

[Discover Packages](#) > [Standard library](#) > [database](#) > [sql](#) > [driver](#) 


driver

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [6](#) |Imported by: [17,682](#)

Details

 Valid [go.mod](#) file  Redistributable license  Tagged version  Stable version [Learn more](#)

Repository

cs.opensource.google/go/go

Links

 [Report a Vulnerability](#) Documentation 

<> Documentation

Overview

Package driver defines interfaces to be implemented by database drivers as used by package sql.

Most code should use package sql.

The driver interface has evolved over time. Drivers should implement Connector and DriverContext interfaces. The Connector.Connect and Driver.Open methods should never return ErrBadConn. ErrBadConn should only be returned from Validator, SessionResetter, or a query method if the connection is already in an invalid (e.g. closed) state.

All Conn implementations should implement the following interfaces: Pinger, SessionResetter, and Validator.

If named parameters or context are supported, the driver's Conn should implement: ExecerContext, QueryerContext, ConnPrepareContext, and ConnBeginTx.

To support custom data types, implement NamedValueChecker. NamedValueChecker also allows queries to accept per-query options as a parameter by returning ErrRemoveArgument from CheckNamedValue.

If multiple result sets are supported, Rows should implement RowsNextResultSet. If the driver knows how to describe the types present in the returned result it should implement the following interfaces: RowsColumnTypeScanType, RowsColumnTypeDatabaseTypeName, RowsColumnTypeLength, RowsColumnTypeNullable, and RowsColumnTypePrecisionScale. A given row value may also return a Rows type, which may represent a database cursor value.

Before a connection is returned to the connection pool after use, `IsValid` is called if implemented. Before a connection is reused for another query, `ResetSession` is called if implemented. If a connection is never returned to the connection pool but immediately reused, then `ResetSession` is called prior to reuse but `IsValid` is not called.

Index

Variables

`func IsScanValue(v any) bool`

`func IsValue(v any) bool`

`type ColumnConverter` DEPRECATED

`type Conn`

`type ConnBeginTx`

`type ConnPrepareContext`

`type Connector`

`type Driver`

`type DriverContext`

`type Execer` DEPRECATED

`type ExecerContext`

`type IsolationLevel`

`type NamedValue`

`type NamedValueChecker`

`type NotNull`

`func (n NotNull) ConvertValue(v any) (Value, error)`

`type Null`

`func (n Null) ConvertValue(v any) (Value, error)`

`type Pinger`

`type Queryer` DEPRECATED

`type QueryerContext`

`type Result`

`type Rows`

`type RowsAffected`

`func (RowsAffected) LastInsertId() (int64, error)`

`func (v RowsAffected) RowsAffected() (int64, error)`

`type RowsColumnTypeDatabaseTypeName`

`type RowsColumnTypeLength`

`type RowsColumnTypeNullable`

`type RowsColumnTypePrecisionScale`

`type RowsColumnTypeScanType`

`type RowsNextResultSet`

`type SessionResetter`

`type Stmt`

`type StmtExecContext`

`type StmtQueryContext`

`type Tx`

`type TxOptions`

`type Validator`

[type Value](#)
[type ValueConverter](#)
[type Valuer](#)

Constants

This section is empty.

Variables

[View Source](#)

```
var Bool boolType
```

Bool is a ValueConverter that converts input values to bools.

The conversion rules are:

- booleans are returned unchanged
- for integer types, 1 is true 0 is false, other integers are an error
- for strings and []byte, same rules as strconv.ParseBool
- all other types are an error

[View Source](#)

```
var DefaultParameterConverter defaultConverter
```

DefaultParameterConverter is the default implementation of ValueConverter that's used when a Stmt doesn't implement ColumnConverter.

DefaultParameterConverter returns its argument directly if IsValue(arg). Otherwise, if the argument implements Valuer, its Value method is used to return a Value. As a fallback, the provided argument's underlying type is used to convert it to a Value: underlying integer types are converted to int64, floats to float64, bool, string, and []byte to themselves. If the argument is a nil pointer, ConvertValue returns a nil Value. If the argument is a non-nil pointer, it is dereferenced and ConvertValue is called recursively. Other types are an error.

