Database

Grafana uses a database to persist settings between restarts. In fact, if you don't specify one, Grafana creates a SQLite3 database file on your local disk. This guide explains how to store and retrieve data from the database.

Grafana supports the following databases:

- MySQL
- PostgreSQL
- SQLite3

Grafana uses the XORM framework for persisting objects to the database. For more information on how to use XORM, refer to the documentation.

Services don't use XORM directly. Instead, services use the *SQL store*, a special type of service that provides an abstraction for the database layer. There are two ways of using the `sqlstore`: using `sqlstore` handlers, and using the `SQLStore` instance.

`sqlstore` handlers

> ***Deprecated:*** *We are deprecating `sqlstore` handlers in favor of using the `SQLStore` object directly in each service. Since most services still use the `sqlstore` handlers, we still want to explain how they work.*

The `sqlstore` package allows you to register command handlers that either store, or retrieve objects from the database. `sqlstore` handlers are similar to services:

- Services are command handlers that *contain business logic*.
- `sqlstore` handlers are command handlers that *access the database*.

Register a `sqlstore` handler

> ***Deprecated:*** *Refer to the deprecation note for `sqlstore` handlers.*

To register a handler:

- Create a new file `myrepo.go` in the `sqlstore` package.
- Create a command handler.
- Register the handler in the `init` function:

```
func init() {
    bus.AddHandlerCtx("sql", DeleteDashboard)
}

func DeleteDashboard(ctx context.Context, cmd *models.DeleteDashboardCommand) error
{
    return inTransactionCtx(ctx, func(sess *DBSession) error {
        _, err := sess.Exec("DELETE FROM dashboards WHERE dashboard_id=?",
cmd.DashboardID)
        return err
    })
}
```

Here, `inTransactionCtx` is a helper function in the `sqlstore` package that provides a session, that lets you execute SQL statements.

`SQLStore`

As opposed to a `sqlstore` handler, the `SQLStore` is a service itself. The `SQLStore` has the same responsibility however: to store and retrieve objects, to and from the database.

To use the `SQLStore`, inject it in your service struct:

```
type MyService struct {
    SQLStore *sqlstore.SQLStore `inject:""`
}
```

You can now make SQL queries in any of your command handlers or event listeners:

```
func (s *MyService) DeleteDashboard(ctx context.Context, cmd
*models.DeleteDashboardCommand) error {
    if err := s.SQLStore.WithDbSession(ctx, func(sess *sqlstore.DBSession) error {
        _, err := sess.Exec("DELETE FROM dashboards WHERE dashboard_id=?",
cmd.DashboardID)
        return err
    })
}
```

For transactions, use the `WithTransactionalDbSession` method instead.

Migrations

As Grafana evolves, it becomes necessary to create *schema migrations* for one or more database tables.

To see all the types of migrations you can add, refer to migrations.go.

Before you add a migration, make sure that you:

- Never change a migration that has been committed and pushed to main.
- Always add new migrations, to change or undo previous migrations.

Add a migration using one of the following methods:

- Add migrations in the `migrations` package.
- Implement the `DatabaseMigrator` for the service.

Important: If there are previous migrations for a service, use that method. By adding migrations using both methods, you risk running migrations in the wrong order.

Add migrations in `migrations` package

Most services have their migrations located in the migrations package.

To add a migration:

- Open the migrations.go file.

- In the `AddMigrations` function, find the `addXxxMigration` function for the service you want to create a migration for.
- At the end of the `addXxxMigration` function, register your migration:

Example

Implement `DatabaseMigrator`

During initialization, SQL store queries the service registry, and runs migrations for every service that implements the DatabaseMigrator interface.

To add a migration:

- If needed, add the `AddMigration(mg *migrator.Migrator)` method to the service.
- At the end of the `AddMigration` method, register your migration:

```
func (s *MyService) AddMigration(mg *migrator.Migrator) {
    // ...

    mg.AddMigration("Add column age", NewAddColumnMigration(table, &Column{
        Name:     "age",
        Type:     migrator.DB_BigInt,
        Nullable: true,
    }))
}
```