The powerful toolset of the go-mysql library

Daniël van Eeden

PingCAP

Go meetup Amsterdam, 12 March 2025

Agenda

- Introduction
- Binlog
- Server
- Client

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go). Long time MySQL user.

Who am I?

Daniël van Eeden.

Working for PingCAP on TiDB (MySQL Compatible database, written in Go). Long time MySQL user.

Contributor to:

- Wireshark (MySQL protocol decoding)
- TiDB
- MySQL
- DBD::mysql (Perl)
- ...

Scope

Today I'll demonstrate various usecases of go-mysql to you.

Project History

go-mysql was created by Siddon Tang in 2014 (VP of Engineering at PingCAP).
GitHub metrics: 181 contributors, 4.7k stars.
LICENSE: MIT, BSD-3-Clause

Usage

- Spirit (Online Schema Change Tool by Cash App, presented here last year)
- gh-ost (Online Schema Change Tool by GitHub)
- LoongCollector (Observability Data Collector by Alibaba)
- YDB Federated Query (by YDB Platform)
- TiDB DM (Data Migration Tool by PingCAP)
- . . . and 593 more repositories according to GitHub

Examples

Examples are made to fit on slides.

Examples

Examples are made to fit on slides.
Error checking is left as an excercise to the reader.

Examples

Examples are made to fit on slides.
Error checking is left as an excercise to the reader.
So is adding the right import statement

```
import (
        "github.com/go-mysql-org/go-mysql/driver"
        "github.com/go-mysql-org/go-mysql/client"
        "github.com/go-mysql-org/go-mysql/mysql"
        "github.com/go-mysql-org/go-mysql/server"
)
```

go-mysql

- `go-mysql` is native Go code, no C (cgo) is used.

replication

- The MySQL binlog records changes. (so no `SELECT`, etc)
- `go-mysql` can request the binlog stream from the server, just like a MySQL replica.
- This requires the binlog to be enabled and requires the right permissions.
- Using `binlog_format=ROW` is recommended.
- `go-mysql` can also read binlog files directly.

replication

```
func main() {
        cfg := replication.BinlogSyncerConfig{
                ServerID: 123,
                Flavor:   "mysql",
                Host:     "127.0.0.1",
                Port:     3306,
                User:     "root",
                Password: "",
        }
        syncer := replication.NewBinlogSyncer(cfg)
        streamer, _ := syncer.StartSync(mysql.Position{"binlog.000001", 4})
        for {
                ev, _ := streamer.GetEvent(context.Background())

                if e, ok := ev.Event.(*replication.RowsEvent); ok {
                        for _, r := range e.Rows {
                                fmt.Printf("value of first column: %d\n", r[0])
                        }
                }
        }
}
```

replication

```
2025/03/12 09:05:19 INFO create BinlogSyncer config="{ServerID:123 Flavor:mysql
    ↪ Host:127.0.0.1 Port:3306 User:root Password: Localhost: Charset:
    ↪ SemiSyncEnabled:false RawModeEnabled:false TLSConfig:<nil> ParseTime:false
    ↪ TimestampStringLocation:UTC UseDecimal:false RecvBufferSize:0
    ↪ HeartbeatPeriod:0s ReadTimeout:0s MaxReconnectAttempts:0 DisableRetrySync:
    ↪ false VerifyChecksum:false DumpCommandFlag:0 Option:<nil> Logger:0
    ↪ xc00011e190 Dialer:0x6cd2e0 RowsEventDecodeFunc:<nil>
    ↪ TableMapOptionalMetaDecodeFunc:<nil> DiscardGTIDSet:false EventCacheCount
    ↪ :10240 SynchronousEventHandler:<nil>}"
2025/03/12 09:05:19 INFO begin to sync binlog from position position="(binlog
    ↪ .000001, 4)"
2025/03/12 09:05:19 INFO Connected to server flavor=mysql version=9.2.0
2025/03/12 09:05:19 INFO rotate to next binlog file=binlog.000001 position=4
value of first column: 1
value of first column: 2
value of first column: 3
```