[View Source](#)

```
var ErrBadConn = errors.New("driver: bad connection")
```

ErrBadConn should be returned by a driver to signal to the sql package that a driver.Conn is in a bad state (such as the server having earlier closed the connection) and the sql package should retry on a new connection.

To prevent duplicate operations, ErrBadConn should NOT be returned if there's a possibility that the database server might have performed the operation. Even if the server sends back an error, you shouldn't return ErrBadConn.

Errors will be checked using errors.Is. An error may wrap ErrBadConn or implement the Is(error) bool method.

```
var ErrRemoveArgument = errors.New("driver: remove argument from query")
```

ErrRemoveArgument may be returned from NamedValueChecker to instruct the sql package to not pass the argument to the driver query interface. Return when accepting query specific options or structures that aren't SQL query arguments.

```
var ErrSkip = errors.New("driver: skip fast-path; continue as if unimplemented")
```

ErrSkip may be returned by some optional interfaces' methods to indicate at runtime that the fast path is unavailable and the sql package should continue as if the optional interface was not implemented. ErrSkip is only supported where explicitly documented.

```
var Int32 int32Type
```

Int32 is a ValueConverter that converts input values to int64, respecting the limits of an int32 value.

```
var ResultNoRows noRows
```

ResultNoRows is a pre-defined Result for drivers to return when a DDL command (such as a CREATE TABLE) succeeds. It returns an error for both LastInsertId and RowsAffected.

```
var String stringType
```

String is a ValueConverter that converts its input to a string. If the value is already a string or []byte, it's unchanged. If the value is of another type, conversion to string is done with `fmt.Sprintf("%v", v)`.

Functions

func IsScanValue

```
func IsScanValue(v any) bool
```

IsScanValue is equivalent to IsValue. It exists for compatibility.

func IsValue

```
func IsValue(v any) bool
```

IsValue reports whether v is a valid Value parameter type.

Types

type **Conn**

```
type Conn interface {
    // Prepare returns a prepared statement, bound to this connection.
    Prepare(query string) (Stmt, error)

    // Close invalidates and potentially stops any current
    // prepared statements and transactions, marking this
    // connection as no longer in use.
    //
    // Because the sql package maintains a free pool of
    // connections and only calls Close when there's a surplus of
    // idle connections, it shouldn't be necessary for drivers to
    // do their own connection caching.
    //
    // Drivers must ensure all network calls made by Close
    // do not block indefinitely (e.g. apply a timeout).
    Close() error

    // Begin starts and returns a new transaction.
    //
    // Deprecated: Drivers should implement ConnBeginTx instead (or additionally).
    Begin() (Tx, error)
}
```

Conn is a connection to a database. It is not used concurrently by multiple goroutines.

Conn is assumed to be stateful.

type **ConnBeginTx**

added in go1.8

```
type ConnBeginTx interface {
    // BeginTx starts and returns a new transaction.
    // If the context is canceled by the user the sql package will
    // call Tx.Rollback before discarding and closing the connection.
    //
    // This must check opts.Isolation to determine if there is a set
    // isolation level. If the driver does not support a non-default
    // level and one is set or if there is a non-default isolation level
    // that is not supported, an error must be returned.
    //
    // This must also check opts.ReadOnly to determine if the read-only
    // value is true to either set the read-only transaction property if supported
    // or return an error if it is not supported.
    BeginTx(ctx context.Context, opts TxOptions) (Tx, error)
}
```

ConnBeginTx enhances the Conn interface with context and TxOptions.

type ConnPrepareContext

added in go1.8

```
type ConnPrepareContext interface {
    // PrepareContext returns a prepared statement, bound to this connection.
    // context is for the preparation of the statement,
    // it must not store the context within the statement itself.
    PrepareContext(ctx context.Context, query string) (Stmt, error)
}
```

