**MIGRATION**

**Migration:**

Migrations are like version control for your database, allowing your team to define and share the application's database schema definition. If you have ever had to tell a teammate to manually add a column to their local database schema after pulling in your changes from source control, you've faced the problem that database migrations solve.

The **Laravel Schema** [**facade**](https://laravel.com/docs/8.x/facades)provides database agnostic support for **creating** and **manipulating** tables across all of Laravel's supported database systems. Typically, migrations will use this facade to create and modify database tables and columns.

**php artisan config:Cache**

**php artisan make:migration create\_posts\_table**

**php artisan make: migration create\_posts\_table --create=posts // with structure**

**Php artisan migrate**

public function up()

{

Schema::create('clients', function (Blueprint $table) {

$table->id();//$table->increments('id');

or

$table->id(‘custome\_id’); // you can also add custom id

// it provide default created\_at/ updated\_at column by default

$table->timestamps();

$table->integer('votes');

$table->string('username'); // default size 255

$table->char('name', 100)->nullable();

$table->char('name', 100)->index();

$table->char('name', 100)->unique();

$table->text('description1');

$table->longText('description');

$table->timeTz('sunrise', $precision = 0);

$table->dateTimeTz('created\_at1', $precision = 0);

$table->double('amount1', 8, 2);

$table->float('amount', 8, 2); $table->enum('status',['Block','Unblock'])->default('Unblock');

$table->bigInteger('mobile');

$table->unsignedBigInteger('cate\_id'); // foreign key declare $table->foreign(‘cate\_id’)->references(‘id’)->on(‘categories’);

$table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade')

});

}

**Add column after create table**

**php artisan make:migration create\_countries\_table**

**php artisan make:migration add\_columns\_to\_countries\_table**

1. **How to do config database in laravel.**

Ans.

**Step 1: Install Laravel**

composer create-project --prefer-dist laravel/laravel myapp

**Step 2: Set Up Environment Variables**

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=your\_database\_name

DB\_USERNAME=your\_database\_user

DB\_PASSWORD=your\_database\_password

**Step 3: Update the Configuration File**

'mysql' => [

'driver' => 'mysql',

'host' => env('DB\_HOST', '127.0.0.1'),

'port' => env('DB\_PORT', '3306'),

'database' => env('DB\_DATABASE', 'forge'),

'username' => env('DB\_USERNAME', 'forge'),

'password' => env('DB\_PASSWORD', ''),

'unix\_socket' => env('DB\_SOCKET', ''),

'charset' => 'utf8mb4',

'collation' => 'utf8mb4\_unicode\_ci',

'prefix' => '',

'strict' => true,

'engine' => null,

],

**Step 4: Run Database Migrations**

php artisan migrate

1. **Call MySQLi Store Procedure from Laravel.**

Ans.

**Step 1: Add MySQLi to Your Project**

**Install MySQLi**

MySQLi is a PHP extension and should be available in your PHP installation. Ensure it is enabled in your php.ini file.

**Create a MySQLi Service Provider**

To integrate MySQLi with Laravel, you can create a service provider. Run the following command to generate a service provider:

php artisan make:provider MySQLiServiceProvider

**Step 2: Calling a Stored Procedure**

1. **Create a Controller**:

php artisan make:controller ProcedureController

1. **Call the Stored Procedure**:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProcedureController extends Controller

{

public function callProcedure()

{

$mysqli = app('mysqli');

// Assuming you have a stored procedure named `getUsers`

$stmt = $mysqli->prepare("CALL getUsers()");

if ($stmt) {

$stmt->execute();

$result = $stmt->get\_result();

$users = [];

while ($row = $result->fetch\_assoc()) {

$users[] = $row;

}

$stmt->close();

return response()->json($users);

} else {

return response()->json(['error' => 'Failed to prepare statement'], 500);

}

}

}

**Step 3: Define Routes**

**Route::get('/call-procedure', [ProcedureController::class, 'callProcedure']);**

1. **Apply Curd Operation through Query Builder for Employee Management.**

Ans.

**Step 1: Set Up the Database and Model**

database and create an **employees** table.

**Migration**

php artisan make:migration create\_employees\_table --create=employees

**Run the migration:**

php artisan migrate

**Model**

php artisan make:model Employee

**Step 2: Create a Controller for Employee Management**

**Generate a controller:**

php artisan make:controller EmployeeController

**Step 3: Implement CRUD Operations**

**Import the DB facade:**

use Illuminate\Support\Facades\DB;

**Create Operation**

public function create(Request $request)

{

$validated = $request->validate([

'name' => 'required|string|max:255',

'email' => 'required|string|email|max:255|unique:employees',

'position' => 'required|string|max:255',

'salary' => 'required|numeric',

]);

DB::table('employees')->insert([

'name' => $validated['name'],

'email' => $validated['email'],

'position' => $validated['position'],

'salary' => $validated['salary'],

'created\_at' => now(),

'updated\_at' => now(),

]);

return response()->json(['message' => 'Employee created successfully.'], 201);

}

**Read Operation**

public function index()

{

$employees = DB::table('employees')->get();

return response()->json($employees);

}

**Update Operation**

public function update(Request $request, $id)

{

$validated = $request->validate([

'name' => 'sometimes|string|max:255',

'email' => 'sometimes|string|email|max:255|unique:employees,email,'.$id,

'position' => 'sometimes|string|max:255',

'salary' => 'sometimes|numeric',

]);

$updated = DB::table('employees')

->where('id', $id)

->update(array\_merge($validated, ['updated\_at' => now()]));

if ($updated) {

return response()->json(['message' => 'Employee updated successfully.']);

} else {

return response()->json(['message' => 'Employee not found or no changes made.'], 404);

}

}

**Delete Operation**

public function destroy($id)

{

$deleted = DB::table('employees')->where('id', $id)->delete();

if ($deleted) {

return response()->json(['message' => 'Employee deleted successfully.']);

} else {

return response()->json(['message' => 'Employee not found.'], 404);

}

}

**Step 4: Define Routes**

use App\Http\Controllers\EmployeeController;

Route::get('/employees', [EmployeeController::class, 'index']);

Route::get('/employees/{id}', [EmployeeController::class, 'show']);

Route::post('/employees', [EmployeeController::class, 'create']);

Route::put('/employees/{id}', [EmployeeController::class, 'update']);

Route::delete('/employees/{id}', [EmployeeController::class, 'destroy']);

1. **Create All Migration for Employee management.**

Ans.

**Step 1: Create the Employees Table**

php artisan make:migration create\_employees\_table --create=employees

**Step 2: Create the Departments Table**

php artisan make:migration create\_departments\_table --create=departments

**Step 3: Create the Employee\_Department Pivot Table**

php artisan make:migration create\_employee\_department\_table --create=employee\_department

**Step 4: Create the Roles Table**

php artisan make:migration create\_roles\_table --create=roles

**Step 5: Create the Employee\_Role Pivot Table**

php artisan make:migration create\_employee\_role\_table --create=employee\_role