canal

```
type MyEventHandler struct {
        canal.DummyEventHandler
}

func (h *MyEventHandler) OnRow(e *canal.RowsEvent) error {
        for _, r := range e.Rows {
                fmt.Printf("action=%s first_col=%#v\n", e.Action, r[0])
        }
        return nil
}

func main() {
        cfg := canal.NewDefaultConfig()
        cfg.Addr = "127.0.0.1:3306"
        cfg.User = "root"
        cfg.Dump.ExecutionPath = ""
        c, _ := canal.NewCanal(cfg)
        c.SetEventHandler(&MyEventHandler{})
        gtid, _ := mysql.ParseGTIDSet(mysql.MySQLFlavor,
                "896e7882-18fe-11ef-ab88-22222d34d411:1")
        c.StartFromGTID(gtid)
}
```

replication

```
2025/03/12 09:14:06 INFO skip dump, use last binlog replication position or GTID
    ↪ set file="" position=0 "GTID set"=896e7882-18fe-11ef-ab88-22222d34d411:1
2025/03/12 09:14:06 INFO begin to sync binlog from GTID set "GTID set"=896e7882-18
    ↪ fe-11ef-ab88-22222d34d411:1
2025/03/12 09:14:06 INFO Connected to server flavor=mysql version=9.2.0
2025/03/12 09:14:06 INFO start sync binlog at GTID set gset=896e7882-18fe-11ef-
    ↪ ab88-22222d34d411:1
2025/03/12 09:14:06 INFO rotate to next binlog file=binlog.000001 position=4
2025/03/12 09:14:06 INFO received fake rotate event nextLogName=binlog.000001
2025/03/12 09:14:06 INFO log name changed, the fake rotate event will be handled
    ↪ as a real rotate event
2025/03/12 09:14:06 INFO rotate binlog pos="(binlog.000001, 4)"
action=insert first_col="Hello Amsterdam!"
```

server

```
type DemoHandler struct {
        server.EmptyHandler
}

func (h DemoHandler) HandleQuery(query string) (*mysql.Result, error) {
        if query == `SELECT 2+2` {
                r, _ := mysql.BuildSimpleResultset(
                        []string{"result"},
                        [][]interface{}{{"5"}}, false)
                return mysql.NewResult(r), nil
        }
        return nil, nil
}

func main() {
        l, _ := net.Listen("tcp", "127.0.0.1:4000")
        c, _ := l.Accept()
        conn, _ := server.NewConn(c, "root", "", DemoHandler{})
        for {
                if err := conn.HandleCommand(); err != nil {
                        panic(err)
                }
        }
}
```

replication

```
$ mysql -u root -h 127.0.0.1 -P 4000
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10001
Server version: 8.0.11

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql-8.0.11> SELECT 2+2;
+--------+
| result |
+--------+
| 5      |
+--------+
1 row in set (0.00 sec)
```

server

Demo: gomysqlite

server

```
mysql> CREATE TABLE t1(id INT PRIMARY KEY);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES(1),(2),(3);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t1;
+------+
| id   |
+------+
| 1    |
| 2    |
| 3    |
+------+
3 rows in set (0.00 sec)
```

server

```
mysql> SELECT sqlite_version();
+------------------+
| sqlite_version() |
+------------------+
| 3.46.0           |
+------------------+
1 row in set (0.00 sec)

mysql> PRAGMA table_list;
+--------+--------------------+-------+------+------+--------+
| schema | name               | type  | ncol | wr   | strict |
+--------+--------------------+-------+------+------+--------+
| main   | t1                 | table | 1    | 0    | 0      |
| main   | sqlite_schema      | table | 5    | 0    | 0      |
| temp   | sqlite_temp_schema | table | 5    | 0    | 0      |
+--------+--------------------+-------+------+------+--------+
3 rows in set (0.00 sec)
```

