Oracle OpenWorld 2019

SAN FRANCISCO

Using the Go Language for Efficient Oracle Database Applications [DEV6708]

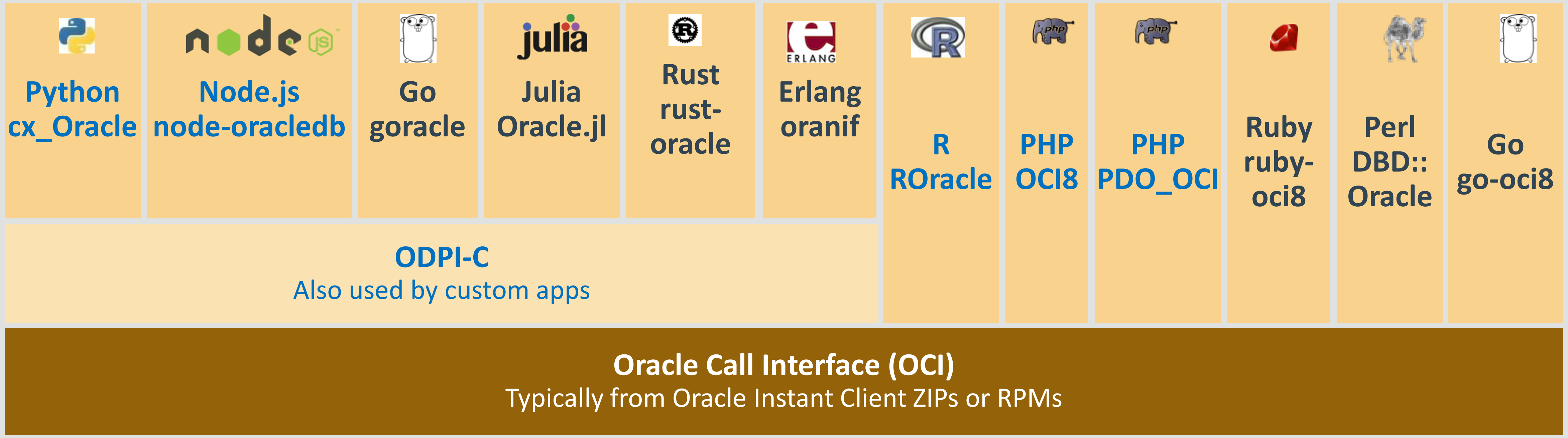
Anthony Tuininga

anthony.tuininga@oracle.com

Data Access Development
Oracle Database

Safe Harbor

Oracle Database for the Developer



🟦 Oracle Open Source Drivers

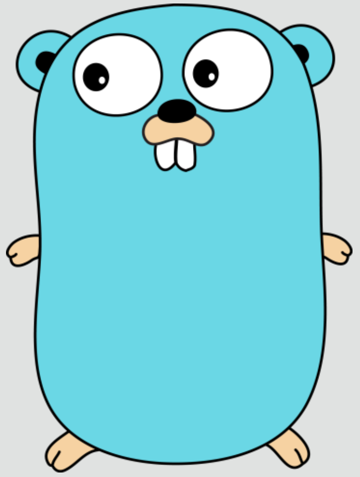
⬛ Third Party Open Source Drivers

⬜ Oracle Proprietary Drivers
Oracle Call Interface, Oracle C++ Call Interface, ODBC, JDBC, ODP.NET, Pro*C, Pro*COBOL, SQLJ, OLE DB, OLE DB for OLAP

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 1.12
  - Consideration of version 2 features underway



Popularity

- Increasing in popularity
  - Stack Overflow put Go in #13 earlier this year
  - TIOBE index for September 2019 puts Go in #14
  - IEEE puts Go at #10 in 2019
  - GitHub in 2019 puts Go as 4th for pull requests
- Docker container system and Kubernetes container management system written in Go

What Go Has

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

What Go Does 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

```go
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

Primitive 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
}
```

Structures/Interfaces (continued)

```
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)

```go
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

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

Channels (continued)

```
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, _ := sql.Open("goracle", "godb/welcome@localhost/orclpdb1")
    defer db.Close()
    db.Ping()
}
```

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
);
```

Database Setup (continued)

```
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

```go
sql := "select IntCol, StringCol from TestQuery where IntCol < 100"
rows, _ := db.Query(sql)
defer rows.Close()

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

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

DML Execution

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

```
Time inserted was 2019-09-03 12:00:09 +0000 UTC
```

Transactions

```
tx, _ := db.Begin()

