-per-connection
thread-pool-size= 20

extra-port=#
extra-max-connections=#
```

- Known deficiencies:
  - Queries that run for long time or wait for IO or locks occupy the pool threads
  - Can be slower than traditional model in high-concurrency scenarios

Croatian _ci collations for unicode

MySQL:

```
MySQL [test]> show collation like '%croatian%';
+--------------------+---------+----+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+----+---------+----------+---------+
| latin2_croatian_ci | latin2  | 27 |         | Yes      |       1 |
| cp1250_croatian_ci | cp1250  | 44 |         | Yes      |       1 |
+--------------------+---------+----+---------+----------+---------+
2 rows in set (0.00 sec)
```

MariaDB:

```
MariaDB [test]> show collation like '%croatian%';
+--------------------+---------+-----+---------+----------+---------+
| Collation          | Charset | Id  | Default | Compiled | Sortlen |
+--------------------+---------+-----+---------+----------+---------+
| latin2_croatian_ci | latin2  |  27 |         | Yes      |       1 |
| cp1250_croatian_ci | cp1250  |  44 |         | Yes      |       1 |
| utf8_croatian_ci   | utf8    | 213 |         | Yes      |       8 |
| ucs2_croatian_ci   | ucs2    | 149 |         | Yes      |       8 |
+--------------------+---------+-----+---------+----------+---------+
4 rows in set (0.00 sec)
```

FederatedX storage engine

- MySQL has Federated storage engine disabled by default (because they have not fixed bugs in Federated).
- MariaDB instead added Federated X:
  - Developed by original author, Patrick Galbraith.
  - An improved version of MySQL's Federated storage engine
    - Bugfixes
    - Transaction support
    - Actively maintained

Compatibility with MySQL

Client libraries	Yes
Client-server protocol	Yes
Command line tool names/ locations, argument syntax, etc	Yes*
SQL dialect	Yes
Server plugins	Not tested
Replication master-slave	Yes, if both have the used features
Data directory (start one server on datadir from another)	MySQL->MariaDB: **YES** Backwards: as long as both servers support used features**

* - this means that MariaDB by default has the same port/socket/binary names as MySQL and you can't install both side-by-side
Packaging scripts install MariaDB instead of MySQL and MariaDB picks up MySQL database files

**- e.g. can't just go back if you used PBXT or utf8_croatian_ci

Infrastructure related work

- Buildbot
  - Replacement of MySQL Ab's pro priority build system
  - Before you could work for MySQL from anywhere. Now anyone can work on MySQL.
- Remove compiler warnings
  - Valgrind is your friend
- Remove mutexes
- Fully integrated and extended test suite
- Working on Random query generator and storage engine independent test suite.
- Open Knowledgebase; A free documentation for MariaDB and MySQL.

MariaDB 5.2

"Only small and/or safe features"

- Now in gamma (RC); Stable release in 1-2 months
- New features
  - Virtual columns (based on contribution by Andrey Zhakov)
  - userstatsv2 patch (Percona, Inc)
  - mysqlbinlog —rewrite-database (with support for RBR)
  - Partitioned MyISAM key cache (up to 70 % improvement)
  - Pluggable authentication
  - New engines: SPHINX and OQGRAPH
- Expected updates
  - New engines: SPIDER
  - Phone home (to make it possible to target development on used features)

Virtual columns

- Patch by Andrey Zhakov

