Instituto Politécnico Nacional
Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Ingeniería Telemática.

Bases de Datos Distribuidos.

Integrantes: **Lisardo René Morgado Reséndiz.**
Moreno Galicia Jesús Antonio.
López Navarrete Sergio Hidekel

Profesor: De La Cruz Sosa Carlos.

Grupo: 3TM3.

Fecha: miércoles 09 de marzo de 2022.

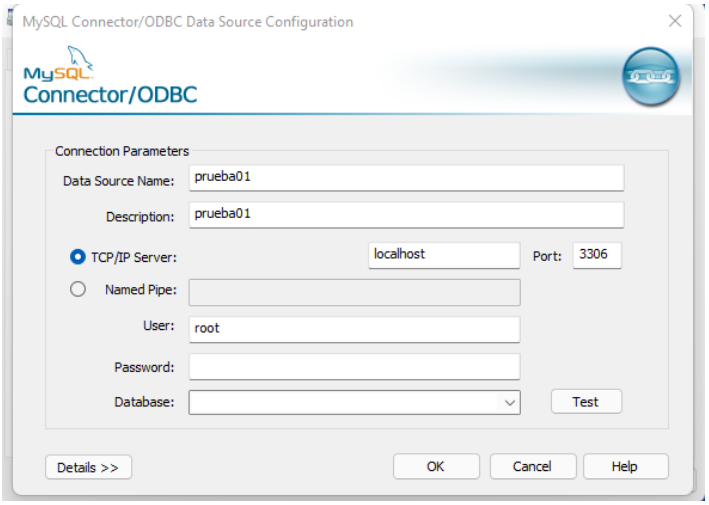
Practica DAO.

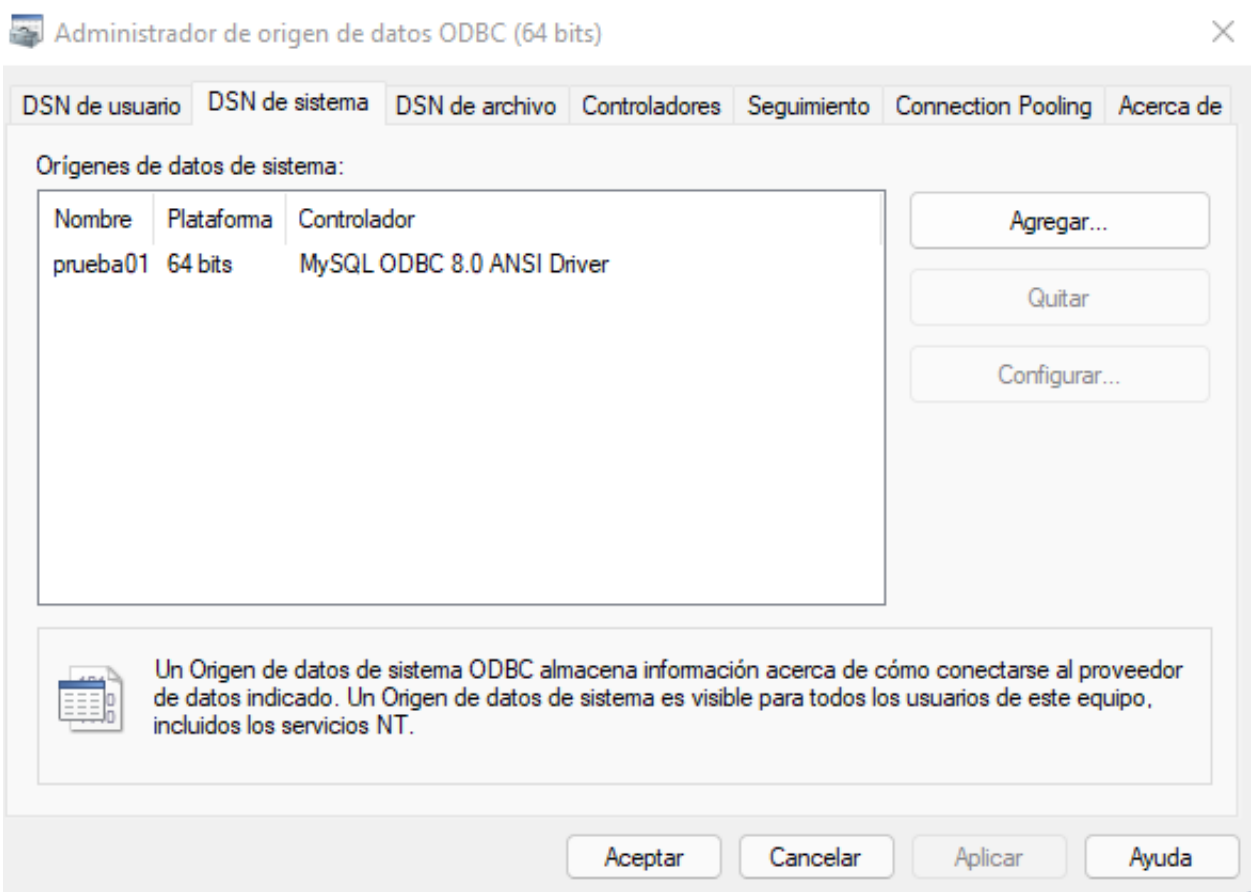
(DAO para gestionar dos bases de datos en dos instancias de BD relacionales SQL Server y MySQL.)

