athenasql - an opensource AWS Athena database driver for Go

Developed by Uber ATG

Henry Fuheng Wu* Mingjie Lai† Matt Ranney‡

February 09, 2020

Contents

1 AthenaSQL **2**
1.1 Features . . . 2
1.2 How to set up/install/test `athenasql` . . . 2
1.2.1 Prerequisites - AWS Credentials & S3 Query Result Bucket . . . 2
1.2.2 Installation . . . 3
1.2.3 Tests . . . 3
1.2.3.1 Unit Test . . . 3
1.2.3.2 Integration Test . . . 4
1.3 How to use `athenasql` . . . 4
1.3.1 Get Started - A Simple Query . . . 4
1.3.2 Support Multiple AWS Authentication Methods . . . 5
1.3.2.1 Use AWS CLI Config For Authentication . . . 5
1.3.2.2 Use `athenasql` Config For Authentication . . . 6
1.3.3 Full Support of All Data Types . . . 7
1.3.4 Query With Workgroup and Tag . . . 8
1.3.5 Prepared Statement Support for Athena DB . . . 10
1.3.6 `DB.Exec()` and `DB.ExecContext()` . . . 11
1.3.7 Mask Columns with Specific Values . . . 12
1.3.8 Query Cancellation . . . 13
1.3.9 Missing Value Handling . . . 14
1.3.10 Read-Only Mode . . . 15
1.4 Limitations of Go/Athena SDK's and `athenasql`'s Solution . . . 16
1.4.1 Column number mismatch in `GetQueryResults` of Athena Go SDK . . . 16
1.4.1.1 `ColumnInfo` has more number of cloumns than `Rows[0].Data` . . . 16
1.4.1.2 `ColumnInfo` has cloumns but `Rows` are empty . . . 16
1.4.2 Type Loss for map, struct, array etc . . . 17
1.5 FAQ . . . 18
1.5.1 Does `athenasql` support database reconnection? . . . 18
1.5.2 Does `athenasql` support batched query? . . . 19

*henry.wu@uber.com, Uber ATG, 579 20th St, San Francisco, CA 94107
†mlai@uber.com, Uber ATG, 579 20th St, San Francisco, CA 94107
‡mranney@uber.com, Uber ATG, 3011 Smallman Street, Pittsburgh, PA 15201

1.5.3 How to use `athenasql` to get total row number of result set? 19
1.5.4 Is there any way to randomly access row with `athenasql`? 19
1.5.5 Does `athenasql` support getting the rows affected by my query? 19
1.6 Development Status: Stable 20
1.7 Contributing 20
1.7.1 `athenasql` UML Class Diagram 20
1.8 Reference 22

1 AthenaSQL

`athenasql` is a full-featured AWS Athena database driver for Go developed at Uber ATG. It provides a hassle-free way of querying AWS Athena database with Go standard library. It not only provides basic features of Athena Go SDK, but addresses some SDK's limitation, improves and extends it. It also includes advanced features like Athena workgroup and tagging creation, driver read-only mode and so on.

1.1 Features

Except the basic features provided by Go `database/sql` like error handling, database pool and reconnection, `athenasql` supports the following features out of box:

- Support multiple AWS authorization methods (1.3.2)
- Full support of Athena Basic Data Types
- Full support of Athena Advanced Type for queries with Geospatial identifiers, ML and UDFs (1.3.3)
- Full support of *ALL* Athena Query Statements, including DDL, DML and UTILITY
- Support newly added `INSERT INTO...VALUES`
- Athena workgroup and tagging support including remote workgroup creation (1.3.4)
- Go sql's Prepared statement support (1.3.5)
- Go sql's `DB.Exec()` and `db.ExecContext()` support(1.3.6)
- Query cancelling support (1.3.8)
- Mask columns with specific values (1.3.7)
- Database missing value handling (1.3.9)
- Read-Only mode (1.3.10)

1.2 How to set up/install/test `athenasql`

1.2.1 Prerequisites - AWS Credentials & S3 Query Result Bucket

To be able to query AWS Athena, you need to have an AWS account at Amazon AWS's website. To give it a shot, a free tier account is enough. You also need to have a pair of AWS `access key ID` and `secret access key`. You can get it from AWS Security Credentials section of Identity and Access Management (IAM). If you don't have one, please create it. The following is a screenshot from my temporary free tier account:

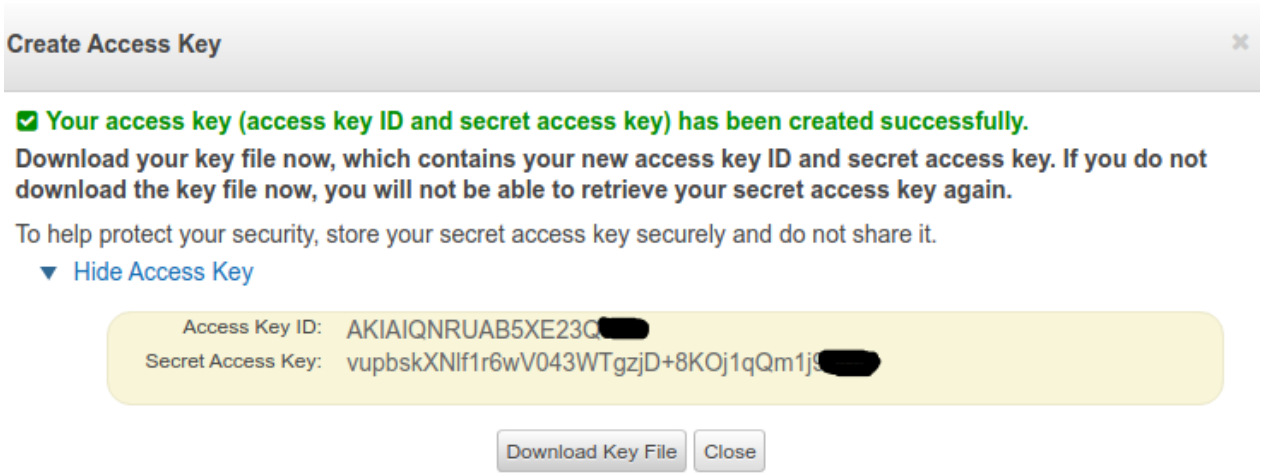

Figure 1: How to create AWS credentials

In addition to AWS credentials, you also need an s3 bucket to store query result. Just go to AWS S3 web console page to create one. In the examples below, the s3 bucket I use is `s3://henrywuqueryresults/`.