ConnPrepareContext enhances the Conn interface with context.

type Connector

added in go1.10

```
type Connector interface {
    // Connect returns a connection to the database.
    // Connect may return a cached connection (one previously
    // closed), but doing so is unnecessary; the sql package
    // maintains a pool of idle connections for efficient re-use.
    //
    // The provided context.Context is for dialing purposes only
    // (see net.DialContext) and should not be stored or used for
    // other purposes. A default timeout should still be used
    // when dialing as a connection pool may call Connect
    // asynchronously to any query.
    //
    // The returned connection is only used by one goroutine at a
    // time.
    Connect(context.Context) (Conn, error)

    // Driver returns the underlying Driver of the Connector,
    // mainly to maintain compatibility with the Driver method
    // on sql.DB.
    Driver() Driver
}
```

A Connector represents a driver in a fixed configuration and can create any number of equivalent Conns for use by multiple goroutines.

A Connector can be passed to `sql.OpenDB`, to allow drivers to implement their own `sql.DB` constructors, or returned by `DriverContext`'s `OpenConnector` method, to allow drivers access to context and to avoid repeated parsing of driver configuration.

If a Connector implements `io.Closer`, the `sql` package's `DB.Close` method will call `Close` and return error (if any).

type Driver

```
type Driver interface {
    // Open returns a new connection to the database.
    // The name is a string in a driver-specific format.
```

```
//
// Open may return a cached connection (one previously
// closed), but doing so is unnecessary; the sql package
// maintains a pool of idle connections for efficient re-use.
//
// The returned connection is only used by one goroutine at a
// time.
Open(name string) (Conn, error)
}
```

Driver is the interface that must be implemented by a database driver.

Database drivers may implement DriverContext for access to contexts and to parse the name only once for a pool of connections, instead of once per connection.

type DriverContext

added in go1.10

```
type DriverContext interface {
    // OpenConnector must parse the name in the same format that Driver.Open
    // parses the name parameter.
    OpenConnector(name string) (Connector, error)
}
```

If a Driver implements DriverContext, then sql.DB will call OpenConnector to obtain a Connector and then invoke that Connector's Connect method to obtain each needed connection, instead of invoking the Driver's Open method for each connection. The two-step sequence allows drivers to parse the name just once and also provides access to per-Conn contexts.

type Execer DEPRECATED [Show](#)

type ExecerContext

added in go1.8

```
type ExecerContext interface {
    ExecContext(ctx context.Context, query string, args []NamedValue) (Result, error)
}
```

ExecerContext is an optional interface that may be implemented by a Conn.

If a Conn does not implement ExecerContext, the sql package's DB.Exec will fall back to Execer; if the Conn does not implement Execer either, DB.Exec will first prepare a query, execute the statement, and then close the statement.

ExecContext may return ErrSkip.

ExecContext must honor the context timeout and return when the context is canceled.

type IsolationLevel

added in go1.8

```
type IsolationLevel int
```

IsolationLevel is the transaction isolation level stored in TxOptions.

This type should be considered identical to sql.IsolationLevel along with any values defined on it.

type NamedValue

added in go1.8

```
type NamedValue struct {
    // If the Name is not empty it should be used for the parameter identifier and
    // not the ordinal position.
    //
    // Name will not have a symbol prefix.
    Name string

    // Ordinal position of the parameter starting from one and is always set.
    Ordinal int

    // Value is the parameter value.
    Value Value
}
```

NamedValue holds both the value name and value.

type NamedValueChecker

added in go1.9

```
type NamedValueChecker interface {
    // CheckNamedValue is called before passing arguments to the driver
    // and is called in place of any ColumnConverter. CheckNamedValue must do type
    // validation and conversion as appropriate for the driver.
    CheckNamedValue(*NamedValue) error
}
```

NamedValueChecker may be optionally implemented by Conn or Stmt. It provides the driver more control to handle Go and database types beyond the default Values types allowed.

