ORACLE®

The Go Language: Principles and Practices for Oracle Database

DEV5047

Anthony Tuininga
Data Access Development
Oracle Database
October 22, 2018

ORACLE CODE ONE

Live for the Code

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

About Me

Anthony Tuininga

- Creator and maintainer of cx_Oracle

Contact information

- anthony.tuininga@oracle.com
- @AnthonyTuininga

Oracle Database for the Developer

LANGUAGE	API
C	OCI, ODBC, ODPI-C
C++	OCCI
Java	JDBC
.NET	ODP.NET
Node.js	node-oracledb
Python	cx_Oracle
PHP	OCI8, PDO_OCI
R	ROracle
Erlang	erloci
Perl	DBD::Oracle
Ruby	ruby-oci8, ruby-odpi
Rust	mimir, rust-oracle
Go	goracle, rana, mattn

- Oracle Proprietary Drivers
- Oracle Open Source Drivers
- Third Party Open Source Drivers

... and Pro*C, OLE DB, Pro*COBOL, Pro*Fortran, SQLJ, OLE DB

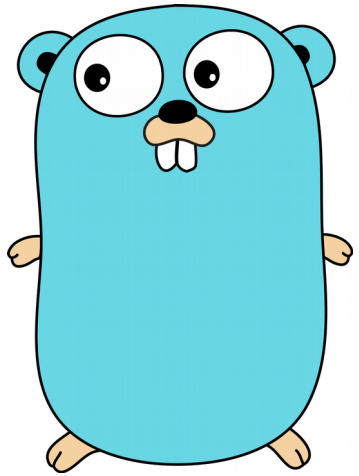
Program Agenda

1. Introduction
2. Types in Go
3. General Database Support
4. Oracle Database Support

Introduction

What is Go?

- General purpose, cross platform, open source programming language
- In the tradition of C but with many changes to make code shorter, simpler and safer
- Started at Google in 2007
  - Goal: “Fast, statically typed, compiled language that feels like a dynamically typed, interpreted language”
  - Open source in 2009
  - Version 1.0 released 2012, now at version 1.11
  - Consideration of version 2 features underway



Popularity

- Increased in popularity rapidly in last three years
  - Stack Overflow put Go in #16 earlier this year
  - TIOBE index for October 2018 puts Go in #12
  - GitHub puts Go as 7th fastest growing language (14,000+ repositories)
- Docker container system and Kubernetes container management system written in Go

What Does Go Have

- Fast compilation times
- Built-in concurrency primitives
- Built-in support for Unicode
- Automatic memory management
- Binaries are standalone and have no dependencies
- Simple cross compilation
- Easily extensible via packages
- Type inference
- Can integrate with C code

What Does Go Not Have

- No forward declarations or header files
- No exception handling (explicit error checking)
- No type hierarchy/inheritance (interfaces inferred by compiler)
- No overloading of methods/operators
- No implicit conversions
- No declaration of public/private (uses case of first letter of identifiers)

Simple Example

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, world!")
}
```

```
go run hello.go

go build hello.go
./hello

GOOS=windows go build hello.go
```

Types in Go

Basic Types

- Integer types (int, int8, int16, int32, int64, uint, etc.)
- Floating point types (float32, float64)
- Complex types (complex64, complex128)
- String
- Bool

Derived Types

- Pointers, functions
- Arrays, slices, maps
- Structures, interfaces
- Channels

Arrays/Slices

```go
var arr [5]int
fmt.Println("1 array:", arr)
for i := range arr {
    arr[i] = i * i
}
fmt.Println("2 array:", arr)

slice := arr[1:3]
fmt.Println("3 slice:", slice)
slice[1] = 15
fmt.Println("4 array:", arr)
fmt.Println("5 slice:", slice)
```

```
1 array: [0 0 0 0 0]
2 array: [0 1 4 9 16]
3 slice: [1 4]
4 array: [0 1 15 9 16]
5 slice: [1 15]
```

Structures/Interfaces

```
type TwoDShape interface {
    Area() float64
    Description() string
}

type Circle struct {
    Radius float64
}

func (circle *Circle) Area() float64 {
    return math.Pi * circle.Radius * circle.Radius
}

func (circle *Circle) Description() string {
    return fmt.Sprintf("Circle of radius %v", circle.Radius)
}
```

Structures/Interfaces (continued)

```
type Triangle struct {
    Base, Height float64
}

func (triangle *Triangle) Area() float64 {
    return 0.5 * triangle.Base * triangle.Height
}

func (triangle *Triangle) Description() string {
    return fmt.Sprintf("Triangle of base %v and height %v", triangle.Base,
            triangle.Height)
}
```

Structures/Interfaces (continued)