In most cases, you need the following 4 prerequisites [1]:

- S3 Output bucket
- `access key ID`
- `secret access key`
- AWS region

For more details on `athenasql`'s support on AWS credentials & S3 query result bucket, please refer to section Support Multiple AWS Authorization Methods.

1.2.2 Installation

```
go get github.com/uber/athenasql
```

1.2.3 Tests

We provide unit tests and integration tests in the codebase.

1.2.3.1 Unit Test

All the unit tests are self-contained and passed in no-internet environment. Test coverage is `100%`.

```
$ cd $GOPATH/src/github.com/uber/athenasql/go
$ go test -coverprofile=coverage.out github.com/uber/athenasql/go  && \
 go tool cover -func=coverage.out |grep -v 100.0%
ok      github.com/uber/athenasql/go    9.255s  coverage: 100.0% of statements
```

[1] AWS Security Token Service(STS) does authentication with no need of usual credentials.

1.2.3.2 Integration Test

All integration tests are under `examples` folder. Please make sure all prerequisites are met so that you can run the code on your own machine.

All the code snippets in `examples` folder are fully tested in our machines. For example, to run some stress and crash test, you can use `examples/perf/concurrency.go`. Build it first:

```
$cd $GOPATH/src/github.com/uber/athenasql
$go build examples/perf/concurrency.go
```

Run it, wait for some output but not all, and unplug your network cable:

```
$./concurrency > examples/perf/concurrency.output.`date +"%Y-%m-%d-%H-%M-%S"`.log
58,13,53,54,78,96,32,48,40,11,35,31,65,61,1,73,74,22,34,49,80,5,69,37,0,79,
2020/02/09 13:49:29 error [38]RequestError: send request failed
caused by: Post https://athena.us-east-1.amazonaws.com/: dial tcp:
lookup athena.us-east-1.amazonaws.com: no such host
...
2020/02/09 13:49:29 error [89]RequestError: send request failed
caused by: Post https://athena.us-east-1.amazonaws.com/: dial tcp:
lookup athena.us-east-1.amazonaws.com: no such host
```

You can see `RequestError` is thrown out from the code. The active Athena queries failed because the network is down. Now re-plugin your cable and wait for network coming back, you can see the program automatically reconnects to Athena, and resumes to output data correctly:

```
72,25,92,98,15,93,41,7,8,90,81,56,66,2,18,84,87,63,
44,45,82,99,86,3,52,76,71,16,39,67,23,12,42,17,4,
```

1.3 How to use `athenasql`

`athenasql` is very easy to use. What you need to do it to import it in your code and then use the standard Go `database/sql` as usual.

```
import athenasql "github.com/uber/athenasql/go"
```

The following are coding examples to demonstrate `athenasql`'s features and how you should use `athenasql` in youe Go application. Please be noted the code is for demonstration purpose only, so please follow your own coding style or best practice if necessary.

1.3.1 Get Started - A Simple Query

The following is the simplest example for demonstration purpose. The source code is available at dml_select_simple.go[2].

```
package main

import (
```

[2]All the code snippets are tested and stored at github(https://github.com/uber/athenasql/tree/master/examples). Please check the link for latest update.

```
    "database/sql"
    "fmt"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // Step 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    // Step 2. Open Connection.
    db, _ := sql.Open(drv.DBDriverName, conf.Stringify())
    // Step 3. Query and print results
    var url string
    _ = db.QueryRow("SELECT url from sampledb.elb_logs limit 1").Scan(&url)
    fmt.Println(url)
}
```

To make it work for you, please replace `OutputBucket`, `Region`, `AccessID` and `SecretAccessKey` with your own values. `sampledb` is provided by Amazon so you don't have to worry about it.

To Build it:

```
$ go build examples/query/dml_select_simple.go
```

Run it and you can see output like:

```
$ ./dml_select_simple
https://www.example.com/articles/553
```

1.3.2 Support Multiple AWS Authentication Methods

`athenasql` uses access keys(Access Key ID and Secret Access Key) to sign programmatic requests to AWS. When if the AWS_SDK_LOAD_CONFIG environment variable was set, `athenasql` uses `Shared Config`, respects AWS CLI Configuration and Credential File Settings and gives it even higher priority over the values set in `athenasql.Config`.

1.3.2.1 Use AWS CLI Config For Authentication

When environment variable `AWS_SDK_LOAD_CONFIG` is set, it will read `aws_access_key_id`(AccessID) and `aws_secret_access_key`(SecretAccessKey) from `~/.aws/credentials`, `region` from `~/.aws/config`. For details about `~/.aws/credentials` and `~/.aws/config`, please check here.

But you still need to specify correct `OutputBucket` in `athenasql.Config` because it is not in the AWS client config.

