

虚谷数据库

Go 语言标准接口 V1.0.0

开发指南

文档版本 01

发布日期 2024-03-15



版权所有 © 2024 成都虚谷伟业科技有限公司。

声明

未经本公司正式书面许可，任何企业和个人不得擅自摘抄、复制、使用本文档中的部分或全部内容，且不得以任何形式进行传播。否则，本公司将保留追究其法律责任的权利。

用户承诺在使用本文档时遵守所有适用的法律法规，并保证不以任何方式从事非法活动。不得利用本文档内容进行任何侵犯他人权益的行为。

商标声明



为成都虚谷伟业科技有限公司的注册商标。

本文档提及的其他商标或注册商标均非本公司所有。

注意事项

您购买的产品或服务应受本公司商业合同和条款的约束，本文档中描述的部分产品或服务可能不在您的购买或使用范围之内。由于产品版本升级或其他原因，本文档内容将不定期进行更新。

除非合同另有约定，本文档仅作为使用指导，所有内容均不构成任何声明或保证。

成都虚谷伟业科技有限公司

地址：四川省成都市锦江区锦盛路 138 号佳霖科创大厦 5 楼 3-14 号

邮编：610023

网址：www.xugudb.com

前言

概述



本文档主要介绍虚谷数据库 Go 语言驱动接口的主要功能和其简单的外围应用，旨在帮助使用虚谷数据库服务的应用开发人员快速开发有关于数据库交互的接口编程。

读者对象

- 数据库管理员
- 软件工程师
- 技术支持工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 注意	用于传递设备或环境安全警示信息，若不可避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。
 说明	对正文中重点信息的补充说明。“说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2024-03-15	第一次发布

目录

1	Go 概述	1
1.1	Go 介绍	1
1.2	接口标准	1
1.3	功能特性	1
2	快速入手	2
2.1	搭建开发环境	2
2.1.1	Windows x86_64 平台	2
2.1.2	Linux x86_64 平台	3
2.2	Go 版本要求	3
3	函数接口	4
3.1	type DB	4
3.1.1	Open	4
3.1.2	SetConnMaxLifetime	4
3.1.3	SetMaxIdleConns	5
3.1.4	Begin	6
3.1.5	Close	6
3.1.6	Conn	7
3.1.7	Driver	8
3.1.8	Exec	8
3.1.9	ExecContext	8
3.1.10	Ping	9
3.1.11	PingContext	10
3.1.12	Prepare	10
3.1.13	PrepareContext	11
3.1.14	Query	12
3.1.15	QueryContext	13
3.1.16	QueryRow	14

3.1.17	QueryRowContext	15
3.2	type Rows	16
3.2.1	Close	16
3.2.2	Columns	16
3.2.3	Next	17
3.2.4	NextResultSet	18
3.2.5	Scan	19
3.3	type Stmt	20
3.3.1	Close	20
3.3.2	Exec	21
3.3.3	Query	22
3.3.4	QueryRow	23
3.4	type Tx	24
3.4.1	Commit	24
3.4.2	Exec	24
3.4.3	Prepare	25
3.4.4	Query	26
3.4.5	QueryRow	27
3.4.6	Rollback	28
3.4.7	Stmt	29
4	异常处理	30
4.1	错误码	30

1 Go 概述

1.1 Go 介绍

Go (Golang) 编程语言是一门可以让程序员更加高效编程的语言。

Go 具有简洁、高效的特点。它的并发机制使得编写程序更容易，可以最大限度地利用多核的处理能力，同时支持构造灵活的模块化程序。Go 可以快速编译成机器代码，同时还具有垃圾回收机制和反射机制。这是一种快速的静态类型的编译语言，感觉像是一种动态类型的解释语言。

虚谷数据库 Go 语言访问接口是一套专用于 Go 编程语言访问和操作虚谷数据库的编程接口，旨在帮助使用 Go 编程语言进行软件开发的开发人员更方便快捷的与虚谷数据库服务进行交互。