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

tx.Commit()
```

Concurrent DML

```
func Insert100(db *sql.DB, startingNum int, c chan int) {
    tx, _ := db.Begin()
    for i := 0; i < 100; i++ {
        val := startingNum + i
        tx.Exec("insert into TestInsert values (:1, :2, :3)",
                val, "Test String " + strconv.Itoa(val),
                time.Now())
    }
    tx.Commit()
    c <- 1
}
```

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/go-goracle/goracle
  - Youngest driver (about two years old)
  - Most active development
  - based on ODPI-C
- https://github.com/rana/ora
  - Oldest driver (about five years old)
  - Development has mostly ceased
- https://github.com/mattn/go-oci8
  - Next oldest driver (about four years old)

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()
}
db.Exec("insert into TestInsert values (:1, :2, :3)", intVals,
        strVals, dateVals)
```

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)
db.Exec("begin pkg_TestArrays.TestArrays(:1, :2, :3); end;",
        goracle.PlSQLArrays, numElems, sql.Out{Dest: &numArr},
        sql.Out{Dest: &strArr})
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 – As String/Bytes

```
rows, _ := db.Query("select IntCol, CLOBCol from TestCLOBs")
defer rows.Close()

var intCol int
var clobCol string
for rows.Next() {
    rows.Scan(&intCol, &clobCol)
    fmt.Printf("IntCol=%v, CLOBCol=%v bytes\n", intCol,
            len(clobCol))
}
```

LOBs – As Reader

```
rows, _ := db.Query("select IntCol, CLOBCol from TestCLOBs",
          goracle.LobAsReader())
defer rows.Close()

var intCol int
var clobCol interface{}
for rows.Next() {
    rows.Scan(&intCol, &clobCol)
    clob, _ := clobCol.(*goracle.Lob)
    data, _ := ioutil.ReadAll(clob)
    fmt.Printf("IntCol=%v, CLOBCol=%v bytes\n", intCol, len(data))
}
```

Nested Cursors

```
sql := "select cursor(select * from TestQuery) from dual"
rows, _ := db.Query(sql)
defer rows.Close()
for rows.Next() {
    var nestedRows sql.Rows
    rows.Scan(&nestedRows)
    defer nestedRows.Close()
    for nestedRows.Next() {
        ...
```

AQ – Enqueue

```
ctx, _ := context.WithTimeout(context.Background(),
        10 * time.Second)
q, _ := goracle.NewQueue(ctx, db, "RAW_QUEUE", "")
defer q.Close()

tx, _ := db.Begin()
message := goracle.Message{Raw: []byte("Message 1")}
q.Enqueue([]goracle.Message{message})
tx.Commit()
```

AQ – Dequeue

```
ctx, _ := context.WithTimeout(context.Background(),
        10 * time.Second)
q, _ := goracle.NewQueue(ctx, db, "RAW_QUEUE", "")
defer q.Close()
tx, _ := db.Begin()
messages := make([]goracle.Message, 5)
n, _ := q.Dequeue(messages)
for i := 0; i < n; i++ {
    fmt.Printf("Received message: %s\n", string(messages[i].Raw))
}
tx.Commit()
```

Named Object Types – Setup

```
create or replace type udt_Book as object (
    Title          varchar2(100),
    Authors        varchar2(100),
    Price          number(5,2)
);
/

create table Books (
    Id             number(9) not null,
    Book           udt_book not null
);
```

Named Object Types – Insert

```
tx, _ := db.Begin()
ctx, _ := context.WithTimeout(context.Background(),
        30 * time.Second)
objType, _ := goracle.GetObjectType(ctx, tx, "UDT_BOOK")
obj, _ := objType.NewObject()
defer obj.Close()
obj.Set("TITLE", "The Fellowship of the Ring")
obj.Set("AUTHORS", "J.R.R. Tolkien")
obj.Set("PRICE", 12.50)
tx.Exec("insert into Books values (:1, :2)", 1, obj)
tx.Commit()
```

Named Object Types – Fetch

```
var idVal int
var obj *goracle.Object
row := db.QueryRow("select Id, Book from Books")
row.Scan(&idVal, &obj)
defer obj.Close()
authors, _ := obj.Get("AUTHORS")
title, _ := obj.Get("TITLE")
price, _ := obj.Get("PRICE")
fmt.Printf("Authors: %s\nTitle: %s\nPrice: %.2f\n", authors, title,
        price)
```

Resources

Mail:	anthony.tuininga@oracle.com
Twitter:	@AnthonyTuininga
Facebook:	https://tinyurl.com/FBOracle
goracle:	https://github.com/go-goracle/goracle

Thank You

Anthony Tuininga

anthony.tuininga@oracle.com

Data Access Development
Oracle Database