The sql package checks for value checkers in the following order, stopping at the first found match: Stmt.NamedValueChecker, Conn.NamedValueChecker, Stmt.ColumnConverter, DefaultParameterConverter.

If CheckNamedValue returns ErrRemoveArgument, the NamedValue will not be included in the final query arguments. This may be used to pass special options to the query itself.

If ErrSkip is returned the column converter error checking path is used for the argument. Drivers may wish to return ErrSkip after they have exhausted their own special cases.

type NotNull

```
type NotNull struct {
    Converter ValueConverter
}
```


NotNull is a type that implements ValueConverter by disallowing nil values but otherwise delegating to another ValueConverter.

func (NotNull) ConvertValue

```
func (n NotNull) ConvertValue(v any) (Value, error)
```

type Null

```
type Null struct {  
    Converter ValueConverter  
}
```

Null is a type that implements ValueConverter by allowing nil values but otherwise delegating to another ValueConverter.

func (Null) ConvertValue

```
func (n Null) ConvertValue(v any) (Value, error)
```

type Pinger

added in go1.8

```
type Pinger interface {  
    Ping(ctx context.Context) error  
}
```

Pinger is an optional interface that may be implemented by a Conn.

If a Conn does not implement Pinger, the sql package's DB.Ping and DB.PingContext will check if there is at least one Conn available.

If Conn.Ping returns ErrBadConn, DB.Ping and DB.PingContext will remove the Conn from pool.

type Queryer

DEPRECATED

[Show](#)

added in go1.1

type QueryerContext

added in go1.8

```
type QueryerContext interface {  
    QueryContext(ctx context.Context, query string, args []NamedValue) (Rows, error)  
}
```

QueryerContext is an optional interface that may be implemented by a Conn.

If a Conn does not implement QueryerContext, the sql package's DB.Query will fall back to Queryer; if the Conn does not implement Queryer either, DB.Query will first prepare a query, execute the statement, and then close the statement.

QueryContext may return ErrSkip.

QueryContext must honor the context timeout and return when the context is canceled.

type Result

```
type Result interface {  
    // LastInsertId returns the database's auto-generated ID  
    // after, for example, an INSERT into a table with primary  
    // key.  
    LastInsertId() (int64, error)  
  
    // RowsAffected returns the number of rows affected by the  
    // query.  
    RowsAffected() (int64, error)  
}
```

Result is the result of a query execution.

type Rows

```
type Rows interface {  
    // Columns returns the names of the columns. The number of  
    // columns of the result is inferred from the length of the  
    // slice. If a particular column name isn't known, an empty  
    // string should be returned for that entry.  
    Columns() []string  
  
    // Close closes the rows iterator.  
    Close() error  
  
    // Next is called to populate the next row of data into  
    // the provided slice. The provided slice will be the same  
    // size as the Columns() are wide.  
    //  
    // Next should return io.EOF when there are no more rows.  
    //  
    // The dest should not be written to outside of Next. Care  
    // should be taken when closing Rows not to modify  
    // a buffer held in dest.  
    Next(dest []Value) error  
}
```

Rows is an iterator over an executed query's results.

type RowsAffected

```
type RowsAffected int64
```

RowsAffected implements Result for an INSERT or UPDATE operation which mutates a number of rows.

func (RowsAffected) LastInsertId

```
func (RowsAffected) LastInsertId() (int64, error)
```

func (RowsAffected) RowsAffected

```
func (v RowsAffected) RowsAffected() (int64, error)
```

type RowsColumnTypeDatabaseTypeName

added in go1.8

```
type RowsColumnTypeDatabaseTypeName interface {  
    Rows  
    ColumnTypeDatabaseTypeName(index int) string  
}
```