1.2 接口标准

虚谷数据库 Go 语言访问接口的开发标准完全基于 Go 的源码包 `database/sql` 进行开发和设计，对外提供的应用层接口即是 `database/sql` 的标准接口。接口的具体使用方式可参见[Go 语言的官方文档](#)。

接口底层通讯部分采用自主研发的通讯协议进行设计，从而保证访问接口对外的使用一致性，对内的安全可靠。

1.3 功能特性

虚谷数据库 Go 语言访问接口在与虚谷数据库进行交互的过程中，支持以下的功能特性：

- 支持与虚谷数据库单机及集群的连接访问。
- 支持 IPS 特性连接虚谷数据库集群。
- 完全支持 SQL 标准语法。
- 支持大对象数据类型。
- 支持 SQL 语句的预准备。
- 支持多 SQL 语句执行及多结果集获取。

2 快速入手

2.1 搭建开发环境

2.1.1 Windows x86_64 平台

操作步骤

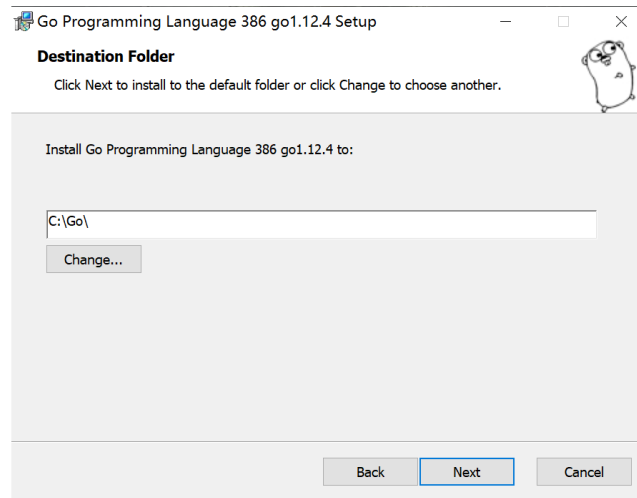
1. 访问[Golang 官方网站](#)下载已经编译完成的 Windows 版本源码包。

图 2-1 Windows 配置环境变量

go1.12.4.linux-arm64.tar.gz	Archive	Linux	ARMv8	99MB	67d7b43179c2d89a2ed0eaf3b477a2231e97612b491178aee02163916917d6f
go1.12.4.linux-armv6l.tar.gz	Archive	Linux	ARMv6	101MB	c45457b4d890116e7b79e92823001e6735501016eae228335c1e204e9b9e71492
go1.12.4.linux-ppc64le.tar.gz	Archive	Linux	ppc64le	99MB	5164212f1c08e47e4f6c4e952c2927a44347811021a949921a477e0b99095931216
go1.12.4.linux-s390x.tar.gz	Archive	Linux	s390x	106MB	0eab0c298c990a71455531e4eac977cc7398a692145a0e557b3d950942467a
go1.12.4.windows-386.zip	Archive	Windows	x86	116MB	0e0c49c71c058ae9829523842669979b414a48c479bc783d3a5212126045981165
go1.12.4.windows-386.msi	Installer	Windows	x86	102MB	71d09c22258d7499cc9b89987c32942bcb9d9a2259b45f1c2aa979e4e42d55
go1.12.4.windows-amd64.zip	Archive	Windows	x86-64	135MB	25b083e8acc2f262b7b8edf463baef2145a42522d41101888f91641e4d4dca0

2. 双击运行上一步骤下载的 go1.12.4windows-386.msi 文件进行安装，默认安装路径 C:\Go\。

图 2-2 Windows 配置环境变量



3. 准备适配版本的 minGW 编译器。
4. 联系虚谷数据库产品团队获取 Golang 语言驱动包 go-driver-xugusql。
5. 引用 go-driver-xugusql 包进行软件开发。

```
package main

import (
    _ "go-driver-xugusql" /* 引用 go-driver-xugusql 的路径 */
    "database/sql"
)
```


