2025's Ultimate Guide to Using Golang PostgreSQL

Database Management System | PostgreSQL

Updated
November 19, 2024

By
Manisha Jena

Time to read
13 mins read

For building scalable and efficient applications, integrating with a reliable and robust database system is crucial. PostgreSQL is one such popular **relational database management system** that forms a formidable duo with the programming language Golang(Go).

In this comprehensive guide, we will take you through the Golang PostgreSQL setup, along with advanced topics such as CRUD(Create, Read, Update, Delete) operations and also performing complex queries. Read on to understand the **need to connect PostgreSQL with Golang and harness** the power of PostgreSQL into your Golang applications.

Go, often known as **Golang**, is an open-source, compiled, and statically typed programming language developed by Google. It is designed to be simple, fast, readable, and efficient.

Go was created in 2007 at Google to boost programming efficiency in an era of multicore, networked devices, and enormous codebases. The creators aimed to respond to criticism of existing languages used at Google while retaining their desirable characteristics:

- Run-time efficiency and static typing (like C)
- Usability and readability (like Python or JavaScript)
- Multiprocessing and high-performance networking

You can also execute CRUD operations of PostgreSQL using GO. For further information about Golang, you can visit the official website.

Golang PostgreSQL Overview

Here is a Golang PostgreSQL tutorial with a step-by-step process to help you understand the implementation of the Golang PostgreSQL connection.

- Golang PostgreSQL: Creating a Database
- Golang PostgreSQL: Connecting Golang to PostgreSQL
- Golang PostgreSQL: Insert data into Table
- Golang PostgreSQL: Update data into Table
- Golang PostgreSQL: Delete data from Table
- Golang PostgreSQL: Performing Queries
- Golang PostgreSQL: Retrieving Records

1) Golang PostgreSQL: Creating a database

You can create a database in the PostgreSQL shell. In this database, you can further create and execute tables and all other functionalities after connecting Golang PostgreSQL.

Connect to the freshly created database using the meta-command ‘c’ followed by the database name, as illustrated here:

```
\c DB_1;
```

Now, after moving to the database, you can create a table within the database using the following command:

```
CREATE TABLE Students (
Name TEXT,
Roll_number INT PRIMARY KEY,
);
```

2) Golang PostgreSQL: Connecting Golang to the PostgreSQL Database

Create a file called main.go that can be used to connect Golang to the PostgreSQL database. You can refer to the following Golang script to code the connection information into this file:

- **Example 1**

```
package main

import (
"fmt"
"database/sql"
"net/http"
"log"
_ "github.com/lib/pq"
)
```

```
database only once.
func init() {
var err error

connStr :=
"postgres://postgres:password@localhost/DB_1?
sslmode=disable"
db, err = sql.Open("postgres", connStr)

if err != nil {
panic(err)
}

if err = db.Ping(); err != nil {
panic(err)
}
// this will be printed in the terminal, confirming
the connection to the database
fmt.Println("The database is connected")
}
```

- **Example 2**

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
```

```
    dbname   = "DB_1"
)

func main() {
        // connection string
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s
password=%s dbname=%s sslmode=disable", host, port,
user, password, dbname)

        // open database
    db, err := sql.Open("postgres", psqlconn)
    CheckError(err)

        // close database
    defer db.Close()

        // check db
    err = db.Ping()
    CheckError(err)

    fmt.Println("Connected!")
}

func CheckError(err error) {
    if err != nil {
        panic(err)
    }
}
```

3) Golang PostgreSQL: Insert data into Table

Using Golang along with PostgreSQL, you can create a table and then insert the records into the table. The code written to insert data into the

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5400
    user     = "postgres"
    password = "man1234"
    dbname   = "DB_1"
)

func main() {
    psqlconn := fmt.Sprintf("host=%s port=%d user=%s
password=%s dbname=%s sslmode=disable", host, port,
user, password, dbname)

    db, err := sql.Open("postgres", psqlconn)
    CheckError(err)

    defer db.Close()

    // insert
    // hardcoded
    insertStmt := `insert into "Students"("Name",
"Roll_Number") values('Jacob', 20)`
    _, e := db.Exec(insertStmt)
    CheckError(e)

    // dynamic
    insertDynStmt := `insert into "Students"("Name",
```

```
}

func CheckError(err error) {
    if err != nil {
        panic(err)
    }
}
```