RowsColumnTypeDatabaseTypeName may be implemented by Rows. It should return the database system type name without the length. Type names should be uppercase. Examples of returned types: "VARCHAR", "NVARCHAR", "VARCHAR2", "CHAR", "TEXT", "DECIMAL", "SMALLINT", "INT", "BIGINT", "BOOL", "[]BIGINT", "JSONB", "XML", "TIMESTAMP".

type RowsColumnTypeLength

added in go1.8

```
type RowsColumnTypeLength interface {  
    Rows  
    ColumnTypeLength(index int) (length int64, ok bool)  
}
```

RowsColumnTypeLength may be implemented by Rows. It should return the length of the column type if the column is a variable length type. If the column is not a variable length type ok should return false. If length is not limited other than system limits, it should return math.MaxInt64. The following are examples of returned values for various types:

```
TEXT          (math.MaxInt64, true)  
varchar(10)   (10, true)  
nvarchar(10)  (10, true)  
decimal       (0, false)  
int           (0, false)  
bytea(30)     (30, true)
```

type RowsColumnTypeNullable

added in go1.8

```
type RowsColumnTypeNullable interface {  
    Rows  
    ColumnTypeNullable(index int) (nullable, ok bool)  
}
```

RowsColumnTypeNullable may be implemented by Rows. The nullable value should be true if it is known the column may be null, or false if the column is known to be not nullable. If the column nullability is unknown, ok should be false.

type RowsColumnTypePrecisionScale

added in go1.8

```
type RowsColumnTypePrecisionScale interface {  
    Rows  
    ColumnTypePrecisionScale(index int) (precision, scale int64, ok bool)  
}
```

RowsColumnTypePrecisionScale may be implemented by Rows. It should return the precision and scale for decimal types. If not applicable, ok should be false. The following are examples of returned values for various types:

```
decimal(38, 4)    (38, 4, true)  
int               (0, 0, false)  
decimal          (math.MaxInt64, math.MaxInt64, true)
```

type RowsColumnTypeScanType

added in go1.8

```
type RowsColumnTypeScanType interface {  
    Rows  
    ColumnTypeScanType(index int) reflect.Type  
}
```

RowsColumnTypeScanType may be implemented by Rows. It should return the value type that can be used to scan types into. For example, the database column type "bigint" this should return "reflect.TypeOf(int64(0))".

type RowsNextResultSet

added in go1.8

```
type RowsNextResultSet interface {  
    Rows  
  
    // HasNextResultSet is called at the end of the current result set and  
    // reports whether there is another result set after the current one.  
    HasNextResultSet() bool  
  
    // NextResultSet advances the driver to the next result set even  
    // if there are remaining rows in the current result set.  
    //  
    // NextResultSet should return io.EOF when there are no more result sets.  
    NextResultSet() error  
}
```

RowsNextResultSet extends the Rows interface by providing a way to signal the driver to advance to the next result set.

type SessionResetter

added in go1.10

```

type SessionResetter interface {
    // ResetSession is called prior to executing a query on the connection
    // if the connection has been used before. If the driver returns ErrBadConn
    // the connection is discarded.
    ResetSession(ctx context.Context) error
}

```

SessionResetter may be implemented by Conn to allow drivers to reset the session state associated with the connection and to signal a bad connection.

type Stmt

```

type Stmt interface {
    // Close closes the statement.
    //
    // As of Go 1.1, a Stmt will not be closed if it's in use
    // by any queries.
    //
    // Drivers must ensure all network calls made by Close
    // do not block indefinitely (e.g. apply a timeout).
    Close() error

    // NumInput returns the number of placeholder parameters.
    //
    // If NumInput returns >= 0, the sql package will sanity check
    // argument counts from callers and return errors to the caller
    // before the statement's Exec or Query methods are called.
    //
    // NumInput may also return -1, if the driver doesn't know
    // its number of placeholders. In that case, the sql package
    // will not sanity check Exec or Query argument counts.
    NumInput() int

    // Exec executes a query that doesn't return rows, such
    // as an INSERT or UPDATE.
    //
    // Deprecated: Drivers should implement StmtExecContext instead (or additionally).
    Exec(args []Value) (Result, error)

    // Query executes a query that may return rows, such as a
    // SELECT.
    //
    // Deprecated: Drivers should implement StmtQueryContext instead (or additionally).
    Query(args []Value) (Rows, error)
}