Configuración de los servidores vinculados.

En esta práctica implementamos dos diferentes proveedores de bases de datos una instancia SQL Server y una MySQL, con la cual vamos a realizar una conexión que implementaremos en la arquitectura DAO.

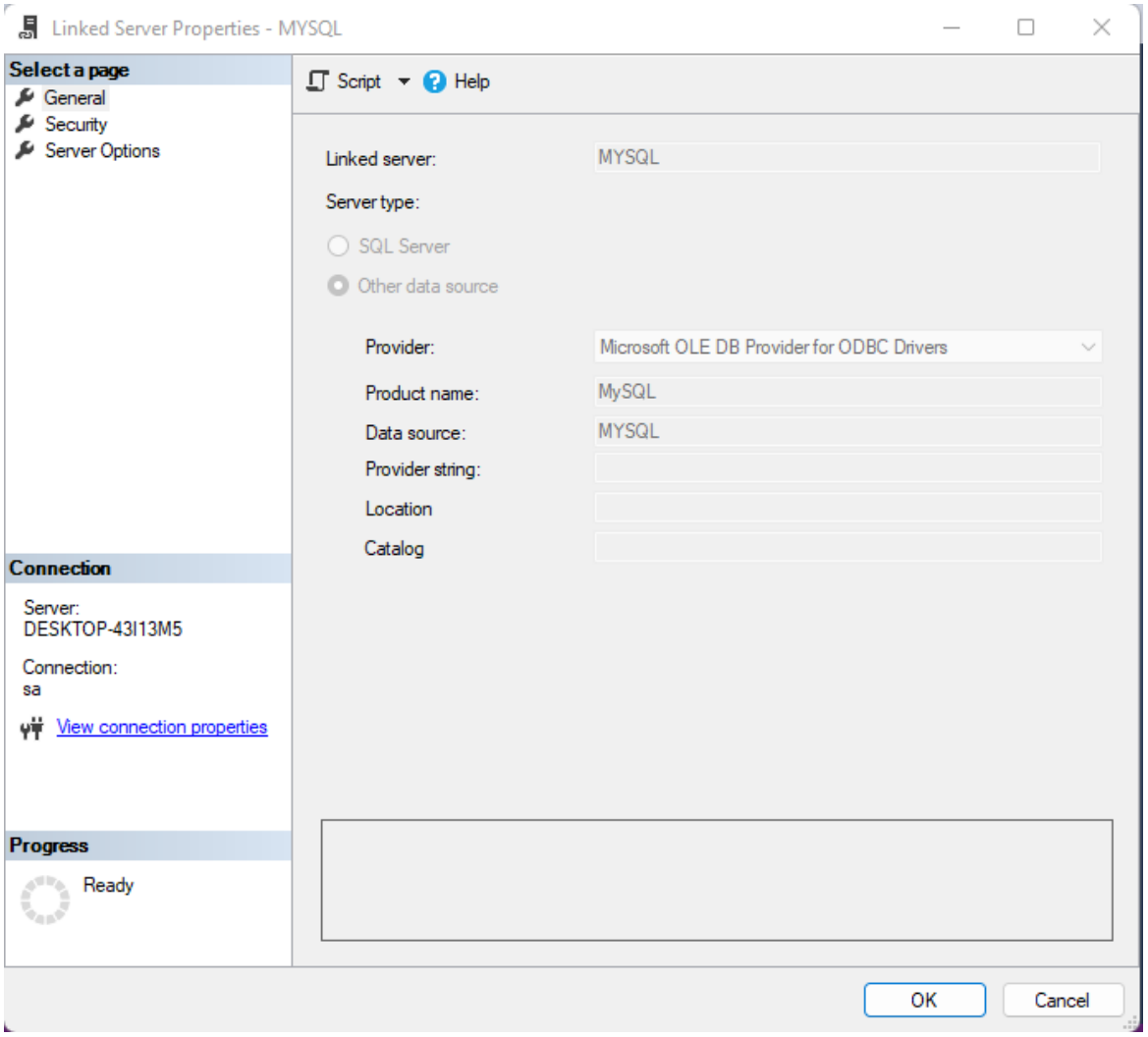
Configuramos el ODBC de MySQL con el cual vamos a establecer una conexión al esquema que tenemos alojado en MySQL. Para ello debemos descargar el OBDC directamente de la página de MySQL.