4) Golang PostgreSQL: Update data into Table

In Golang, you can use the following command to update your data in the table you have created.

```
// update
updateStmt := `update "Students" set "Name"=$1,
"Roll_Number"=$2 where "id"=$3`
_, e := db.Exec(updateStmt, "Rachel", 24, 8)
CheckError(e)
```

You can check your updated table by using the following command in the PostgreSQL shell.

```
Select * from Students;
```

The output will show all the records in the Students table.

5) Golang PostgreSQL: Update data into Table

In Golang, you can use the following command to delete any row in the table you have created.

```
// Delete
deleteStmt := `delete from "Students" where id=$1`
_, e := db.Exec(deleteStmt, 1)
CheckError(e)
```

You can check your updated table by using the following command in the PostgreSQL shell.

```
Select * from Students;
```

The output will show all the records in the Students table.

6) Golang PostgreSQL: Performing Queries

In Golang, you can use the following command to query the records from your table.

```
rows, err := db.Query(`SELECT "Name", "Roll_Number"
FROM "Students"`)
CheckError(err)

defer rows.Close()
for rows.Next() {
```

```
    err = rows.Scan(&name, &roll_number)
    CheckError(err)

    fmt.Println(name, roll_number)
}

CheckError(err)
```

7) Golang PostgreSQL: Retrieving Records

The different sections while retrieving records using Golang PostgreSQL connection are as follows:

A) The GoLang struct() comand

The struct command is used to build a collection of fields that correspond to the fields in a given table. These fields will be used as placeholders for the values of the columns in the table.

```
type sandbox struct {
id int
Firstname string
Lastname string
Age int
}
```

B) The package main command

The Golang package main command that follows directs the Golang compiler to create the file as an executable file:

C) Importing the Golang dependencies

The following dependencies must be imported to access the methods that will aid with database interaction. To import the dependencies, you can use the following command:

```
import (

"database/sql"
_ "github.com/lib/pq"
"fmt"
"net/http"

)
```

Following is a breakdown for importing the Golang dependencies:

- Importing the **database/sql** allows it to interact idiomatically with the database.
- The **pq package**, the Golang PostgreSQL driver, is preceded by an **underscore _**, which instructs Golang to import the package whether or not it is explicitly utilized in the code.
- To perform any sort of formatting in the string displays, the I/O formatting must be employed. This is why the **fmt package** is required.
- Finally, for client/server communication, import the **net/http** package.

D) Retrieving a PostgreSQL Record

The code to retrieve a PostgreSQL record is as follows:

```
func retrieveRecord(w http.ResponseWriter, r
*http.Request) {
```

```
http.Error(w, http.StatusText(405),
http.StatusMethodNotAllowed)
return
}

// We assign the result to 'rows'
rowsRs, err := db.Query("SELECT * FROM Students")

if err != nil {
http.Error(w, http.StatusText(500),
http.StatusInternalServerError)
return
}
defer rowsRs.Close()


// creates placeholder of the sandbox
snbs := make([]sandbox, 0)


// we loop through the values of rows
for rows.Next() {
snb := sandbox{}
err := rowsRs.Scan(&snb.name, &snb.roll_number)
if err != nil {
log.Println(err)
http.Error(w, http.StatusText(500), 500)
return
}
snbs = append(snbs, snb)
}

if err = rowsRs.Err(); err != nil {
http.Error(w, http.StatusText(500), 500)
```

```
// loop and display the result in the browser
for _, snb := range snbs {
fmt.Fprintf(w, "%d %s %s %d\n", snb.name,
snb.roll_number)
}

}
```

Example of Using Go with Postgres

To demonstrate Go's integration with PostgreSQL, let's create a simple application that connects to a Postgres database, inserts a record, and retrieves it. Here's Golang Postgres example:

1. Setup

Install the required PostgreSQL driver for Go:

```
go get github.com/lib/pq
```

2. Example

```
package main

import (
        "database/sql"
        "fmt"
        "log"

        _ "github.com/lib/pq"
)
```

```
password=yourpassword dbname=yourdb sslmode=disable"
        db, err := sql.Open("postgres", connStr)
        if err != nil {
                log.Fatal(err)
        }
        defer db.Close()

        // Insert data
        insertSQL := `INSERT INTO users (name, age)
VALUES ($1, $2) RETURNING id`
        var id int
        err = db.QueryRow(insertSQL, "Alice",
25).Scan(&id)
        if err != nil {
                log.Fatal(err)
        }
        fmt.Printf("Inserted record ID: %d\n", id)

        // Retrieve data
        querySQL := `SELECT name, age FROM users
WHERE id = $1`
        var name string
        var age int
        err = db.QueryRow(querySQL, id).Scan(&name,
&age)
        if err != nil {
                log.Fatal(err)
        }
        fmt.Printf("Retrieved Record - Name: %s, Age:
%d\n", name, age)
}
```

- **Connecting to the database**: Replace `username`, `yourpassword`, and `yourdb` with your database credentials.

Querying data: The `SELECT` query retrieves a specific record using the ID.

Why connect PostgreSQL with Golang?

Here are some key reasons to connect PostgreSQL with Golang:

1. Golang's performance in building fast and scalable network applications makes it a great fit to take advantage of PostgreSQL's reliability and scalability as a database server.
2. PostgreSQL's ACID compliance provides reliable transactions critical for data integrity. Paired with Golang's built-in concurrency primitives, both ensure robust data applications.
3. PostgreSQL's native JSON and JSONB column types mesh well with Go's great JSON support to build modern data-driven REST APIs faster.
4. The golang database/sql package along with postgresql driver provide a simple database/sql interface for programmatic access to PostgreSQL databases avoiding complex querying.
5. Golang compiles into standalone machine code making lightweight and portable applications that simplify DevOps deployment whether on-premise or cloud (Kubernetes etc).

Also, take a look at how you can setup Django PostgreSQL Connection and also understand the basic schema operations you can perform on PostgreSQL.

Best Practices for Golang and PostgreSQL Integration

- **Use Environment Variables:** Store sensitive credentials like database URLs securely in environment variables.
- **Use Connection Pooling:** Utilize libraries like `pgx` or `sql.DB` for efficient connection management.

Leverage Transactions: Use transactions for operations requiring atomicity to ensure data consistency.

- **Handle Errors Gracefully:** Check for and log errors at every step to simplify debugging.

Conclusion

This article illustrated the need to connect **Golang PostgreSQL**. You had an in-depth understanding of all the steps involved in implementing the PostgreSQL Golang connection and the different operations executed.

Now, you can move forward and create your application in Golang backed by PostgreSQL.

Ingesting and transferring data into your desired warehouse using ETL for Data Analysis may be a time-consuming procedure with PostgreSQL as your data source. The problem gets much more overwhelming when you consider how much money and resources are necessary to hire data engineers and analysts to make sense of this data.

However, with **Hevo** at your fingertips, you won't have to worry about your **PostgreSQL Data Replication** demands. Loading your data into your selected data warehouse will only take a few minutes.

Sign up for a 14-day free trial and simplify your data integration process. Check out the pricing details to understand which plan fulfills all your business needs.

FAQs

1. Does Postgres use Golang?

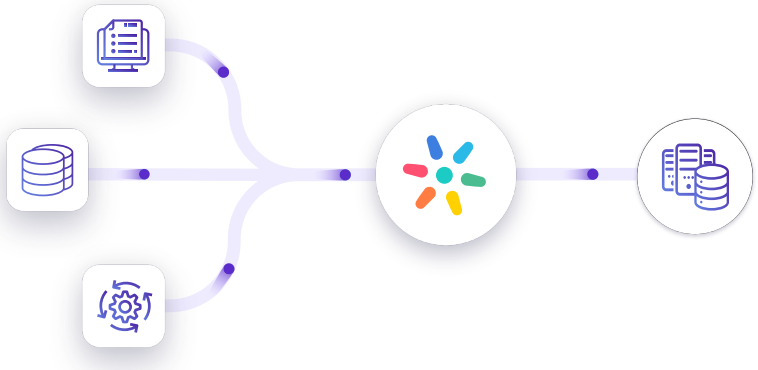
PostgreSQL itself does not use Golang; it is written in C. However, Golang (Go) can be used to interact with PostgreSQL databases through various client libraries.