`OutputBucket` is critical in Athena. Even if you have a default value set in Athean web console, you must pass one programmatically or you will get error: `No output location provided. An output location is required either through the Workgroup result configuration setting or as an API input.`

The sample code below enforces AWS_SDK_LOAD_CONFIG is set, so `athenasql`'s AWS Session will be created from the configuration values from the shared config (~/.aws/config) and shared credentials (~/.aws/credentials) files. Even if we pass all dummy values as parameters in `NewDefaultConfig()` except `OutputBucket`, they are overridden by the values in AWS CLI config files, so it doesn't really matter.

```
// To use AWS CLI's Config for authentication
func useAWSCLIConfigForAuth() {
    os.Setenv("AWS_SDK_LOAD_CONFIG", "1")
    // 1. Set AWS Credential in Driver Config.
    conf, err := drv.NewDefaultConfig(secret.OutputBucketProd, drv.DummyRegion,
        drv.DummyAccessID, drv.DummySecretAccessKey)
    if err != nil {
        return
    }
    // 2. Open Connection.
    db, _ := sql.Open(drv.DriverName, conf.Stringify())
    // 3. Query and print results
    var i int
    _ = db.QueryRow("SELECT 456").Scan(&i)
    println("with AWS CLI Config:", i)
    os.Unsetenv("AWS_SDK_LOAD_CONFIG")
}
```

If your AWS CLI setting is valid like mine, this function should output:

```
with AWS CLI Config: 456
```

1.3.2.2 Use `athenasql` Config For Authentication

When environment variable `AWS_SDK_LOAD_CONFIG` is NOT set, you need to pass valid(NOT dummy) `region`, `accessID`, `secretAccessKey` into `athenasql.NewDefaultConfig()`, in addition to `outputBucket`.

The sample code below ensure `AWS_SDK_LOAD_CONFIG` is not set, then pass four valid parameters into `NewDefaultConfig()`:

```
// To use athenasql's Config for authentication
func useAthenaSQLConfigForAuth() {
    os.Unsetenv("AWS_SDK_LOAD_CONFIG")
    // 1. Set AWS Credential in Driver Config.
    conf, err := drv.NewDefaultConfig(secret.OutputBucketDev, secret.Region,
        secret.AccessID, secret.SecretAccessKey)
    if err != nil {
        return
    }
    // 2. Open Connection.
    db, _ := sql.Open(drv.DriverName, conf.Stringify())
    // 3. Query and print results
    var i int
```

```
    _ = db.QueryRow("SELECT 123").Scan(&i)
    println("with AthenaSQL Config:", i)
}
```

The sample output:

```
with AthenaSQL Config: 123
```

The full code is here at examples/auth.go.

1.3.3 Full Support of All Data Types

As we said, `athenasql` supports all Athena data types. In the following sample code, we use an SQL statement to `SELECT` som simple data of all the advanced types and then print them out.

```
package main

import (
    "context"
    "database/sql"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, err := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    if err != nil {
        panic(err)
    }
    // 2. Open Connection.
    dsn := conf.Stringify()
    db, _ := sql.Open(drv.DBDriverName, dsn)
    // 3. Query and print results
    query := "SELECT JSON '\"Hello Athena\"', " +
        "ST_POINT(-74.006801, 40.70522), " +
        "ROW(1, 2.0),  INTERVAL '2' DAY, " +
        "INTERVAL '3' MONTH, " +
        "TIME '01:02:03.456', " +
        "TIME '01:02:03.456 America/Los_Angeles', " +
        "TIMESTAMP '2001-08-22 03:04:05.321 America/Los_Angeles';"
    rows, err := db.Query(query)
    if err != nil {
        panic(err)
    }
    defer rows.Close()
    drv.PrintColumnRows(rows)
}
```

Sample output:

```
"Hello Athena",00 00 00 00 01 01 00 00 00 20 25 76 6d 6f 80 52 c0 18 3e 22 a6 44 5a 44 40,
{field0=1, field1=2.0},2 00:00:00.000,0-3,0000-01-01T01:02:03.456-07:52,
0000-01-01T01:02:03.456-07:52,2001-08-22T03:04:05.321-07:00
```

we can see `athenasql` can handle all these advanced types correctly.

1.3.4 Query With Workgroup and Tag

`athenasql` supports workgroup and tagging features of Athena. When you query Athena, you can specify the workgroup and tags attached with your query. Resource/cost tagging are based on workgroup. If the workgroup doesn't exist , by default it will be created programmatically.

If you want to disable programmatically creating workgroup and tags, you need to explicitly call:

```
Config.SetWGRemoteCreationAllowed(false)
```

In this case, you need to make sure the workgroup you specifies must exist, or you will get error. An example is like below:

```
package main

import (
    "database/sql"
    "fmt"
    "log"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    wgTags := drv.NewWGTags()
    wgTags.AddTag("Uber User", "henry.wu@uber.com")
    wgTags.AddTag("Uber ID", "123456")
    wgTags.AddTag("Uber Role", "SDE")
    // Specify that workgroup `henry_wu` is used for the following query
    wg := drv.NewWG("henry_wu", nil, wgTags)
    conf.SetWorkGroup(wg)
    // comment out the line below to allow remote workgroup creation and
    // the query will be successful!!!
    conf.SetWGRemoteCreationAllowed(false)
    // 2. Open Connection.
    dsn := conf.Stringify()
    db, _ := sql.Open(drv.DBDriverName, dsn)
    // 3. Query and print results
    rows, err := db.Query("select url from sampledb.elb_logs limit 3")
    if err != nil {
        log.Fatal(err)
        return
```

```
    }
    defer rows.Close()

    var url string
    for rows.Next() {
        if err := rows.Scan(&url); err != nil {
            log.Fatal(err)
        }
        fmt.Println(url)
    }
}
```

But I don't have a workgroup named `henry_wu` in AWS Athena, so I got sample output:

```
2020/01/20 15:29:52 Workgroup henry_wu doesn't exist and workgroup remote creation
 is disabled.
```

After commenting out `conf.SetWGRemoteCreationAllowed(false)` at line 27, the output becomes:

```
https://www.example.com/articles/553
http://www.example.com/images/501
https://www.example.com/images/183
```

and I can see a new workgroup named `henry_wu` is created in AWS Athena console: https://us-east-2.console.aws .amazon.com/athena/workgroups/home