Con el ODBC nos permitirá realizar la conexión a la base de datos MySQL. Posteriormente crearemos los Linked service para establecer las conexiones entre SQL Server y MySQL

Realizamos la configuración del Linked Server.

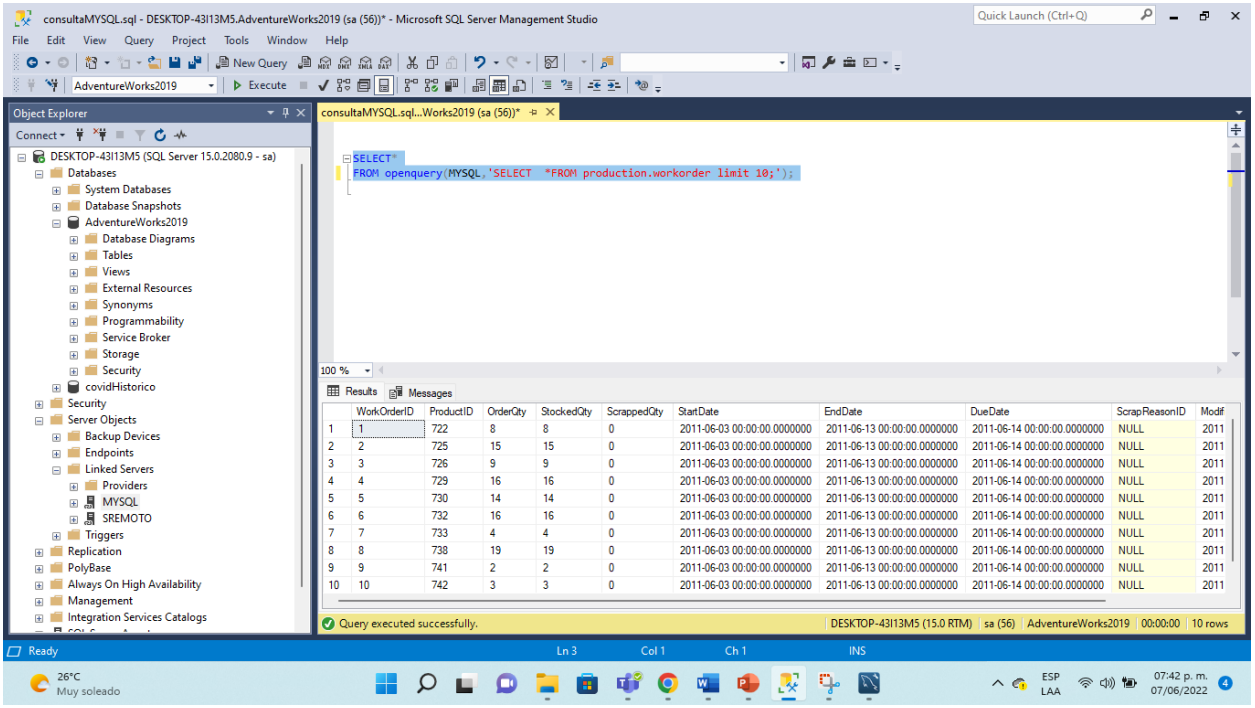


El tipo de proveedor seleccionamos el de OLE DB for OBDC, en product name colocamos el nombre del servicio y en Data source colocamos el ODBC que creamos, en este caso el nombre es MYSQL.

