DWARVES FOUNDATION

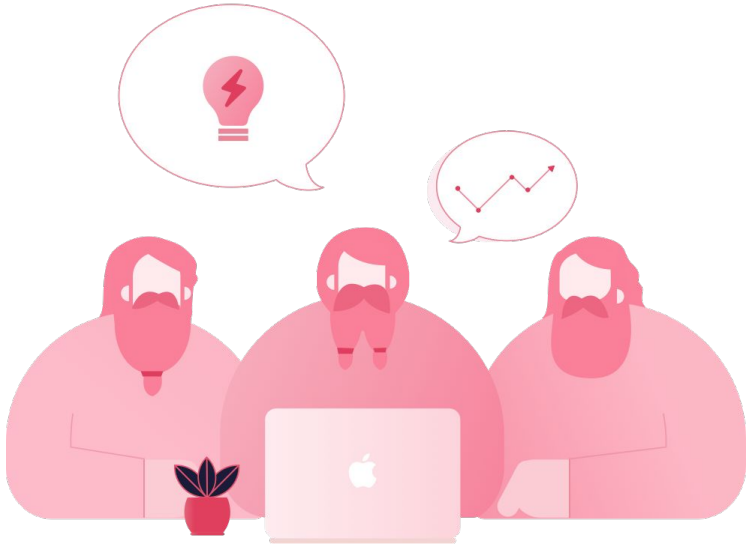
Database Access and ORM

Hieu Phan

Engineer Lead

@hieupq

andy@dwarvesv.com



Agenda

Topic summary

1. SQL Databases and Executing Queries
2. Object-Relational Mapping (ORM)
3. GORM and XORM in Golang
4. Demo

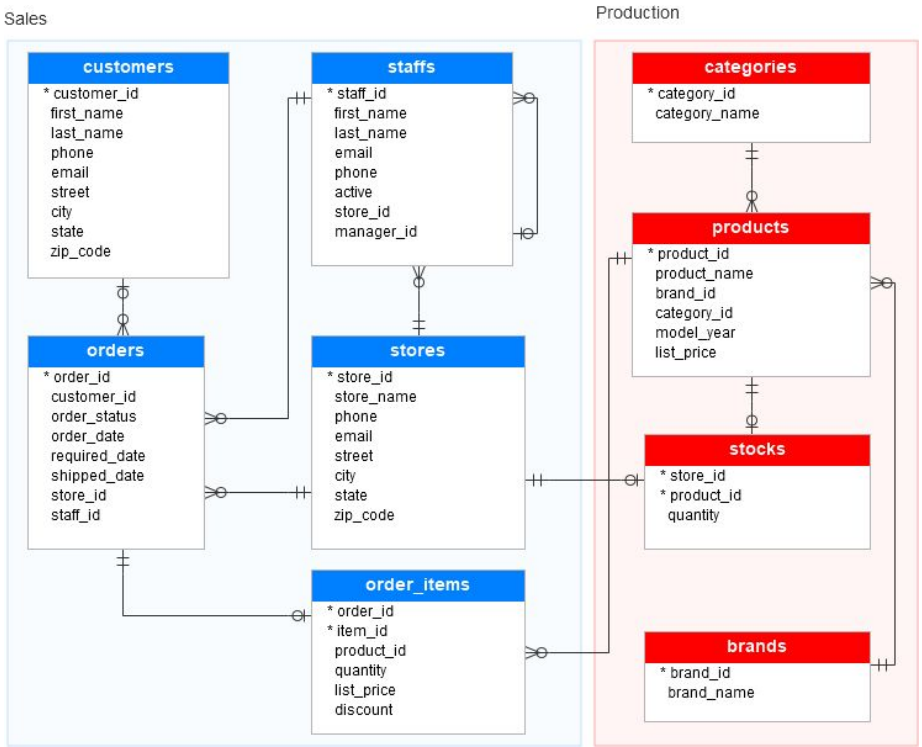
SQL Databases and Executing Queries

SQL Databases

- SQL stands for Structured Query Language
- SQL became a standard language in 1986 by the American National Standards Institute (ANSI) and later adopted by the International Organization for Standardization (ISO)