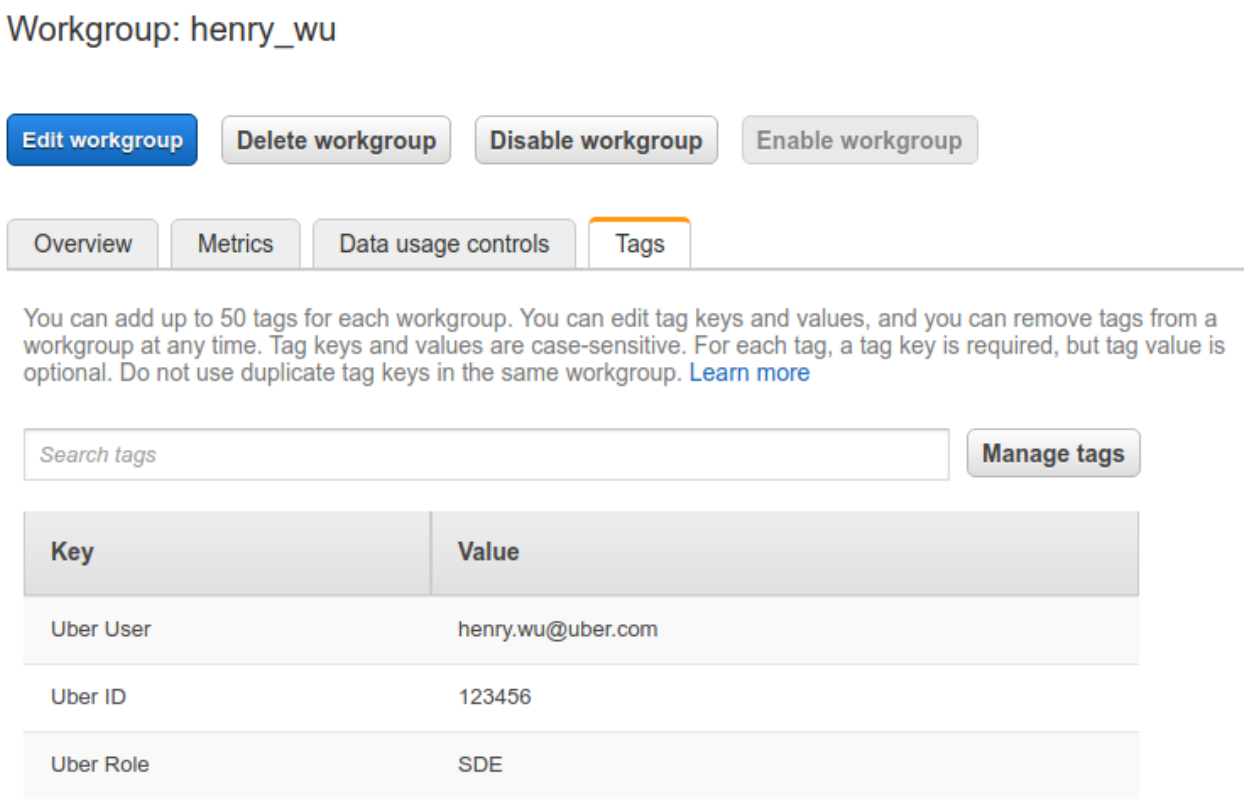

Figure 2: Athena Workgroup and Tags Automatic Creation

1.3.5 Prepared Statement Support for Athena DB

Athena doesn't support prepared statement originally. However, it could be very helpful in some scenarios like where part of the query is from user input. `athenasql` supports prepared statements to help you to deal with those scenarios. An example is as follows:

```
package main

import (
    "database/sql"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    // 2. Open Connection.
    db, _ := sql.Open(drv.DBDriverName, conf.Stringify())
    // 3. Prepared Statement
    statement, err := db.Prepare("CREATE TABLE sampledb.urls AS " +
        "SELECT url FROM sampledb.elb_logs where request_ip=? limit ?")
    if err != nil {
        panic(err)
    }
    // 4. Execute prepared Statement
    if result, e := statement.Exec("244.157.42.179", 2); e == nil {
        if rowsAffected, err := result.RowsAffected(); err == nil {
            println(rowsAffected)
        }
    }
}
```

Sample output:

```
2
```

You can also use the `?` syntax with `DB.Query()` or `DB.Exec()` directly.

```
    rows, err := db.Query("SELECT request_timestamp,elb_name "+
        "from sampledb.elb_logs where url=? limit 1",
        "https://www.example.com/jobs/878")
    if err != nil {
        return
    }
    drv.PrintColumnRows(rows)
```

Sample Output:

```
request_timestamp,elb_name
2015-01-06T04:03:01.351843Z,elb_demo_006
```

1.3.6 DB.Exec() and DB.ExecContext()

According to Go source code, DB.Exec() and DB.ExecContext() execute a query that doesn't return rows, such as an INSERT or UPDATE. It's true that you can use DB.Exec() and DB.Query() interchangeably to execute the same SQL statements. [3] However, the two methods are for different use cases and return different types of results. According to Go database/sql library, the result returned from DB.Exec() can tell you how many rows were affected by the query and the last inserted ID for INSERT INTO statement, which is always *-1* for Athena because auto-increment primary key feature is not supported by Athena.[4] In contrast, DB.Query() will return a sql.Rows object which includes all columns and rows details.