Una vez creado el servidor remoto, realizamos una consulta para poder la validar que todo salió bien.

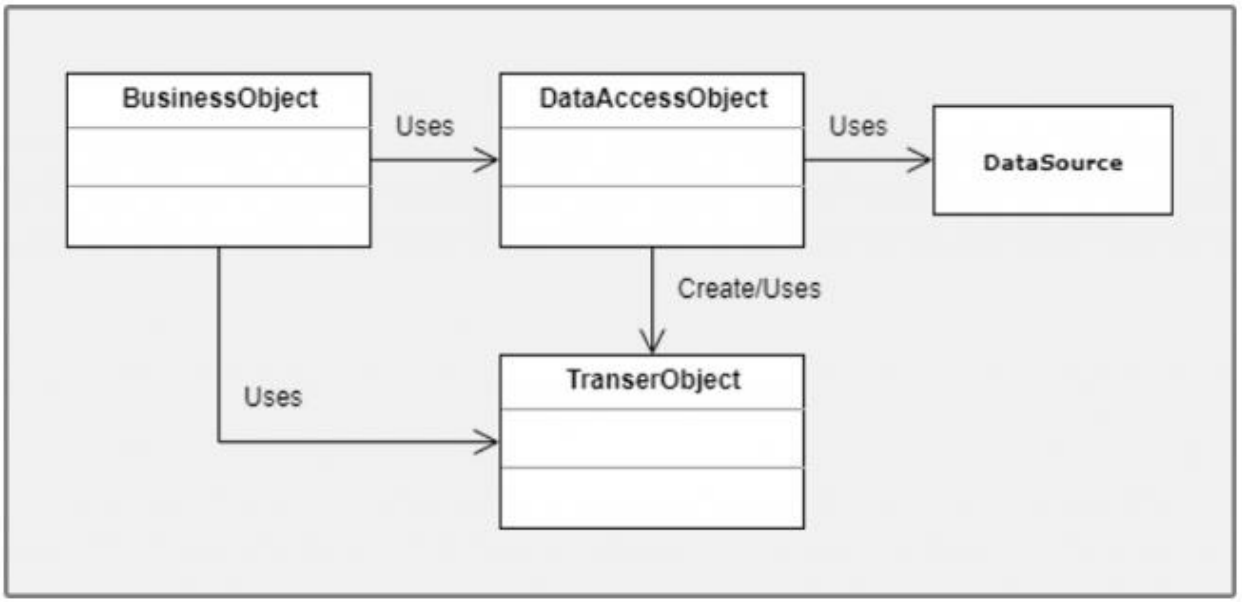
```
SELECT*
FROM openquery(MYSQL,'SELECT  *FROM production.workorder limit 10;');
```

Una vez ejecuta la consulta observamos que si nos mostró la información solicitada.



Implementación del patrón de diseño DAO.

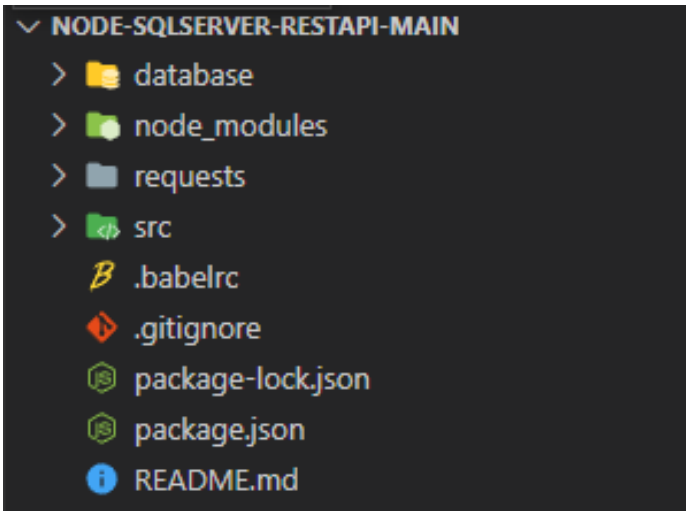
El patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.