2.1.2 Linux x86_64 平台

操作步骤

1. 访问[Golang 官方网站](#)下载已经编译完成的 Linux 版本源码包。

图 2-3 Windows 配置环境变量

File name	Kind	OS	Arch	Size	SHA256 Checksum
go1.12.4.src.tar.gz	Source			21MB	4e1fc3a610c48182c47ab8c5b0c9e4c3c42b945455aa2ba952964ad9df11467
go1.12.4.darwin-amd64.tar.gz	Archive	macOS	x86-64	122MB	50af1aa9b4783358d58a125c5a72a1ba411635c0e8f25b58e59138096730a49
go1.12.4.darwin-amd64.pkg	Installer	macOS	x86-64	121MB	5b3c6aa4d31146809a2776c685e09190185307074e45af25614472f8a6a3c594
go1.12.4.freebsd-386.tar.gz	Archive	FreeBSD	x86	104MB	8095af056a75ecf72c4a4c6517b03996f39444c2b4017ba8df9b5c5f5640444
go1.12.4.freebsd-amd64.tar.gz	Archive	FreeBSD	x86-64	122MB	64130606b12082bf12771f54b82910d8b46120513aa8271391a220f45956e4b
go1.12.4.linux-386.tar.gz	Archive	Linux	x86	104MB	4b45c519567c13c9c45930475d1723758c88c6b74499125ab4821e9413838457
go1.12.4.linux-amd64.tar.gz	Archive	Linux	x86-64	122MB	d7d1f11884af55840713dc1747f27a790cbcaa448f6c9c151bb010aa65442f5

2. 解压文件至本地目录，并将其 go/bin 目录写入环境变量 PATH。
3. 准备 GCC 编译环境。
4. 联系虚谷数据库产品团队获取语言驱动包 go-driver-xugusql。
5. 引用 go-driver-xugusql 包进行软件开发。

```
package main

import (
    "go-driver-xugusql" /* 引用 go-driver-xugusql 的路径 */
    "database/sql"
)
```

2.2 Go 版本要求

虚谷数据库 Go 语言访问接口的整个开发过程基于 go version 1.12.4 完成，因此在进行虚谷数据库 Go 语言访问接口开发环境的搭建时建议安装 go version 1.12.4 或更高版本。

3 函数接口

3.1 type DB

3.1.1 Open

Open 打开一个由其数据库驱动名称和特定于驱动程序的数据源名称指定的数据库，该名称通常由数据库名称和连接信息组成。大多数用户将通过特定于驱动程序的连接来打开数据库，该函数返回 DB 类型的指针，该指针可安全地供多个 goroutine 并发使用，并维护自己的空闲连接。因此，Open 函数应仅被调用一次。

语法格式

```
func Open (driverName, dataSourceName string) (*DB, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
    ...
}
```

3.1.2 SetConnMaxLifetime

SetConnMaxLifetime 设置可以重用连接的最长时间，过期的连接可以在重新使用之前延迟关闭。如果给定参数值 ≤ 0 ，则连接永远重复使用。

语法格式

```
func (db *DB) SetConnMaxLifetime(d time.Duration)
```

示例

```
package main

import (
    _ "xugusql"
)
```

```
"database/sql"
"log"
"time"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Dura, err := time.ParseDuration("30m")
    if err != nil {
        log.Fatal(err)
    }

    db.SetConnMaxLifetime(Dura)

    db.Close()
}
```

3.1.3 SetMaxIdleConns

SetMaxIdleConns 设置空闲连接池中的最大连接数。如果 MaxOpenConns 大于 0 但小于新的 MaxIdleConns，则新的 MaxIdleConns 将减少以匹配 MaxOpenConns 限制。如果参数 ≤ 0 ，则不保留空闲连接。当前的默认最大空闲连接数为 2，在将来的版本中可能会更改。