When the only concern is if the execution is successful or not, DB.Exec() is preferred to DB.Query() . The best coding practice is:

```
if _, err := DB.Exec(`<SQL_STATEMENT>`); err != nil {
    log_or_panic(err)
}
```

In cases of INSERT INTO, CTAG and CVAS, you may want to know when the execution is successful how many rows are affected by your query. Then you can use result.RowsAffected() as demonstrated in the following example:

```
package main

import (
    "context"
    "database/sql"
    "fmt"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    var conf *drv.Config
    var err error
    if conf, err = drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey"); err != nil {
        panic(err)
    }
    // 2. Open Connection.
    db, _ := sql.Open(drv.DBDriverName, conf.Stringify())
    // 3. Execute and print results
    if _, err = db.ExecContext(context.Background(),
        "DROP TABLE IF EXISTS sampledb.urls"); err != nil {
        panic(err)
    }
```

[3] DB.ExecContext() and DB.QueryContext() are the same way.

[4] Even INDEX is not supported in Athena. Please refer to: https://docs.aws.amazon.com/athena/latest/ug/unsupported-ddl.html

```
    var result sql.Result
    if result, err = db.Exec("CREATE TABLE sampledb.urls AS "+
        "SELECT url FROM sampledb.elb_logs where request_ip=? limit ?",
        "244.157.42.179", 1); err != nil {
        panic(err)
    }
    fmt.Println(result.RowsAffected())

    if result, err = db.Exec("INSERT INTO sampledb.urls VALUES (?),(?),(?)",
        "abc", "efg", "xyz"); err != nil {
        panic(err)
    }
    fmt.Println(result.RowsAffected())
    fmt.Println(result.LastInsertId()) // not supported by Athena
}
```

Sample output:

```
1
3
```

1.3.7 Mask Columns with Specific Values

Sometimes, database contains sensitive information and you may need to mask columns with specific values. If you don't want to display some columns, you can mask them by calling:

`Config.SetMaskedColumnValue("columnName", "maskValue")`

For example, if you want to mask all rows of column `password`, you can specify:

`Config.SetMaskedColumnValue("password", "xxx")`

Then all the passwords will be displayed as `xxx` in the query result set. The following is an example to mask column `url` in the result set.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    conf.SetMaskedColumnValue("url", "xxx")
    // 2. Open Connection.
```

```
    dsn := conf.Stringify()
    db, _ := sql.Open(drv.DBDriverName, dsn)
    // 3. Query and print results
    rows, err := db.Query("select request_timestamp, url from " +
        "sampledb.elb_logs limit 3")
    if err != nil {
        log.Fatal(err)
        return
    }
    defer rows.Close()

    var requestTimestamp string
    var url string
    for rows.Next() {
        if err := rows.Scan(&requestTimestamp, &url); err != nil {
            log.Fatal(err)
        }
        fmt.Println(requestTimestamp + "," + url)
    }
}
```

Sample Output:

```
2015-01-03T12:00:00.516940Z,xxx
2015-01-03T12:00:00.902953Z,xxx
2015-01-03T12:00:01.206255Z,xxx
```

1.3.8 Query Cancellation

AWS Athena is priced upon the data size it scanned. To save money, `athenasql` supports query cancellation. In the following example, the query is cancelled if it is not complete after 2 seconds.

```
package main

import (
    "context"
    "database/sql"
    "fmt"
    "log"
    "time"

    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
```

```
    // 2. Open Connection.
    dsn := conf.Stringify()
    db, _ := sql.Open(drv.DBDriverName, dsn)
    // 3. Query cancellation after 2 seconds
    ctx, _ := context.WithTimeout(context.Background(), 2*time.Second)
    rows, err := db.QueryContext(ctx, "select count(*) from sampledb.elb_logs")
    if err != nil {
        log.Fatal(err)
        return
    }
    defer rows.Close()

    var requestTimestamp string
    var url string
    for rows.Next() {
        if err := rows.Scan(&requestTimestamp, &url); err != nil {
            log.Fatal(err)
        }
        fmt.Println(requestTimestamp + "," + url)
    }
}
```

Sample Output:

```
2020/01/20 15:28:35 context deadline exceeded
```

1.3.9 Missing Value Handling

It is common to have missing values in S3 file, or Athena DB. When this happens, you can specify if you want to use `empty string` or `default data` as the missing value, whichever is better to facilitate your data processing. The default data for Athena column type are defined as below:

```
func (r *Rows) getDefaultValueForColumnType(athenaType string) interface{} {
    switch athenaType {
    case "tinyint", "smallint", "integer", "bigint":
        return 0
    case "boolean":
        return false
    case "float", "double", "real":
        return 0.0
    case "date", "time", "time with time zone", "timestamp",
        "timestamp with time zone":
        return time.Time{}
    default:
        return ""
    }
}
```

By default, we use empty string to replace missing values and empty string is preferred to default

data. To use `default data`, you have to explicitly call:

```
Config.SetMissingAsEmptyString(false)
Config.SetMissingAsDefault(true)
```

But if you are strict with your data integrity and want an error raised when data are missing, you can set both of them `false`.

1.3.10 Read-Only Mode

When read-only mode is enabled in `athenasql`, it only allows retrieving information from Athena database. Any writing and modification to the database will raise an error. This is useful in some cases. By default, read-only mode is disabled. To enable it, you need to explicitly call:

```
Config.SetReadOnly(true)
```

The following is one example. It enables read-only mode in line 19, but tries to create a new table with CTAS statement. It ends up with raising an error.