```
CREATE TABLE t1 (
 <name> <type> [GENERATED ALWAYS] AS <expression>
[MATERIALIZED|VIRTUAL]
)
```

Future: beyond MariaDB 5.2

MariaDB 5.3 expected to be Beta soon. RC at end of year.

Features we're working on right now:

- Batched Key Access
  - Backport from MySQL 6.0, fixing known bugs
- Subquery optimizations
  - Backport from MySQL 6.0 and 6.x
  - Additional optimization improvements
- Hash joins
- Varchar and blob support for HEAP tables
- Better and pluggable replication
- **Dynamic columns**

- **We will merge latest MariaDB tree with MySQL 5.5 when MySQL 5.5 is stable enough.**

Relational databases versus others

- In the 80's the relational model replaced the hierarchical model.
- Many Object database systems was developed and researched in the 90's but never gained widespread usage (compared to SQL). Now they are getting popular again for certain tasks.
- NoSQL (a better name would be “distributed key-value store”) has been a hot topic the last 2-3 years, mainly because of their ability to spread load over many machines. (Twitter promoting Casandra, Google + BigTable, MongoDB, CouchDB, Databases in the Cloud)

Relational databases versus others

Common reasons I heard for NOT to use a relational database (SQL):

1) Performance (Both for single queries and high load)
2) Data doesn't fit the relational model (data to hard to store, manipulate or index)
3) I don't want to learn 'SQL'; (Knowing one language is good enough)

Relational databases versus others

Reasons to use an open source relational database (SQL):

1) Interfaces exists to all languages and most applications.
2) Easy to manipulate complex relations without having to make application complex.
3) Usually require much less application/framework coding (as SQL takes care of joins, order by, group etc).
4) You can choose between ACID (default) or not (MariaDB / MySQL feature to trade between safe/fast)
5) You are not bound to one vendor (It's possible to switch to another database if things doesn't work out).
6) Most performance problems can be fixed (with a “small hack”) to get similar speed as a NoSQL database.

Relational databases; Performance

Performance problems can normally be split into the following groups:

1) You want a single complex query to go faster.
2) You have a lot of users that does **a lot** of very simple queries.

Performance: Complex queries

- Bad schema (Bad indexing, wrong schema layout)
  - Fix it (It doesn't help to change model)
- Data is stored in inefficiently (not using SSD efficiently)
  - Use another storage engine or database to get better disk layout, better SSD handling or column storage.
- Can't index things properly (like text)
  - Use a storage engine / module that supports full text index (SPHINX, MyISAM or Aria)
- Lookups takes too long time
  - Store indexed data in memory or on faster device
  - Ensure that your query is optimized properly (using wrong join method can be very time consuming)
- Use external caching (memcache)

Performance: Simple queries

Parsing/optimize overhead is too big (SQL is text based)

- Use prepared statements (removes parsing, may remove some optimization phases)
- Use shortcuts like HANDLER OPEN / READ statements that doesn't have optimization overhead.
- Create a binary protocol for HANDLER statements (no parsing anymore)

- Server/Client protocol has too high overhead
  - Use an embedded SQL server like SQLite or libmysqld
- Can't handle enough users
  - Use replication, clustered (MySQL cluster) or distributed engines (ScaleDB or Calpont)