语法格式

```
func (db *DB) SetMaxIdleConns(n int)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {

    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    db.SetMaxIdleConns(5)
    db.Close()
}
```

3.1.4 Begin

以指定连接的数据库默认隔离基本开启事务。

语法格式

```
func (db *DB) Begin() (*Tx, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()

    if err != nil {
        log.Fatal(err)
    }

    _, err = Tx.Exec("insert into go_1th values(100, 0.2, 0.2)")
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Rollback()
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.1.5 Close

Close() 方法将关闭数据库并阻止启用新查询，然后等待所有已经在服务器上处理的查询完成。

该方法通常很少使用，因为数据库句柄是长期存在的并且在多个 goroutine 之间共享。

语法格式

```
func (db *DB) Close() error
```

示例

```
package main
```

```
import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    db.Close()
}
```

3.1.6 Conn

Conn 方法通过打开新连接或从连接池返回现有连接来返回单个连接。Conn 将阻塞，直到返回连接或取消 ctx。在同一 Conn 上运行的查询将在同一数据库会话中运行。

语法格式

```
func (db *DB) Conn(ctx context.Context) (*Conn, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "fmt"
    "context"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    conn, err := db.Conn(context.Background())
    if err != nil {
        log.Fatal(err)
    }

    _, err = conn.ExecContext(context.TODO(),
        "insert into go_lth values(100, 0.2, 0.2)")
    if err != nil {
        log.Fatal(err)
    }

    conn.Close()
    db.Close()
}
```

3.1.7 Driver

Driver 方法将返回当前指定的基础底层驱动。

语法格式

```
func (db *DB) Driver() driver.Driver
```

3.1.8 Exec

Exec 方法执行一条非查询类型的 SQL 语句。

语法格式

```
func (db *DB) Exec (query string, args ...interface{}) (Result, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "context"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    _, err = db.ExecContext(context.Background,
        "create table go_test(c1 int, c2 varchar);" )
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.1.9 ExecContext

ExecContext 方法执行一条非查询类型的 SQL 语句。

语法格式

```
func (db *DB) ExecContext (ctx context.Context, query string, args ...interface{}) (Result, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "context"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    _, err = db.ExecContext(context.Background(),
        "create table go_test(c1 int, c2 varchar);" )
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.1.10 Ping

Ping 方法用于验证当前到数据库的连接是否仍然有效，并在必要时建立连接。

语法格式

```
func (db *DB) Ping () error
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "fmt"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    err = db.Ping()
    if err != nil {
        fmt.Printf("connect xugu dbms ... failed\n")
    } else {
        fmt.Printf("connect xugu dbms ... ok\n")
    }
}
```

```
err = db.Close()
}
```

3.1.11 PingContext

PingContext 方法用于验证当前到数据库的连接是否仍然有效，并在必要时建立连接。

语法格式

```
func (db *DB) PingContext(ctx context.Context) error
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "fmt"
    "context"
)

func main() {
    db, err := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;
        User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    err = db.PingContext(context.Background())
    if err != nil {
        fmt.Printf("connect xugu dbms ... failed\n")
    } else {
        fmt.Printf("connect xugu dbms ... ok\n")
    }

    err = db.Close()
}
```

3.1.12 Prepare

Prepare 方法为以后的 SQL 语句执行创建一个准备好的语句，可以从返回的语句中并发运行多个查询或执行。当不再需要该语句时，调用方必须调用该语句的 Close 方法。

语法格式

```
func (db *DB) Prepare(query string) (*Stmt, error)
```

示例


```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, err := db.Prepare("insert into go_test values(?, ?);")
    if err != nil {
        log.Fatal(err)
    }
    _, err = stmt.Exec(3, "xugusql")
    if err != nil {
        log.Fatal(err)
    }

    stmt.Close()
}
```

3.1.13 PrepareContext

PrepareContext 方法为以后的 SQL 语句执行创建一个准备好的语句，可以从返回的语句中并发运行多个查询或执行。当不再需要该语句时，调用方必须调用该语句的 Close 方法。

语法格式