```
package main

import (
    "context"
    "database/sql"
    "log"
    drv "github.com/uber/athenasql/go"
)

func main() {
    // 1. Set AWS Credential in Driver Config.
    conf, _ := drv.NewDefaultConfig("s3://henrywuqueryresults/",
        "us-east-2", "DummyAccessID", "DummySecretAccessKey")
    conf.SetReadOnly(true)

    // 2. Open Connection.
    dsn := conf.Stringify()
    db, _ := sql.Open(drv.DBDriverName, dsn)
    // 3. Create Table with CTAS statement
    rows, err := db.QueryContext(context.Background(),
      "CREATE TABLE sampledb.elb_logs_new AS " +
        "SELECT * FROM sampledb.elb_logs limit 10;")
    if err != nil {
        log.Fatal(err)
        return
    }
    defer rows.Close()
}
```

Sample Output:

```
2020/01/26 01:10:28 writing to Athena database is disallowed in read-only mode
```

1.4 Limitations of Go/Athena SDK's and `athenasql`'s Solution

1.4.1 Column number mismatch in `GetQueryResults` of Athena Go SDK

1.4.1.1 `ColumnInfo` has more number of cloumns than `Rows[0].Data`

Affected Statements: DESCRIBE TABLE/VIEW, SHOW SCHEMA/TABLE/...

- Sample Query:

```
DESC sampledb.elb_logs
```

- Analysis:

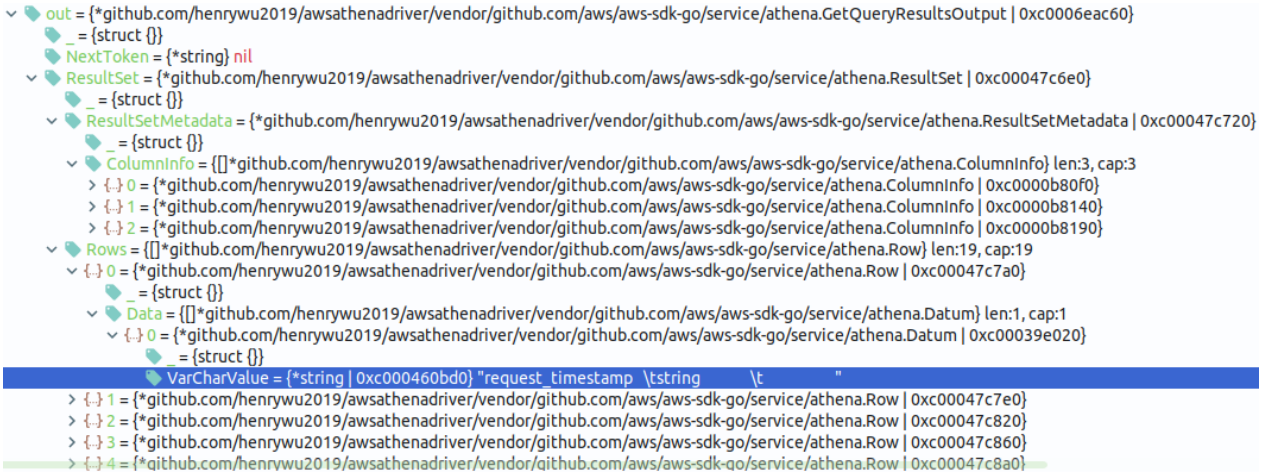

Figure 3: Column number mismatch issue example 1

We can see there are 3 columns according to `ColumnInfo` under `ResultSetMetadata`. But in the first row `Rows[0]`, we see there is only 1 field: `"elb_name \tstring \t "`. I would imagine there could have been 3 items in the `Data[0]`, but somehow the code author doesn't split it with tab(`\t`), so it ends up with only 1 item. The same issue happens for `SHOW` statement.

For more sample code, please check util_desc_table.go, util_desc_view.go, and util_show.go.

- `awsathendriver`'s Solution:

`athenasql` fixes this issue by splitting `Rows[0].Data[0]` string with tab, and replace the original row with a new row which has the same number of data with columns.

1.4.1.2 `ColumnInfo` has cloumns but `Rows` are empty

Affected Statements: `CTAS`, CVAS, INSERT INTO

Sample Query:

```
CREATE TABLE sampledb.elb_logs_copy WITH (
    format = 'TEXTFILE',
```

```
    external_location = 's3://external-location-henrywu/elb_logs_copy',
    partitioned_by = ARRAY['ssl_protocol'])
AS SELECT * FROM sampledb.elb_logs
```

Analysis:

Figure 4: Column number mismatch issue example 2

In the above CTAS statement, we see there is one column of type `bigint` named `"rows"` in the resultset, but `ResultSet.Rows` is empty. Since there is no row, that one column doesn't make sense, or at least is confusing. The same issue happens for `INSERT INTO` statement.

- `awsathendriver`'s Solution:

Because this issue happens only in statements CTAS, `CVAS`, and `INSERT INTO`, where `UpdateCount` is always valid and is the only meaningful information returned from Athena, `athenasql` sets `UpdateCount` as the value of the returned row.

For more sample code, please check ddl_ctas.go, ddl_cvas.go, and dml_insert_into.go.

1.4.2 Type Loss for map, struct, array etc

One of Athena Go SDK's limitations is the type information could be lost after querying. I think there are two reasons for this type infromation loss.

The first reason is Athena SDK doesn't provide the full type information for complex type data. It assumes the application developers know the data schema and should take the responsibility of data serialization.

To dig into the code, all query results are stored in data structure `ResultSet`. From the UML class graph of `ResultSet` below, we can see the type information are stored in `ColumnInfo`'s pointer to string variable `Type`, which is only a type name of data type, not containing any type metadata. For example, querying a map of `string->boolean` will return the type name `map`, but you cannot find the information `string->boolean` in the `ResultSet`. For simple type like `integer`, `boolean` or `string`, it is sufficient to serialize them to Go type, but for more complex types like `array`, `struct`, `map` or nested types, the type information is lost here.

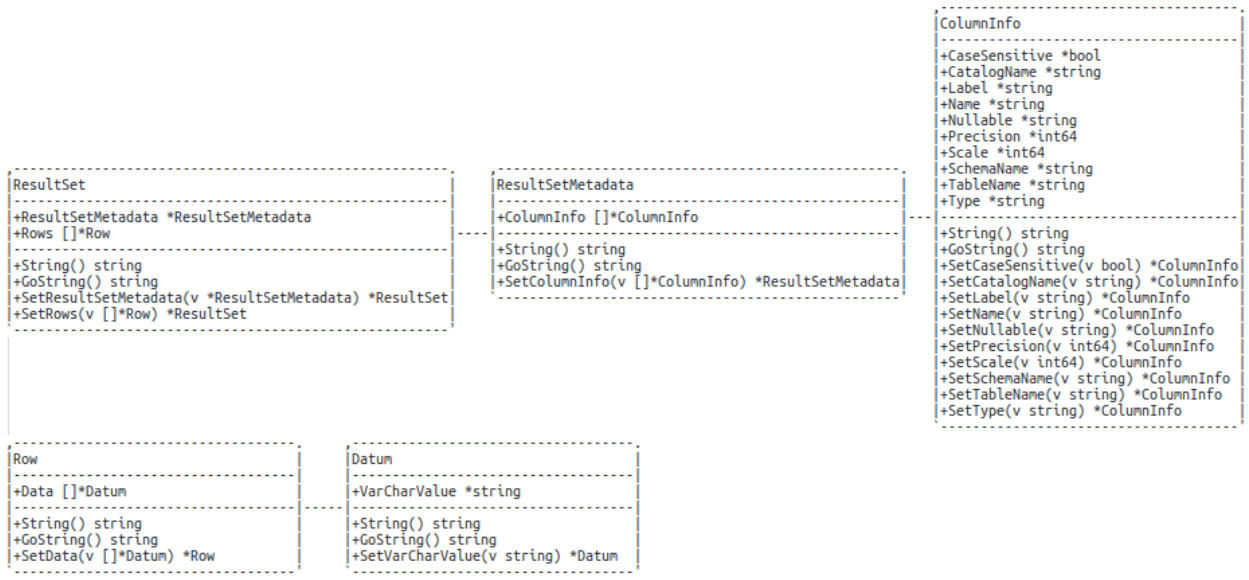

Figure 5: UML class graph of `ResultSet`

The second reason is the difference between Athena data type and Go data type. Some Athena builtin data type like `Row`, `DECIMAL(p, s)`, `varbinary`, `interval year to month` are not supported in Go standard library. Therefore, there is no way to serialize them in driver level.

- `awsathendriver`'s Solution:

For data types: `array`, `map`, `json`, `char`, `varchar`, `varbinary`, `row`, `string`, `binary`, `struct`, `interval year to month`, `interval day to second`, `decimal`, `awsathendriver` returns the string representation of the data. The developers can firstly retrieve the string representation, and then serialize to user defined type on their own.

For time and date types: `date`, `time`, `time with time zone`, `timestamp`, `timestamp with time zone`, `awsathendriver` returns Go's `time.Time`.

Some sample code are available at dml_select_array.go, dml_select_map.go, dml_select_time.go.

1.5 FAQ

The following is a collection of questions from our software developers and data scientists.

1.5.1 Does `athenasql` support database reconnection?

Yes. `database/sql` maintains a connection pool internally and handles connection pooling, reconnecting, and retry logic for you. One pitfall of writing Go sql application is cluttering the code with error-handling and retry. I tested in my application with `athenasql` by turning off and on Wifi and VPN, it works very well with database reconnection.