Simulating NoSQL

You can simulate a key value store in SQL:

CREATE TABLE key_value (key_name int auto_increment primary key, value blob);

By using an appropriate database or engine (MySQL Cluster) you can get similar performance as any NoSQL key-value store.

It's also relative easy to add a storage engine to MariaDB to access any key-value store (Bigtable already exists).

RDBMS doesn't solve all common problems

The (web) store problem:

All items needs: ID, Type, Price, Country, Manufacturer)

A T-Shirt has the following additional properties:

Size, color...

A computer has the following additional properties:

CPU, MHz, memory, Watt...

There is no easy way to store many different types into a relational database!

(It will not work by having one table/types as joins becomes impossible to manage).

RDBMS doesn't solve all common problems

One common solutions to this is:

- Store all the 'extra' columns in a BLOB in some format (HTML?)
  - You need a lot of extra work to manipulate the blob
  - Hard to access column data (usually done in client)
  - Overhead in storage (especially when you use HTML)
  - All values are 'text'

RDBMS doesn't solve all common problems

Another common solution:

- Create one table for all the 'extra' columns:

CREATE TABLE extra (id int auto_increment, extra_column_id char(10), value varchar(255));

INSERT INTO items set type="t-shirt", price=10;

INSERT INTO extra (NULL, LAST_INSERT_ID(), "color", "Blue), (NULL, LAST_INSERT_ID(), "Size", "M");

The problems with this approach is:

- Every access to an extra column requires a key/row lookup
- Slow performance (if database is not optimized for this)
- Big overhead in storage (especially in index)
- Risk for errors as data is not typed

Dynamic columns in MariaDB 5.3

Dynamic columns is a bridge between relational databases and non relational databases

- With dynamic columns all extra columns are stored in a packed blob, maintained by the database.
- You can instantly add more columns, remove or query them for a row.
- You can access columns in the server or retrieve the full blob to the client and manipulate it there.
- You can use virtual columns to create indexes on some values.
    - True indexes for dynamic columns is planned for later.
- Implemented trough functions to enable use by ODBC etc.
- First implementation will use integer to access columns.

Dynamic columns in MariaDB 5.3

Creating a table with a dynamic columns for the store:

CREATE TABLE item (ID int auto_increment primary_key, Type_id int, Price decimal(7,2), Country_id int, Manufacturer_id int, extra column_blob);

Here the 'column_blob' is just a normal blob that is type tagged that it will always contain a set of columns.

Dynamic columns in MariaDB 5.3

Creating/initializing a dynamic_column:

COLUMN_CREATE(column_nr, value, [column_nr,value]...)

INSERT into item (NULL, 1 /* T-shirt */, 10, 1 / * Germany /, 1 /* Nike /, COLUMN_CREATE(1 /* color /, “Blue”, 2 /* Size */, “M”));
INSERT into item (NULL, 2 /* computer /, 1 /* Germany /, 2 /* intel */, COLUMN_CREATE(3 /* cpu */, “T9400”, 5 /* MHz */, 800));

The /*..*/ is just there to make example clearer.

Dynamic columns in MariaDB 5.3

Updating a dynamic column:

COLUMN_ADD(blob,column_nr, value, column_nr,value]...)

UPDATE item SET extra=COLUMN_ADD(extra, 6 /* Memory */, 2048) WHERE id=2;

If the column already exists, it will be overwritten.

Dynamic columns in MariaDB 5.3

Deleting a dynamic column (if it exists):