```
func ShowAreas(shapes ...TwoDShape) {
    for _, s := range shapes {
        fmt.Printf("%s has area %.2f\n", s.Description(), s.Area())
    }
}

func main() {
    ShowAreas(&Circle{5}, &Triangle{4, 6}, &Circle{3},
            &Circle{2.5}, &Triangle{Base: 2, Height: 7.5})
}
```

```
Circle of radius 5 has area 78.54
Triangle of base 4 and height 6 has area 12.00
Circle of radius 3 has area 28.27
Circle of radius 2.5 has area 19.63
Triangle of base 2 and height 7.5 has area 7.50
```

Channels

```go
func calc_sum(c chan int, arr []int) {
    sum := 0
    for _, v := range arr {
        sum += v
    }
    c <- sum
}

func main() {
    c := make(chan int)
    go calc_sum(c, []int{1, 2, 3, 4, 5})
    go calc_sum(c, []int{8, -1, 0, 15, -25, 6, 2})
    go calc_sum(c, []int{1})
    for i := 0; i < 3; i++ {
        sum := <- c
        fmt.Printf("Calculated sum is %v\n", sum)
    }
}
```

```
Calculated sum is 15
Calculated sum is 5
Calculated sum is 1
```

General Database Support

Database Support Overview

- Package name is “database/sql”
- Intended to be generic database API for accessing any type of database
- Covers common cases but excludes database specific extensions
- Drivers implement the interfaces defined in package “database/sql/driver”
- Supports concurrent access by goroutines, context management (timeout, cancellation, etc.), connection pooling, statement caching, etc.

Database Type Support

- Only handles booleans, strings, numbers, dates
- Also has sql.NullInt64, sql.NullFloat64, sql.NullBool, sql.NullString
- Type conversion is permitted, but only if it can be done without data loss

Connecting to a Database

```
import (
    "database/sql"
    _ "gopkg.in/goracle.v2"
)

func main() {

    db, err := sql.Open("goracle", "godb/welcome@localhost/T183")
    if err != nil {
        panic(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        panic(err)
    }

}
```

Database Setup

```
create table TestQuery (
    IntCol          number(9) not null,
    StringCol       varchar2(30) not null
);

create table TestInsert (
    IntCol          number(9) not null,
    StringCol       varchar2(30) not null,
    DateCol         date not null
);

begin
    for i in 1..10 loop
        insert into TestQuery values (power(3, i), '3 ^ ' || to_char(i));
    end loop;
    commit;
end;
/
```

Multiple row query

```
rows, err := db.Query("select IntCol, StringCol from TestQuery where IntCol < 100")
if err != nil {
    panic(err);
}
defer rows.Close()

var intCol, strCol string
for rows.Next() {
    err := rows.Scan(&intCol, &strCol)
    if err != nil {
        panic(err)
    }
    fmt.Printf("IntCol=%s, StrCol=%s\n", intCol, strCol)
}
err = rows.Err()
if err != nil {
    panic(err)
}
```

```
IntCol=3, StrCol=3 ^ 1
IntCol=9, StrCol=3 ^ 2
IntCol=27, StrCol=3 ^ 3
IntCol=81, StrCol=3 ^ 4
```

Single row query

```
var intCol, strCol string
row := db.QueryRow("select IntCol, StringCol from TestQuery where IntCol = :1", 27)
err = row.Scan(&intCol, &strCol)
if err != nil {
    panic(err)
}
fmt.Printf("IntCol=%s, StrCol=%s\n", intCol, strCol)
```

```
IntCol=27, StrCol=3 ^ 3
```

DML Execution

```
var timeInserted time.Time
_, err = db.Exec("insert into TestInsert values (:IntCol, :StrCol, sysdate) " +
                 "returning DateCol into :TimeInserted",
        sql.Named("IntCol", 1),
        sql.Named("StrCol", "Test String 1"),
        sql.Named("TimeInserted", sql.Out{Dest: &timeInserted}))
if err != nil {
    panic(err);
}
fmt.Printf("Time inserted was %v\n", timeInserted)
```

```
Time inserted was 2018-10-15 13:41:41 +0000 UTC
```

Transactions

```
tx, err := db.Begin()
if err != nil {
    panic(err);
}

for i := 0; i < 5; i++ {
    _, err = tx.Exec("insert into TestInsert values (:1, :2, :3)",
           i + 1, "Test String " + strconv.Itoa(i + 1), time.Now())
    if err != nil {
        panic(err);
    }
}

err = tx.Commit()
if err != nil {
    panic(err);
}
```

Concurrent DML

```
 1 func Insert100(db *sql.DB, startingNum int, c chan int) {
 2     tx, err := db.Begin()
 3     if err != nil {
 4         panic(err)
 5     }
 6     for i := 0; i < 100; i++ {
 7         val := startingNum + i
 8         _, err := tx.Exec("insert into TestInsert values (:1, :2, :3)",
 9                 val, "Test String " + strconv.Itoa(val), time.Now())
10         if err != nil {
11             panic(err)
12         }
13     }
14     err = tx.Commit()
15     if err != nil {
16         panic(err)
17     }
18     c <- 1
19 }
```

Concurrent DML (continued)