```
func (db *DB) PrepareContext (ctx context.Context, query string) (*
    Stmt, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "context"
)

func main() {

    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, err := db.Prepare(context.Background(),
```

```

    "insert into go_test values(?, ?);")
    if err != nil {
        log.Fatal(err)
    }
    _, err = stmt.Exec(3, "xugusql")
    if err != nil {
        log.Fatal(err)
    }
    stmt.Close()
}

```

3.1.14 Query

Query 方法执行返回行的查询，通常为 select 语句。

语法格式

```

func (db *DB) Query (query string, args ...interface{}) (*Rows,
    error)

```

示例

```

package main

import (
    _ "xugusql/go-driver-xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
        SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    rows, err := db.Query("select c1, c2 from go_test")
    if err != nil {
        log.Fatal(err)
    }
    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }
    pvals := make([]interface{}, len(cols))
    for key, _ := range pvals {
        dest := make([]byte, 216)

        Pvals[key] = &dest
    } /* end for */

    for rows.Next() {
        err = rows.Scan(pvals...)
        if err != nil {
            log.Fatal(err)
        }
    }
}

```

```
    }
    for _, v := range pvals {
        fmt.Printf("%s\t", string(*v.(*[]byte)))
    }

    fmt.Printf("\n")
}

rows.Close()
}
```

3.1.15 QueryContext

QueryContext 方法执行返回行的查询，通常为 select 语句。

语法格式

```
func (db *DB) QueryContext (ctx context.Context, query string, args
...interface{}) (*Rows, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "context"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
SYSDBA;"
+ "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    rows, err := db.QueryContext(context.Background(),
        "select c1, c2 from go_test")
    if err != nil {
        log.Fatal(err)
    }

    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }
    pvals := make([]interface{}, len(cols))
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    } /* end for */

    for rows.Next() {
        err = rows.Scan(pvals...)
    }
```

```
        if err != nil {
            log.Fatal(err)
        }

        for _, v := range pvals {

            fmt.Printf( "%s\t" , string(*(v.(*[]byte))))
        }
        fmt.Printf("\n")
    }

    rows.Close()
}
```

3.1.16 QueryRow

QueryRow 执行的查询预期最多返回一行。QueryRow 始终返回非 nil 值，错误将延迟到调用 Row 的 Scan 方法之前。如果查询未选择任何行，则在调用 Scan 时返回 ErrNoRows；否则，返回结果集中的第一行，并丢弃其余的行。

语法格式

```
func (db *DB) QueryRow (query string, args ...interface{}) *Row
```

示例

```
package main

import (
    _ "xugusql-go-driver-xugusql"
    "database/sql"
    "log"
    "fmt"
)

func main() {

    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    row := db.QueryRow("select c1,c2,c3 from go_1th;")
    pvals := make([]interface{}, 3)
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    }

    err := row.Scan(pvals...)
    if err != nil {
        log.Fatal(err)
    }
    for _, v := range pvals {
```

```
        fmt.Printf("%s\t", string(*(v.(*[]byte))))
    }
    fmt.Printf("\n")

    db.Close()
}
```

3.1.17 QueryRowContext

QueryRowContext 执行的查询预期最多返回一行。QueryRow 始终返回非 nil 值，错误将延迟到调用 Row 的 Scan 方法之前。如果查询未选择任何行，则在调用 Scan 时返回 ErrNoRows；否则，返回结果集中的第一行，并丢弃其余的行。

语法格式

```
func (db *DB) QueryRowContext (ctx context.Context, query string,
    args ...interface{}) *Row
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "fmt"
    "context"
)

func main() {

    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    row := db.QueryRowContext(context.Background(),
        "select c1,c2,c3 from go_1th;")

    pvals := make([]interface{}, 3)
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    }

    err := row.Scan(pvals...)
    if err != nil {
        log.Fatal(err)
    }

    for _, v := range pvals {
        fmt.Printf("%s\t", string(*(v.(*[]byte))))
    }
}
```