1.5.2 Does `athenasql` support batched query?

No. `athenasql` is an implementation of `sql.driver` in Go `database/sql`, where there is no batch query support. There might be some workaround for some specific case though. For instance, if you want to insert many rows, you can use db.Exec by replacing multiple inserts with one insert and multiple VALUES.

1.5.3 How to use `athenasql` to get total row number of result set?

You have to use `rows.Next()` to iterate all rows and use a counter to get row number. It is because Go `database/sql` was designed in a streaming query way with big data considered. That is why it only supports using `Next()` to iterate. So there is no way for random access of row. In Athena case, we only have random access of all the rows within one result page as the picture shown below:

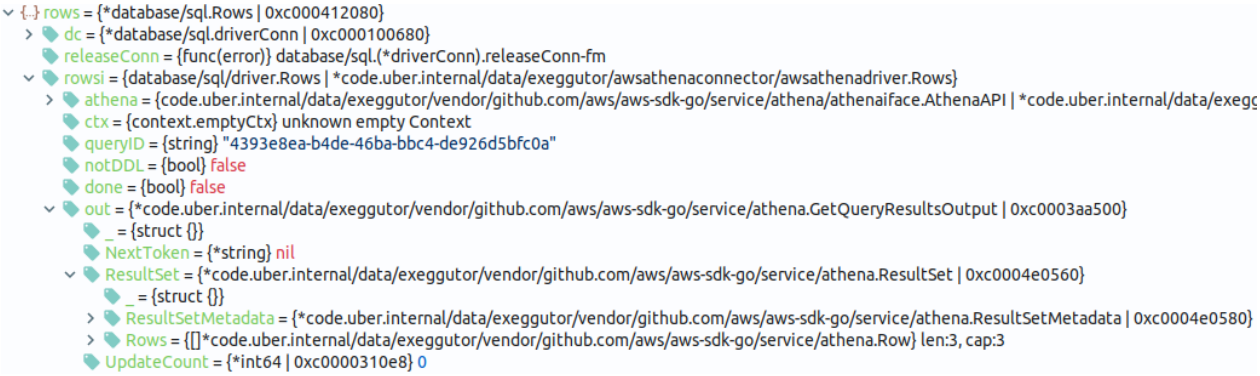

Figure 6: Encapsulation of driver.Rows in sql.Rows

But due to encapsulation, more sepcifically the `rowsi` is *private*, we cannot access it directly like when we using Athena Go SDK. We have to use `Next()` to access it one by one.

1.5.4 Is there any way to randomly access row with `athenasql`?

No. The reason is the same as answer to the previous question.

1.5.5 Does `athenasql` support getting the rows affected by my query?

To put it simple, YES. But there is some limitation and best practice to follow.

The recommended way is to use `DB.Exec()` to get it. Please refer to 1.3.6.

You can get it with `DB.Query()` too. In the returned `ResultSet`, there is an `UpdateCount` member variable. If the query is one of `CTAS`, `CVAS` and `INSERT INTO`, `UpdateCount` will contain meaningful value. The result will be of a one row and one column. The column name is `rows`, and the row is an `int`, which is exactly `UpdateCount`. I would suggest to use `QueryRow` or `QueryRowContext` since it is a one-row result. By the way, the document for `GetQueryResults` seems not very accurate.

In practice, not only `CTAS` statement but also `CVAS` and `INSERT INTO` will make a meaningful `UpdateCount`.

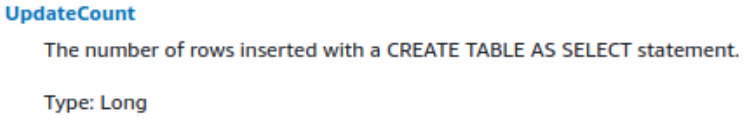

Figure 7: UpdateCount for CTAS, VTAS, and INSERT INTO

1.6 Development Status: Stable

All APIs are finalized, and no breaking changes will be made in the 1.x series of releases.

This library is v1 and follows SemVer strictly.

1.7 Contributing

We encourage and support an active, healthy community of contributors — including you! Details are in the contribution guide and the code of conduct. The athenasql maintainers keep an eye on issues and pull requests, but you can also report any negative conduct to **oss-conduct@uber.com**. That email list is a private, safe space; even the athenasql maintainers don't have access, so don't hesitate to hold us to a high standard.

1.7.1 `athenasql` UML Class Diagram

The following is `athenasql` Package's UML Class Diagram which may help you to understand the code. You can also check the reference section (1.8) for some useful materials.

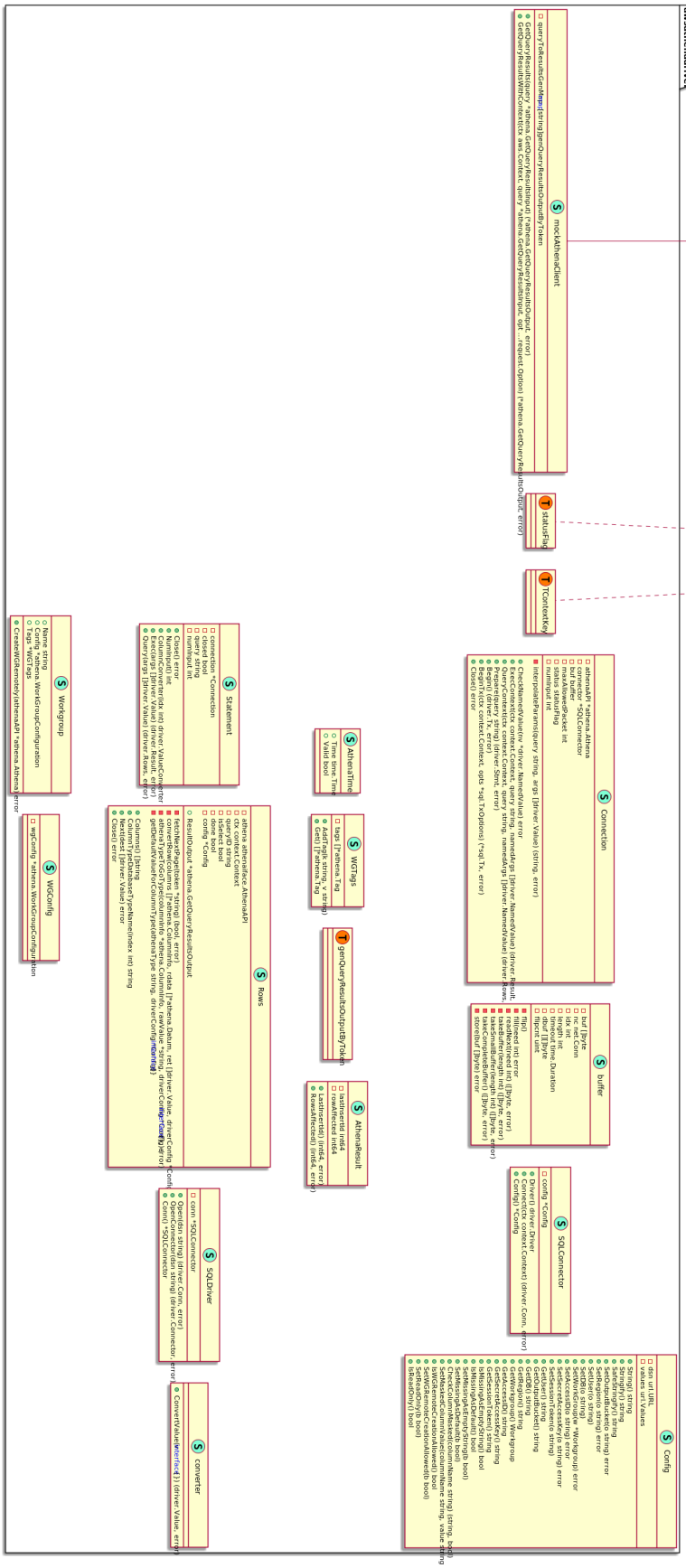

Figure 8: `athenasql` Package's UML Class Diagram

1.8 Reference

- Amazon Athena User Guide
- Amazon Athena API Reference - Describes the Athena API operations in detail.
- Amazon Athena Go Doc
- Data type mappings that the JDBC driver supports between Athena, JDBC, and Java
- Service Quotas
- Go sql connection pool
- Common Pitfalls When Using database/sql in Go
- Implement Sql Database Driver in 100 Lines of Go