server

```
mysql-8.0.11> SELECT * FROM generate_series(
    ->     '2025-02-01 00:00:00'::TIMESTAMP,
    ->     '2025-02-02 23:59:59'::TIMESTAMP,
    ->     '12 hour'
    -> );
+----------------------+
| generate_series      |
+----------------------+
| 2025-02-01T00:00:00Z |
| 2025-02-01T12:00:00Z |
| 2025-02-02T00:00:00Z |
| 2025-02-02T12:00:00Z |
+----------------------+
4 rows in set (0.00 sec)

mysql-8.0.11> SELECT VERSION();
+-------------------------------------------------------------------------------------------------
| version
|
+-------------------------------------------------------------------------------------------------
| PostgreSQL 17.2 (Debian 17.2-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debi
+-------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

server

Included:

- MySQL protocol

NOT Included:

- MySQL syntax (Both TiDB and Vitess have one in Go)
- Optimizer
- Storage Engine, etc

server

Usecases

- Security testing (clients, connectors, etc)
- Education
- Experimentation
- Making other data and/or services available

database/sql driver

`go-mysql/driver` is a pure-Go driver for database/sql.

Very similar to `github.com/go-sql-driver/mysql`.

database/sql driver

```
func main() {
        db, err := sql.Open("mysql", "root@127.0.0.1:3307/test")
        if err != nil {
                panic(err)
        }
        defer db.Close()

        var version string
        db.QueryRow("SELECT VERSION()").Scan(&version)
        fmt.Println(version)
}
```

database/sql driver

```
$ go run main.go
8.4.3
```

low-level client

```
func main() {
        conn, err := client.Connect("127.0.0.1:3307", "root", "", "test")
        if err != nil {
                panic(err)
        }
        defer conn.Quit()

        res, _ := conn.Execute("SELECT VERSION() AS ver")
        defer res.Close()

        version, _ := res.GetStringByName(0, "ver")
        fmt.Println(version)
}
```

low-level client

```
$ go run main.go
8.4.3
```

low-level client

```
func main() {
        conn, err := client.Connect("127.0.0.1:3307", "root", "", "test")
        if err != nil {
                panic(err)
        }
        defer conn.Quit()

        fmt.Println(conn.GetServerVersion())
}
```

low-level client

```
$ go run main.go
8.4.3
```

low-level client

First demo:

```
$ sudo tshark -i lo -p -f 'port 3307' -Y mysql
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback: lo'
    4 0.000494915 MySQL 143 Server Greeting  proto=10 version=8.4.3
    6 0.000550054 MySQL 210 Login Request user=root db=test
    8 0.000648718 MySQL 77 Response  OK
    9 0.000667878 MySQL 94 Request Query SELECT VERSION() AS ver
   10 0.000812981 MySQL 128 Response TABULAR Response
   11 0.000856031 MySQL 71 Request Quit
6 packets captured
```

Second demo:

```
$ sudo tshark -i lo -p -f 'port 3307' -Y mysql
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback: lo'
    4 0.001084928 MySQL 143 Server Greeting  proto=10 version=8.4.3
    6 0.001273072 MySQL 210 Login Request user=root db=test
    8 0.001595487 MySQL 77 Response  OK
    9 0.001772646 MySQL 71 Request Quit
4 packets captured
```

Demo

Demo: Materialized Views

low-level client

Setup:

```
CREATE DATABASE IF NOT EXISTS test;
USE test;
DROP TABLE IF EXISTS t,mv;
CREATE TABLE t(
        id int AUTO_INCREMENT PRIMARY KEY,
        val int NOT NULL
);
CREATE TABLE mv (
        id INT PRIMARY KEY,
        val_avg decimal(8,4) NOT NULL
);
```

Now start `demo-materialized-view`

demo-materialized-view