Nosotros para implementar el DAO utilizamos JavaScript con el cual realizaremos una API la cual nos permitirá realizar diferentes consultas y poder darle un poco mas de orden al código.

Utilizaremos NodeJS, con el levantaremos la conexión y crearemos la API.

Creamos nuestra API.



Realizamos la conexión con la base de datos.

connection.js

src > database > connection.js > ...

```
1   import sql from "mssql";
2   import config from "../config";
3
4   export const dbSettings = {
5     user: config.dbUser,
6     password: config.dbPassword,
7     server: config.dbServer,
8     database: config.dbDatabase,
9     options: {
10      encrypt: true, // for azure
11      trustServerCertificate: true, // change to true for local dev / self-signed certs
12    },
13  };
14
15  export const getConnection = async () => {
16    try {
17      const pool = await sql.connect(dbSettings);
18      return pool;
19    } catch (error) {
20      console.error(error);
21    }
22  };
23
24  export { sql };
25
```

Creamos la parte de los Querys la cual estará relaciona con las rutas para poder llamar la función:

querys.js

src > database > querys.js > ...

```
export const querys = {
  getAllProducts: "SELECT*FROM openquery(MYSQL,'SELECT  *FROM production.workorder limit 10;')",
  getProducById: "SELECT * FROM Products Where Id = @Id",
  addNewProduct:
    "INSERT INTO [webstore].[dbo].[Products] (name, description, quantity) VALUES (@name,@description,
  deleteProduct: "DELETE FROM [webstore].[dbo].[Products] WHERE Id= @Id",
  getTotalProducts: "SELECT COUNT(*) FROM webstore.dbo.Products",
  updateProductById:
    "UPDATE [webstore].[dbo].[Products] SET Name = @name, Description = @description, Quantity = @quan
};

```

Definimos las Rutas:

querys.js | products.routes.js

src > routes > products.routes.js > ...

```
import { Router } from "express";
import {
  getProducts,
  createNewProduct,
  getProductById,
  deleteProductById,
  getTotalProducts,
  updateProductById,
} from "../controllers/products.controller";

const router = Router();

router.get("/products", getProducts);

router.post("/products", createNewProduct);

router.get("/products/count", getTotalProducts);

router.get("/products/:id", getProductById);
```

Aquí implementamos el patrón de diseño DAO el cual nos permite realizar diferentes acciones en la base de datos. El patrón DAO es sin lugar a duda, unos de los más utilizados en la actualidad, ya que es fácil de implementar y proporciona claros beneficios, incluso, si solo tenemos una fuente de datos y esta no cambia, pues permite separar por completo la lógica de acceso a datos en una capa separada y así solo nos preocupamos por la lógica de negocio sin preocuparnos de donde viene los datos o los detalles técnicos para consultarlos o actualizarlos.

querys.js | products.routes.js | products.controller.js

src > controllers > products.controller.js > getProducts

```
1   import { getConnection, querys, sql } from "../database";
2
3   export const getProducts = async (req, res) => {
4     try {
5       const pool = await getConnection();
6       const result = await pool.request().query(querys.getAllProducts);
7       res.json(result.recordset);
8     } catch (error) {
9       res.status(500);
10      res.send(error.message);
11    }
12  };
13
14  export const createNewProduct = async (req, res) => {
15    const { name, description } = req.body;
16    let { quantity } = req.body;
17
18    // validating
19    if (description == null || name == null) {
20      return res.status(400).json({ msg: "Bad Request. Please fill all fields" });
21    }
22
23    if (quantity == null) quantity = 0;
24
25    try {
26      const pool = await getConnection();
27
```

Implementa las operaciones por medio de la aplicación.