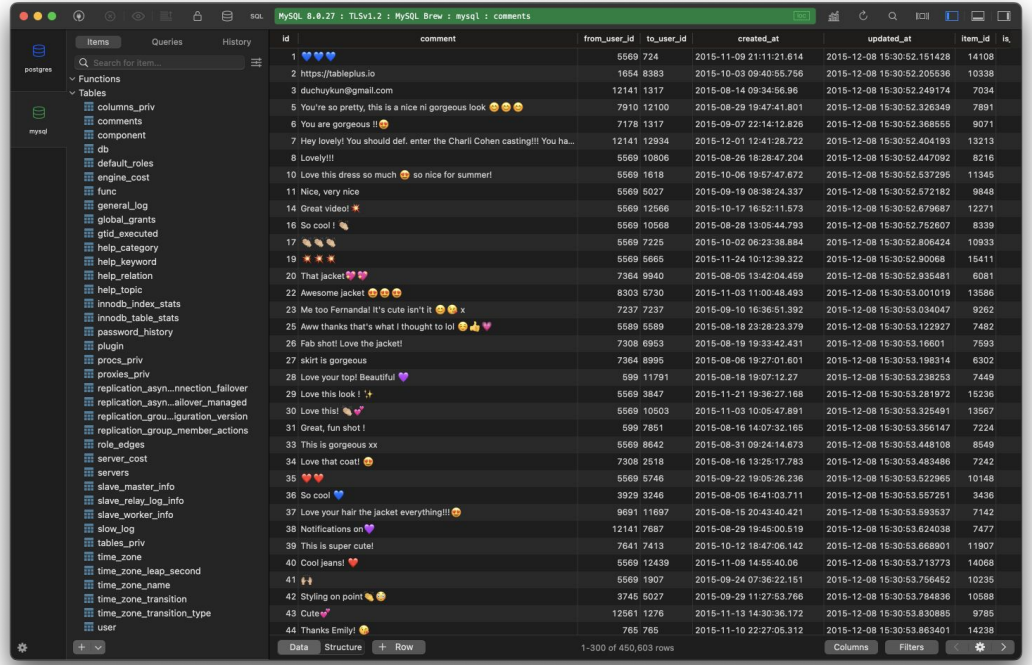
SQL Databases

- SQL Databases are often Relational Database Management Systems
- Data is mapped by 1-to-1, 1-to-many, and many-to-many relationships
- Relationships can be used to help keep data integrity and for joining data together



Executing SQL

- SQL is a standard language for managing and querying data in RDBMS
- Use tools like Beekeeper Studio or SQL or TablePlus to execute queries



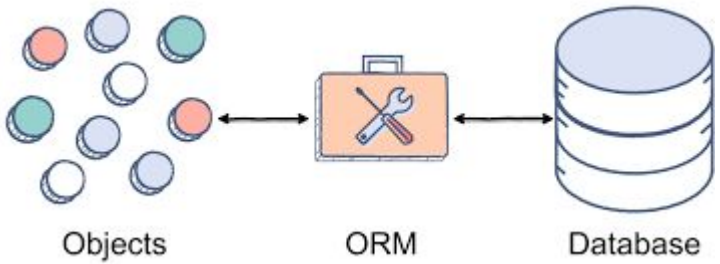
Object-Relational Mapping (ORM)

ORMs

- Object-Relational Mapping (ORM)

  is a technique used in software development to create a bridge between object-oriented programs and relational databases

- ORM tools automate the process of mapping objects to database tables, handling database operations, and managing the persistence of data



GORM and XORM in Golang

GORM and XORM

- GORM follows a code-first approach, where you define Go structs as models and GORM automatically generates the corresponding database tables[3].
- It provides functionalities for schema auto-migration, logging, contexts, prepared statements, associations, constraints, advanced database operations like sharding, and more.

- XORM follows a join function design approach, which allows you to perform database operations with less code.
- It provides features like cache support, transactions, optimistic locking, multiple database support, and reverse engineering tools.