```

Stmt is a prepared statement. It is bound to a Conn and not used by multiple goroutines concurrently.

type StmtExecContext

added in go1.8

```

type StmtExecContext interface {
    // ExecContext executes a query that doesn't return rows, such
    // as an INSERT or UPDATE.
    //
    // ExecContext must honor the context timeout and return when it is canceled.
    ExecContext(ctx context.Context, args []NamedValue) (Result, error)
}

```

StmtExecContext enhances the Stmt interface by providing Exec with context.

type StmtQueryContext

added in go1.8

```

type StmtQueryContext interface {
    // QueryContext executes a query that may return rows, such as a
    // SELECT.
    //
    // QueryContext must honor the context timeout and return when it is canceled.
    QueryContext(ctx context.Context, args []NamedValue) (Rows, error)
}

```

StmtQueryContext enhances the Stmt interface by providing Query with context.

type Tx

```

type Tx interface {
    Commit() error
    Rollback() error
}

```

Tx is a transaction.

type TxOptions

added in go1.8

```

type TxOptions struct {
    Isolation IsolationLevel
    ReadOnly  bool
}

```

TxOptions holds the transaction options.

This type should be considered identical to sql.TxOptions.

type Validator

added in go1.15

```

type Validator interface {
    // IsValid is called prior to placing the connection into the
    // connection pool. The connection will be discarded if false is returned.
    IsValid() bool
}

```

Validator may be implemented by Conn to allow drivers to signal if a connection is valid or if it should be discarded.

If implemented, drivers may return the underlying error from queries, even if the connection should be discarded by the connection pool.

type Value

```
type Value any
```

Value is a value that drivers must be able to handle. It is either nil, a type handled by a database driver's NamedValueChecker interface, or an instance of one of these types:

```
int64
float64
bool
[]byte
string
time.Time
```

If the driver supports cursors, a returned Value may also implement the Rows interface in this package. This is used, for example, when a user selects a cursor such as "select cursor(select * from my_table) from dual". If the Rows from the select is closed, the cursor Rows will also be closed.

type ValueConverter

```
type ValueConverter interface {
    // ConvertValue converts a value to a driver Value.
    ConvertValue(v any) (Value, error)
}
```

ValueConverter is the interface providing the ConvertValue method.

Various implementations of ValueConverter are provided by the driver package to provide consistent implementations of conversions between drivers. The ValueConverters have several uses:

- converting from the Value types as provided by the sql package into a database table's specific column type and making sure it fits, such as making sure a particular int64 fits in a table's uint16 column.
- converting a value as given from the database into one of the driver Value types.
- by the sql package, for converting from a driver's Value type to a user's type in a scan.

type Valuer

```
type Valuer interface {
    // Value returns a driver Value.
    // Value must not panic.
```

```
Value() (Value, error)
}
```

Valuer is the interface providing the Value method.

Types implementing Valuer interface are able to convert themselves to a driver Value.

Source Files

[View all](#) [driver.go](#)[types.go](#)

Why Go

[Use Cases](#)[Case Studies](#)

Get Started

[Playground](#)[Tour](#)[Stack Overflow](#)[Help](#)

Packages

[Standard Library](#)[About Go Packages](#)

About

[Download](#)[Blog](#)[Issue Tracker](#)[Release Notes](#)[Brand Guidelines](#)[Code of Conduct](#)

Connect

[Twitter](#)[GitHub](#)[Slack](#)[r/golang](#)[Meetup](#)[Golang Weekly](#)[Copyright](#)[Terms of Service](#)[Privacy Policy](#)[Report an Issue](#)The Google logo, consisting of four interlocking rings in blue, red, yellow, and green, is located in the bottom right corner of the page.