```
const numBatches = 5
c := make(chan int)
for i := 0; i < numBatches; i++ {
    go Insert100(db, i * 100 + 1, c)
}

for i := 0; i < numBatches; i++ {
    <- c
}
```

Oracle Database Support

Oracle Database Drivers

- https://github.com/rana/ora
  - Oldest driver (about four years old)
  - Development has mostly ceased
- https://github.com/mattn/go-oci8
  - Next oldest driver (about three years old)
  - Still active
- https://github.com/go-goracle/goracle
  - Youngest driver (about a year old)
  - Most active development

Bulk DML

```
const numRows = 500
intVals := make([]int, numRows)
strVals := make([]string, numRows)
dateVals := make([]time.Time, numRows)
for i := range intVals {
    intVals[i] = i + 1
    strVals[i] = "Test String " + strconv.Itoa(i + 1)
    dateVals[i] = time.Now()
}
_, err = db.Exec("insert into TestInsert values (:1, :2, :3)", intVals,
        strVals, dateVals)
if err != nil {
    panic(err);
}
```

PL/SQL Arrays (Package Header)

```
create or replace package pkg_TestArrays as

    type udt_NumberList is table of number index by binary_integer;
    type udt_StringList is table of varchar2(100) index by binary_integer;

    procedure TestArrays (
        a_NumElems              number,
        a_OutNums               out udt_NumberList,
        a_OutStrings            out udt_StringList
    );

end;
/
```

PL/SQL Arrays (Package Body)

```
create or replace package body pkg_TestArrays as

    procedure TestArrays (
        a_NumElems              number,
        a_OutNums               out udt_NumberList,
        a_OutStrings            out udt_StringList
    ) is
    begin
        for i in 1..a_NumElems loop
            a_OutNums(i) := i * i;
            a_OutStrings(i) := 'The square of ' || to_char(i);
        end loop;
    end;

end;
/
```

PL/SQL Arrays

```
import (
    "database/sql"
    goracle "gopkg.in/goracle.v2"
)

const numElems = 5
numArr := make([]int, numElems)
strArr := make([]string, numElems)
_, err = db.Exec("begin pkg_TestArrays.TestArrays(:1, :2, :3); end;",
        goracle.PlSQLArrays, numElems, sql.Out{Dest: &numArr},
        sql.Out{Dest: &strArr})
if err != nil {
    panic(err);
}
for i := range numArr {
    fmt.Printf("%v is %v\n", strArr[i], numArr[i])
}
```

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

LOBS

```
var intCol int
var clobCol interface{}
for rows.Next() {
    err := rows.Scan(&intCol, &clobCol)
    if err != nil {
        panic(err)
    }
    if clob, ok := clobCol.(*goracle.Lob); !ok {
        panic("Not a LOB!")
    } else {
        data, err := ioutil.ReadAll(clob)
        if err != nil {
            panic(err)
        }
        fmt.Printf("IntCol=%v, CLOBCol=%v bytes\n", intCol, len(data))
    }
}
```

Nested Cursors

```
ctx, _ := context.WithTimeout(context.Background(), 10 * time.Second)
rows, err := db.QueryContext(ctx, "select cursor(select * from TestQuery) from dual")
if err != nil {
    panic(err);
}
defer rows.Close()
for rows.Next() {
    var refCursor interface{}
    if err := rows.Scan(&refCursor); err != nil {
        panic(err);
    }
    cursor, err := goracle.WrapRows(ctx, db, refCursor.(driver.Rows))
    if err != nil {
        panic(err);
    }
    defer cursor.Close()
    for cursor.Next() {
        ...
```

Other Sessions: https://tinyurl.com/AppDevOOW18

- Getting Started with GraphQL APIs on Oracle Database with Node.js [DEV4879]
  - Tuesday, Oct 23, 11:30 am - 12:15 pm | Moscone West - Room 2012
- Python and Oracle Database on the Table [TIP4076]
  - Tuesday, Oct 23, 12:30 pm - 1:15 pm | Moscone West - Room 2007
- A Database Proxy for Transparent High Availability, Performance, Routing, and Security [TRN4070]
  - Wednesday, Oct 24, 11:15 am - 12:00 pm | Moscone West - Room 3009
- Meet the Experts: Node.js, Python, PHP, and Go with Oracle Database [MTE6765]
  - Wednesday, Oct 24, 3:00 pm - 3:50 pm | Moscone West - The Hub - Lounge B
- Performance and Scalability Techniques for Oracle Database Applications [TIP4075]
  - Thursday, Oct 25, 12:00 pm - 12:45 pm | Moscone West - Room 3009
- Demo Booth: APD-A03 Moscone South

ORACLE

ORACLE®