Feature 1766 Limited sql sink connections

- Status: Design
- Author: Song Gao
- Discussion: https://github.com/lf-edge/ekuiper/issues/1766

Motivation

Currently, running the rule like following will create a connection to the certain database sink.

```
{
  "id": "rule%v",
  "sql": "SELECT a,b,c from demo",
  "actions": [
     {
      "log": {
      },
      "sql": {
        "url": "mysql://root@127.0.0.1:4000/test",
        "table": "test",
        "fields": ["a","b","c"]
      }
    }
  ]
}
```

If we create 5000 rules, and it will create 5000 connections to the certain database sink which may crash the database sink due to too many connections. Thus, we want to limit the connections to the database sink, and make the rules shared the same database connection pool if they have the same database driver and DSN.

Original Design

For now, each `SinkNode` would maintain one `*sql.DB, when we Open the SinkNode, one *sql.DB would be created and used by this SinkNode`

After the `Rule` was stopped or deleted, the `SinkNode` would be

```
func (m *sqlSink) Open(ctx api.StreamContext) (err error) {
        logger := ctx.GetLogger()
        logger.Debugf("Opening sql sink")

        db, err := util.Open(m.conf.Url)
        if err != nil {
                logger.Errorf("support build tags are %v", driver.KnownBuildTags())
                return err
        }
        m.db = db
        return
}
```

After the `Rule` was stopped or deleted, the `SinkNode` would be `Closed, so as the *sql.DB would be closed.`

```
func (m *sqlSink) Close(_ api.StreamContext) error {
        if m.db != nil {
                return m.db.Close()
        }
        return nil
}
```

In this way, each rule will create one `SQL` connection if it has a `SQL` sink which may cause lots of connections created due to mass rules.

New Implementation

As the Golang documents said, `*sql.DB` is a handle of the database connections, that is to say, `*sql.DB` worked as a database connection pool.

> *// DB is a database handle representing a pool of zero or more*
> *// underlying connections. It's safe for concurrent use by multiple*
> *// goroutines.*

So the idea is to make `SinkNodes` sharing the same `*sql.DB` if they have the same driver and the same DSN. We will maintain a Global Pool for the `*sql.DB` group by the `driver` and the `DSN.`

```
type driverPool struct {
	isTesting bool

	sync.RWMutex
	pool map[string]*dbPool
}
```

```
type dbPool struct {
	isTesting bool
	driver    string

	sync.RWMutex
	pool        map[string]*sql.DB
	connections map[string]int
}
```

In this way, each `SinkNode` will try to get the `*sqlDB` from the `Global Pool. Muliti SinkNode will get the same *sql.DB if they require the same driver and DSN.`

`The connections in dbPool will record the count of the SinkNode which hold the certain *sql.DB. After the SinkNode return the *sql.DB, the count would be minus 1. When the count become 0, the *sql.DB would be removed from the pool as there is no SinkNode holding it any more, as well as in order to avoid the memory leak problem.`

`After we let mulitple SinkNode share one *sql.DB, we can directly used the following method to control the total connections to a certain database instance.`

```
db.SetMaxOpenConns(c.Sink.SinkSQLConf.MaxConnections)
```

In this way, after we created the mass rules to a single database sink, the total count of the connections would be controlled as a fixed number.

Configuration

We will expose the `MaxConnections` in Configuration as following:

```
basic:
  sql:
    maxConnections: 100
```