GORM

- We first open a connection to a sqlite database file. Then use AutoMigrate to create the schema for the Product model.
- To create a new product, we can use db.Create() and pass a Product struct.
- To read a product, we use db.First() which will find the product by primary key ID.
- We can then print the product details.
- This covers the basic CRUD operations for database interactions using GORM in Go.

```go
package main

import (
  "gorm.io/driver/sqlite"
  "gorm.io/gorm"
)

type Product struct {
  gorm.Model
  Code string
  Price uint
}

func main() {
  db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
  if err != nil {
    panic("Failed to connect to database")
  }

  // Migrate schema
  db.AutoMigrate(&Product{})

  // Create product
  db.Create(&Product{Code: "D42", Price: 100})

  // Read product
  var product Product
  db.First(&product, 1)

  // Print product
  println(product.Code) // D42
}
```

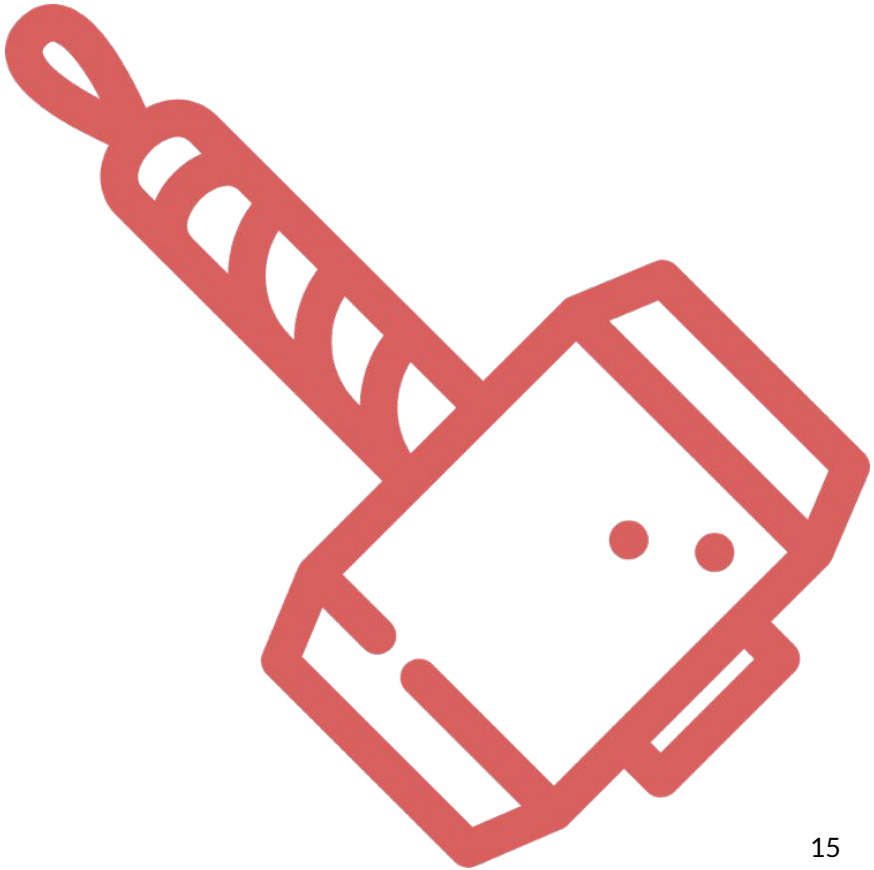
XORM

- We use Sync2() to migrate the schema for the Product model.
- To insert a new product, we can use engine.Insert() and pass the Product struct.
- To query, we use engine.ID(id).Get() to select by primary key.
- The main differences from GORM are the function names like Insert vs Create, and using Get() vs First() for querying. But overall very similar CRUD principles.

```go
package main

import (
  "fmt"

  _ "github.com/mattn/go-sqlite3"
  "xorm.io/xorm"
)

type Product struct {
  Id    int64  `xorm:"pk autoincr"`
  Code  string `xorm:"varchar(255)"`
  Price int64
}

func main() {
  engine, err := xorm.NewEngine("sqlite3", "test.db")
  if err != nil {
    panic(err)
  }

  // Migrate schema
  engine.Sync2(new(Product))

  // Insert product
  product := Product{Code: "D42", Price: 100}
  _, err = engine.Insert(&product)

  // Query product
  var p Product
  has, err := engine.ID(1).Get(&p)

  fmt.Printf("Product: %v\n", p) // Product: {1 D42 100}
}
```

Demo

Demo - Zer0 to Hero



Reference

Resources & Reference links

- https://gorm.io/
- https://xorm.io/

DWARVES FOUNDATION

Thank You

DWARVES FOUNDATION

Q&A