```
}
    fmt.Printf("\n")

    db.Close()
}
```

3.2 type Rows

3.2.1 Close

Close 方法用于关闭 Rows，如果调用 Next 并返回 false，并且没有其他结果集，则将自动关闭 Rows。

语法格式

```
func (s *Rows) Close() error
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
    rows, err := db.Query("select c1, c2 from go_test")
    if err != nil {
        log.Fatal(err)
    }
    rows.Close()
    db.Close()
}
```

3.2.2 Columns

Columns 方法用于返回列名。如果关闭 Rows，则列返回错误。

语法格式

```
func (s *Rows) Columns() ([]string, error)
```

示例

```
package main
```

```
import (
    _ "xugusql"
    "database/sql"
    "log"
)
func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
        SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
    stmt, _ := db.Prepare("select * from go_test where id = ?;")
    rows, err := stmt.Query(3)
    if err != nil {
        log.Fatal(err)
    }
    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }
    for _, name := range cols {
        fmt.Printf("%s\t", name)
    }
    fmt.Printf("\n")
    rows.Close()
    stmt.Close()
    db.Close()
}
```

3.2.3 Next

Next 方法准备下一个结果行，以使用 Scan 方法读取。如果成功，它将返回 true; 如果没有下一个结果行或在准备它时发生错误，则返回 false。

语法格式

```
func (s *Rows) Next() bool
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
        SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    rows, err := db.Query("select c1, c2 from go_test")
    if err != nil {
```

```
        log.Fatal(err)
    }

    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }

    pvals := make([]interface{}, len(cols))
    for key, _ := range pvals {
        dest := make([]byte, 216)
        Pvals[key] = &dest
    } /* end for */

    for rows.Next() {
        err = rows.Scan(pvals...)
        if err != nil {
            log.Fatal(err)
        }

        for _, v := range pvals {
            fmt.Printf("%s\t", string(*(v.([]byte))))
        }
        fmt.Printf("\n")
    }

    rows.Close()
}
```

3.2.4 NextResultSet

NextResultSet 方法准备下一个要读取的结果集。它通常用于报告是否还存在下一个结果集，如果没有其他结果集或前进时出现错误，则报告 false。应该使用 Err 方法来区分这两种情况。

语法格式

```
func (s *Rows) NextResultSet() bool
```

示例

```
package main

import (
    _ "xugusql-go-driver-xugusql"
    "database/sql"
    "log"
    "fmt"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;"+
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;"+
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
}
```



```
rows, err := db.Query( "select * from go_1th;select * from
    go_2th;" )
if err != nil {
    log.Fatal(err)
}

var cols []string
cols, err = rows.Columns()
if err != nil {
    log.Fatal(err)
}
pvals := make([]interface{}, len(cols))
for key, _ := range pvals {
    dest := make([]byte, 216)
    pvals[key] = &dest
}

set := 1
for true {
    fmt.Printf("%dth Result Set:\n", set)
    for rows.Next() {
        err = rows.Scan(pvals...)
        if err != nil {
            log.Fatal(err)
            log.Fatal(err)
        } /* end if */

        for _, v := range pvals {
            fmt.Printf("%s\t", string(*(v.(*[]byte))))
        }
        fmt.Printf("\n")
    } /* end for */

    if !rows.NextResultSet() {
        break
    }

    set++
} /* end for */

rows.Close()
db.Close()
}
```

3.2.5 Scan

Scan 方法将当前 Rows 中的列复制到 dest 指向的值中。dest 中的值数必须与 Rows 中的列数相同。

语法格式

```
func (s *Rows) Scan(dest ...interface{}) error
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    rows, err := db.Query("select c1, c2 from go_test")
    if err != nil {
        log.Fatal(err)
    }