```
mysql-8.4.3> INSERT INTO t(val) VALUES (1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

demo-materialized-view

```
mysql-8.4.3> INSERT INTO t(val) VALUES (1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql-8.4.3> SELECT val_avg FROM mv;
+---------+
| val_avg |
+---------+
|  1.5000 |
+---------+
1 row in set (0.00 sec)
```

demo-materialized-view

```
mysql-8.4.3> INSERT INTO t(val) VALUES (1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql-8.4.3> SELECT val_avg FROM mv;
+---------+
| val_avg |
+---------+
|  1.5000 |
+---------+
1 row in set (0.00 sec)

mysql-8.4.3> SELECT AVG(val) FROM t;
+----------+
| AVG(val) |
+----------+
|   1.5000 |
+----------+
1 row in set (0.01 sec)
```

demo-materialized-view

```
mysql-8.4.3> EXPLAIN SELECT val_avg FROM mv;
+----+-------------+-------+------------+------+~+------+----------+-------+
| id | select_type | table | partitions | type |~| rows | filtered | Extra |
+----+-------------+-------+------------+------+~+------+----------+-------+
|  1 | SIMPLE      | mv    | NULL       | ALL  |~|    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+~+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql-8.4.3> EXPLAIN SELECT AVG(val) FROM t;
+----+-------------+-------+------------+------+~+------+----------+-------+
| id | select_type | table | partitions | type |~| rows | filtered | Extra |
+----+-------------+-------+------------+------+~+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ALL  |~|    2 |   100.00 | NULL  |
+----+-------------+-------+------------+------+~+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

demo-materialized-view

```
2025/01/23 14:05:33 INFO initialized gtid=94e1b69b-d8df-11ef-85d2-c6bc6abebafa
    ↪ :1-14
[2025/01/23 14:05:33] [info] binlogsyncer.go:191 create BinlogSyncer with config
    ↪ ...
[2025/01/23 14:05:33] [info] binlogsyncer.go:463 begin to sync binlog from GTID
    ↪ set 94e1b69b-d8df-11ef-85d2-c6bc6abebafa:1-14
[2025/01/23 14:05:33] [info] binlogsyncer.go:409 Connected to mysql 8.4.3 server
[2025/01/23 14:05:33] [info] binlogsyncer.go:868 rotate to (binlog.000002, 4)
2025/01/23 14:05:50 INFO updating mv total=3 count=2 avg=1.5
```

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position
3. Start to tail the binlog stream

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position
3. Start to tail the binlog stream
4. Filter out row events for the `t` table.

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position
3. Start to tail the binlog stream
4. Filter out row events for the `t` table.
5. Update the `count` and `total` based on the `INSERT` / `UPDATE` / `DELETE` (once per transaction)

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position
3. Start to tail the binlog stream
4. Filter out row events for the `t` table.
5. Update the `count` and `total` based on the `INSERT` / `UPDATE` / `DELETE` (once per transaction)
6. Calculate the `average` as `total/count`

demo-materialized-view

What this demo does is:

1. Connect to MySQL and run `SELECT COUNT(val), SUM(val) FROM t`
2. Record the GTID position
3. Start to tail the binlog stream
4. Filter out row events for the `t` table.
5. Update the `count` and `total` based on the `INSERT` / `UPDATE` / `DELETE` (once per transaction)
6. Calculate the `average` as `total/count`
7. Write the `average` to the `mv` table with a `REPLACE INTO` statement (separate connection).

MySQL and Go

- MySQL has 24-bit integers in some places. There is no `uint24` in Go.
- The MySQL has no specification. It has protocol documentation.
- The client-server protocol documentation is okay.
- The binlog events are not always well documented.

Example issues:

- The collation is a `uint8` in the protocol, but there are collations that are `>255`.
- Recently the GTID format was extended to allow for "tags". With this they introduced a new serialization method. for which people are expected to use their C++ library.
- There are cases where the client changes its behavior depending on the server version instead of the flags.

Also: We just moved from a customer logger to `log/slog`.

Questions?

Thank you!

Daniel.van.Eeden@pingcap.com

https://github.com/go-mysql-org/go-mysql
https://github.com/dveeden/go-mysql-examples