```
COLUMN_DELETE(blob, column_nr, column_nr...);

UPDATE item SET extra=COLUMN_DELETE(extra, 6)
WHERE id=2;
```

Dynamic columns in MariaDB 5.3

Querying a dynamic column:

COLUMN_EXISTS(blob, column_nr);

SELECT * from item where COLUMN_EXISTS(extra,4);

Querying which columns exists:

COLUMN_LIST(blob, column_nr);

SELECT COLUMN_LIST(extra) FROM item WHERE id=1;
→ “1,2”

Dynamic columns in MariaDB 5.3

Retrieving a dynamic column:

COLUMN_GET(blob, column_nr, type);

```
SELECT id, COLUMN_GET(extra, 1 /* color*/,char(255))
from item;
→ 1  Blue
→ 2  NULL
```

You can of course also do things like:

```
SELECT * FROM item where type=1 order by
COLUMN_GET(extra,1, char(255))
```

Dynamic columns in MariaDB 5.3

When will dynamic columns be available?

The design work is in progress (you can still influence it).
Coding should start withing 3 weeks
Should be available in MariaDB 5.3 within 1-2 months.

Dynamic columns in MariaDB 5.3

How will the dynamic column be encoded?

The current proposal is:

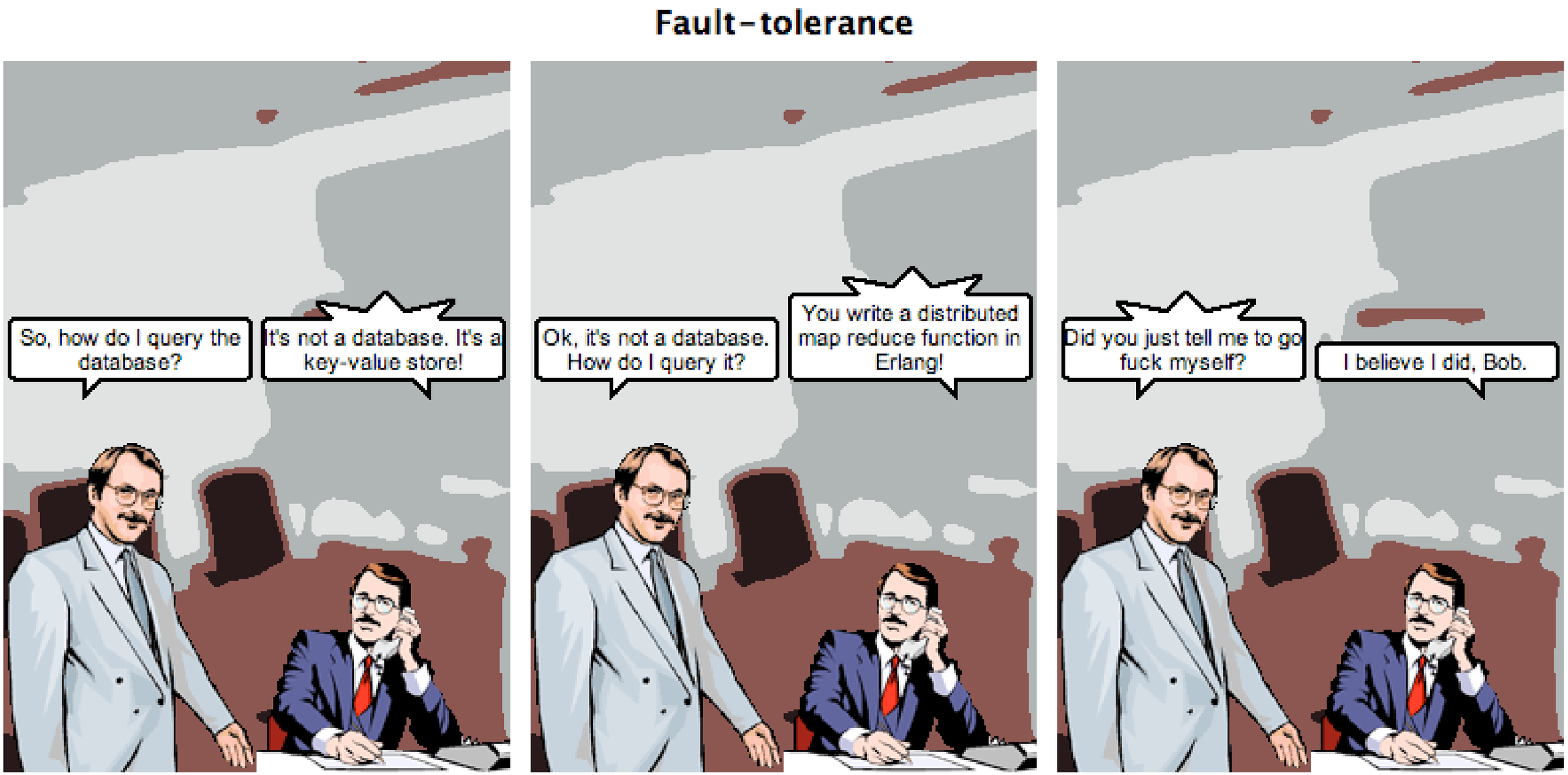
Header: <flag><number_of_columns>
Sorted index: <column_nr><offset><column_nr><offset>

Each column is stored as:
<type><length_if_string_data><data>

The data will probably use the same packing as Google protoize buffers to keep things compact.

Relational databases are easy

Don't be kept a hostage by your database!



Ensure you can access all of it easily by all needed applications!

MariaDB Unlimited Support Subscription

- Covers both MariaDB and MySQL (all stable versions)
- Network of local support partners, backed by the creators of MySQL (developers at Monty Program Ab).
- Fixed price support (now 36 000 USD), not depending on how you use MariaDB/MySQL, how many installations you have or number of beers in your refrigerator
- Unlimited monitoring via MONyog.
- Includes Knowledge Base articles, development hours etc.

Monty program Ab also sell normal server/year support contract.

If you know of anyone that could benefit from MariaDB Unlimited, contact us to get a customer finders fee!

Thanks

Q & A