    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }

    pvals := make([]interface{}, len(cols))
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    } /* end for */

    for rows.Next() {
        err = rows.Scan(pvals...)
        if err != nil {
            log.Fatal(err)
        }

        for _, v := range pvals {
            fmt.Printf("%s\t", string(*v.(*[]byte)))
        }
        fmt.Printf("\n")
    }

    rows.Close()
}
```

3.3 type Stmt

3.3.1 Close

Close 方法会关闭当前的语句句柄。

语法格式

```
func (s *Stmt) Close() error
```

示例

```
package main

import (
    _ "xugusql-go-driver-xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
    stmt, err := db.Prepare("insert into go_test values(?, ?);")
    if err != nil {
        log.Fatal(err)
    }
    stmt.Close()
}
```

3.3.2 Exec

Exec 方法用给定的参数执行一条准备好的 SQL 语句，并返回一个 Result 类型的结果对象。

语法格式

```
func (s *Stmt) Exec(args ...interface{}) (Result, error)
```

示例

```
package main

import (
    _ "xugusql-go-driver-xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, err := db.Prepare("insert into go_test values(?, ?);")
    if err != nil {
        log.Fatal(err)
    }

    res, _ := stmt.Exec(4, "Exec")
    id, _ := res.LastInsertId()
}
```

```
stmt.Close()
db.Close()

}
```

3.3.3 Query

Query 方法用给定的参数执行一条准备好的查询语句，并将查询结果作为 *Rows 返回。

语法格式

```
func (s *Stmt) Query(args ...interface{}) (*Rows, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, _ := db.Prepare("select * from go_test where id = ?;")
    rows, err := stmt.Query(3)
    if err != nil {
        log.Fatal(err)
    }

    var cols []string
    cols, err = rows.Columns()
    if err != nil {
        log.Fatal(err)
    }
    pvals := make([]interface{}, len(cols))
    for key, _ := range pvals {

        dest := make([]byte, 216)
        pvals[key] = &dest
    } /* end for */

    for rows.Next() {
        err = rows.Scan(pvals...)
        if err != nil {
            log.Fatal(err)
        }

        for _, v := range pvals {
            fmt.Printf("%s\t", string(*v.(*[]byte)))
        }
    }
}
```

```
        fmt.Printf("\n")
    }

    rows.Close()
}
```

3.3.4 QueryRow

QueryRow 执行的查询预期最多返回一行。QueryRow 始终返回非 nil 值，错误将延迟到调用 Row 的 Scan 方法之前。如果查询未选择任何行，则在调用 Scan 时返回 ErrNoRows；否则，返回结果集中的第一行，并丢弃其余的行。

语法格式

```
func (s *Stmt) QueryRow (args ...interface{}) *Row
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "fmt"
)

func main() {

    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, _ := db.Prepare("select c1,c2,c3 from go_1th;")
    row := stmt.QueryRow()

    pvals := make([]interface{}, 3)
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    }

    err := row.Scan(pvals...)
    if err != nil {
        log.Fatal(err)
    }

    for _, v := range pvals {
        fmt.Printf("%s\t", string(*v.(*[]byte)))
    }
    fmt.Printf("\n")

    db.Close()
}
```

```
}
```

3.4 type Tx

3.4.1 Commit

提交事务。

语法格式

```
func (tx *Tx) Commit() error
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }

    _, err = Tx.Exec("insert into go_1th values(100, 0.2, 0.2)")
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Commit()
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.4.2 Exec

Exec 方法执行不返回行的查询。例如：INSERT 和 UPDATE。

语法格式

```
func (tx *Tx) Exec(query string, args...interface{}) (Result, error)
)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }

    _, err = Tx.Exec("insert into go_1th values(100, 0.2, 0.2)")
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Rollback()
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.4.3 Prepare

Prepare 方法创建一个准备就绪的语句在事务中使用。返回的语句在事务内运行，并且一旦事务已提交或回滚就不能使用。

语法格式

```
func (tx *Tx) Prepare(query string) (*Stmt, error)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)
```