3. How to connect Postgres to Golang?

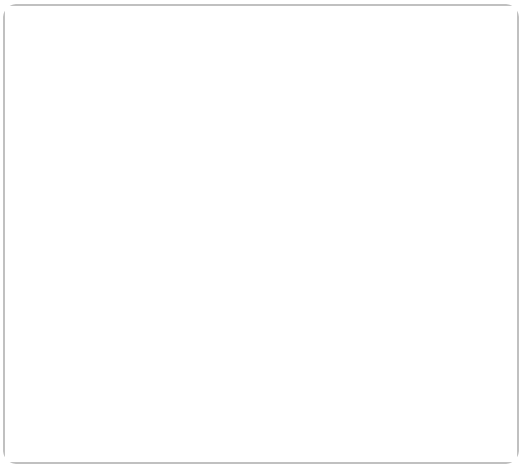
Manisha Jena

Research Analyst, Hevo Data

Manisha Jena is a data analyst with over three years of experience in the data industry and is well-versed with advanced data tools such as Snowflake, Looker Studio, and Google BigQuery. She is an alumna of NIT Rourkela and excels in extracting critical insights from complex database...

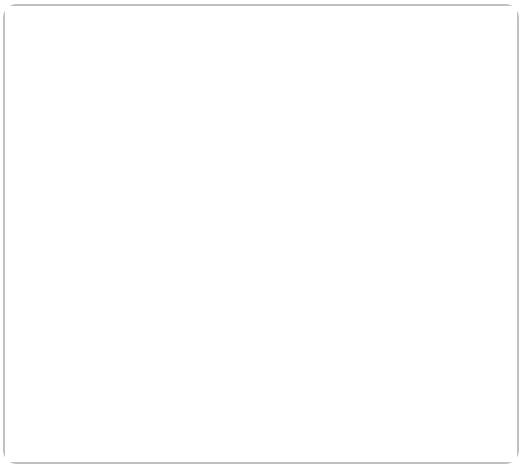


Related articles



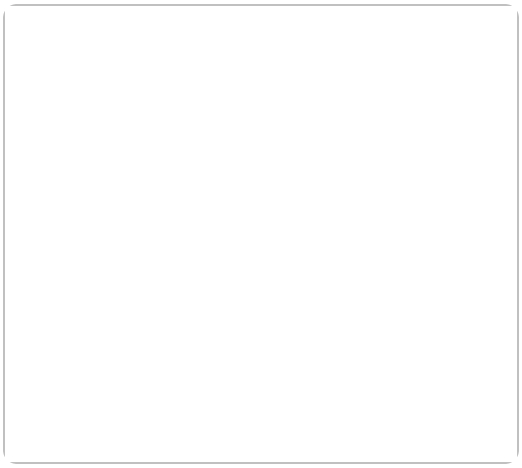
Kamya

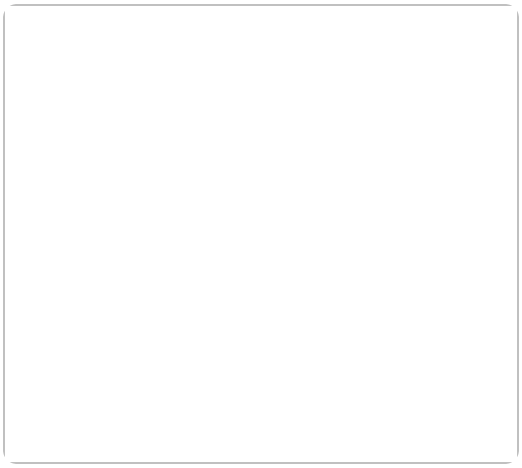
Loop in PostgreSQL: Syntax & Operations Simplified with Examples



Manisha Jena

Working with PostgreSQL Subquery | A 101 Guide





Step Explanation

Sign Up

Cookies Settings Privacy Policy Terms of Use