```
func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
        SYSDBA;"
    + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }

    stmt, _ = Tx.Prepare("insert into go_test values(?, ?);")
    _, err = stmt.Exec(5, "Transaction")
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Rollback()
    if err != nil {
        log.Fatal(err)
    }

    stmt.Close()
    db.Close()
}
```

3.4.4 Query

Query 执行放回行的查询，通常为 SELECT。

语法格式

```
func (tx *Tx) Query(query string, args...interface{}) (*Rows, error
)
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=
        SYSDBA;"
    + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }

    _, err = Tx.Query("select c1, c2 from go_test")
    if err != nil {
```



```
        log.Fatal(err)
    }

    err = Tx.Rollback()
    if err != nil {
        log.Fatal(err)
    }
    stmt.Close()
    db.Close()
}
```

3.4.5 QueryRow

QueryRow 执行的查询预期最多返回一行。QueryRow 始终返回非 nil 值，错误将延迟到调用 Row 的 Scan 方法之前。如果查询未选择任何行，则在调用 Scan 时返回 ErrNoRows；否则，返回结果集中的第一行，并丢弃其余的行。

语法格式

```
func (tx *Tx) QueryRow (query string, args ...interface{}) *Row
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
    "fmt"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;" +
        "DB=SYSTEM;User=SYSDBA;PWD=SYSDBA;" +
        "Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }
    row := Tx.QueryRow("select c1,c2,c3 from go_1th;")
    pvals := make([]interface{}, 3)
    for key, _ := range pvals {
        dest := make([]byte, 216)
        pvals[key] = &dest
    }
    err = row.Scan(pvals...)
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Commit()
}
```

```
    if err != nil {
        log.Fatal(err)
    }

    for _, v := range pvals {
        fmt.Printf("%s\t", string(*(v.(*[]byte))))
    }
    fmt.Printf("\n")

    db.Close()
}
```

3.4.6 Rollback

Rollback 方法将终止事务。

语法格式

```
func (tx *Tx) QueryRow (query string, args ...interface{}) *Row
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")
    Tx, err := db.Begin()
    if err != nil {
        log.Fatal(err)
    }

    _, err = Tx.Exec("insert into go_1th values(100, 0.2, 0.2)")
    if err != nil {
        log.Fatal(err)
    }

    err = Tx.Rollback()
    if err != nil {
        log.Fatal(err)
    }

    db.Close()
}
```

3.4.7 Stmt

Stmt 方法从现有语句返回特定于事务的预准备语句。

语法格式

```
func (tx *Tx) Stmt(stmt *Stmt) *Stmt
```

示例

```
package main

import (
    _ "xugusql"
    "database/sql"
    "log"
)

func main() {
    db, _ := sql.Open("xugusql", "IP=192.168.78.130;DB=SYSTEM;User=SYSDBA;"
        + "PWD=SYSDBA;Port=5138;AUTO_COMMIT=on;CHAR_SET=UTF8")

    stmt, err := db.Prepare("insert into go_test values(?, ?);")
    if err != nil {
        log.Fatal(err)
    }

    Tx, _ := db.Begin()
    _, err = Tx.Stmt(stmt).Exec(3, "xugusql")
    if err != nil {
        log.Fatal(err)
    }

    Tx.Commit()
    stmt.Close()
}
```

4 异常处理

4.1 错误码

错误码	错误表示	描述
E5021	ERR_DUP_VAL_ON_INDEX	违反唯一值约束
E5022	ERR_NO_DATA_FOUND	查询无结果
E7002	ERR_DATA_LEN	数值超界
E10001	ERR_FUNC_NO_EXIST	函数不存在
E13001	ERR_PARSE	语法错误
E17005	ERR_TAB_NO_EXIST	表或视图不存在
E19009	ERR_TAB_EXIST	表已存在
E19132	ERR_SEQ_NO_EXIST	序列值发生器不存在



成都虚谷伟业科技有限公司

联系电话：400-8886236

官方网站：www.xugudb.com