# UF4. MODELO DE DATOS ODMG
## (OBJECT DATABASE MANAGEMENT GROUP)

2° DAM

ACCESO A DATOS

# ESTANDAR ODMG

- **ODMG** fue una **iniciativa** creada por un consorcio de empresas y expertos para **establecer un estándar común para las bases de datos orientadas a objetos**.
- Su objetivo era proporcionar una forma consistente **de trabajar con datos** en sistemas que seguían el **paradigma de programación orientado a objetos.**
- Su objetivo era **resolver la falta de uniformidad** entre las distintas bases de datos orientadas a objetos y **facilitar la interoperabilidad** entre ellas y los **lenguajes de programación.**

# COMPONENTES PRINCIPALES DE ODMG

- **ODL (Object Definition Language):** Similar al SQL para bases de datos relacionales, pero diseñado para definir clases, atributos, métodos y relaciones entre objetos.
- **OQL (Object Query Language):** Un lenguaje de consulta orientado a objetos basado en SQL, adaptado para trabajar con colecciones y relaciones complejas entre objetos.
- **Bindings en lenguajes de programación:** Proporcionaba estándares para integrar las bases de datos con lenguajes de programación como C++, Smalltalk y Java.

# POR QUÉ NO TRIUNFÓ… DEL TODO

Aunque el estándar ODMG tenía un enfoque innovador, no logró convertirse en la norma dominante

**Ascenso de las bases de datos relacionales (RDBMS)**

- En los **años noventa**, las bases de datos relacionales **ya dominaban el mercado** gracias a su solidez, rendimiento y soporte de empresas **como Oracle, IBM y Microsoft.**
- Los **sistemas relacionales** comenzaron a **incluir soporte para objetos** (como tipos definidos por el usuario en **PostgreSQL**), lo que disminuyó la necesidad de bases de datos puramente orientadas a objetos.

## Complejidad de las OODB

- Las bases de datos orientadas a objetos eran más complejas de implementar y mantener.
- El **modelo relacional**, con SQL, ya era **bien comprendido y aceptado** por la industria.

## Falta de adopción por parte de grandes empresas

- El estándar ODMG **no tuvo el respaldo de grandes corporaciones** que lideraban el mercado.
- Las OODB **nunca alcanzaron la popularidad** de los sistemas relacionales.

## Competencia con nuevas tecnologías

- Con el tiempo, surgieron **nuevas soluciones como ORM (Object-Relational Mapping)** en lenguajes como Java y Python (por ejemplo, **Hibernate**), que integraron bases de datos relacionales con programación orientada a objetos sin necesidad de adoptar OODB.

## Final del estándar

- **ODMG** cesó su actividad **en el año 2001**.
- Esto marcó el **fin** del intento formal de **estandarizar las bases de datos orientadas a objetos**

# IMPACTO DE ODMG EN LAS BASES DE DATOS ACTUALES… ALGO QUEDÓ

**Características orientadas a objetos en RDBMS**

- Bases de datos relacionales como **PostgreSQL, Oracle y DB2** han adoptado características orientadas a objetos. Por ejemplo:
  - Tipos de datos definidos por el usuario.
  - Herencia de tablas (como en PostgreSQL).
  - Consultas que manejan estructuras complejas.

**ORM y lenguajes de consulta avanzados → Hibernate**

- El concepto de integrar datos y objetos en lenguajes de programación influyó en la creación de frameworks como **Hibernate (Java),** SQLAlchemy (Python) y ActiveRecord (Ruby on Rails).
- Estos frameworks utilizan ideas similares a **OQL para hacer consultas → HQL**

**Bases de datos no relacionales, NoSQL → MongoDB**

- Las ideas detrás de ODMG resurgieron en el **diseño de bases de datos NoSQL**, especialmente en **bases de datos orientadas a documentos** como **MongoDB**, que almacenan datos en estructuras similares a objetos (JSON).

**Impacto en lenguajes de consulta → JPA**

- Lenguajes como **JPQL (Java Persistence Query Language)** y LINQ (Language Integrated Query en .NET) adoptaron ideas de OQL.

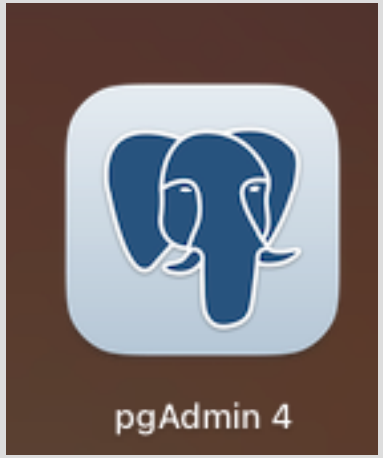
PostgreSQL

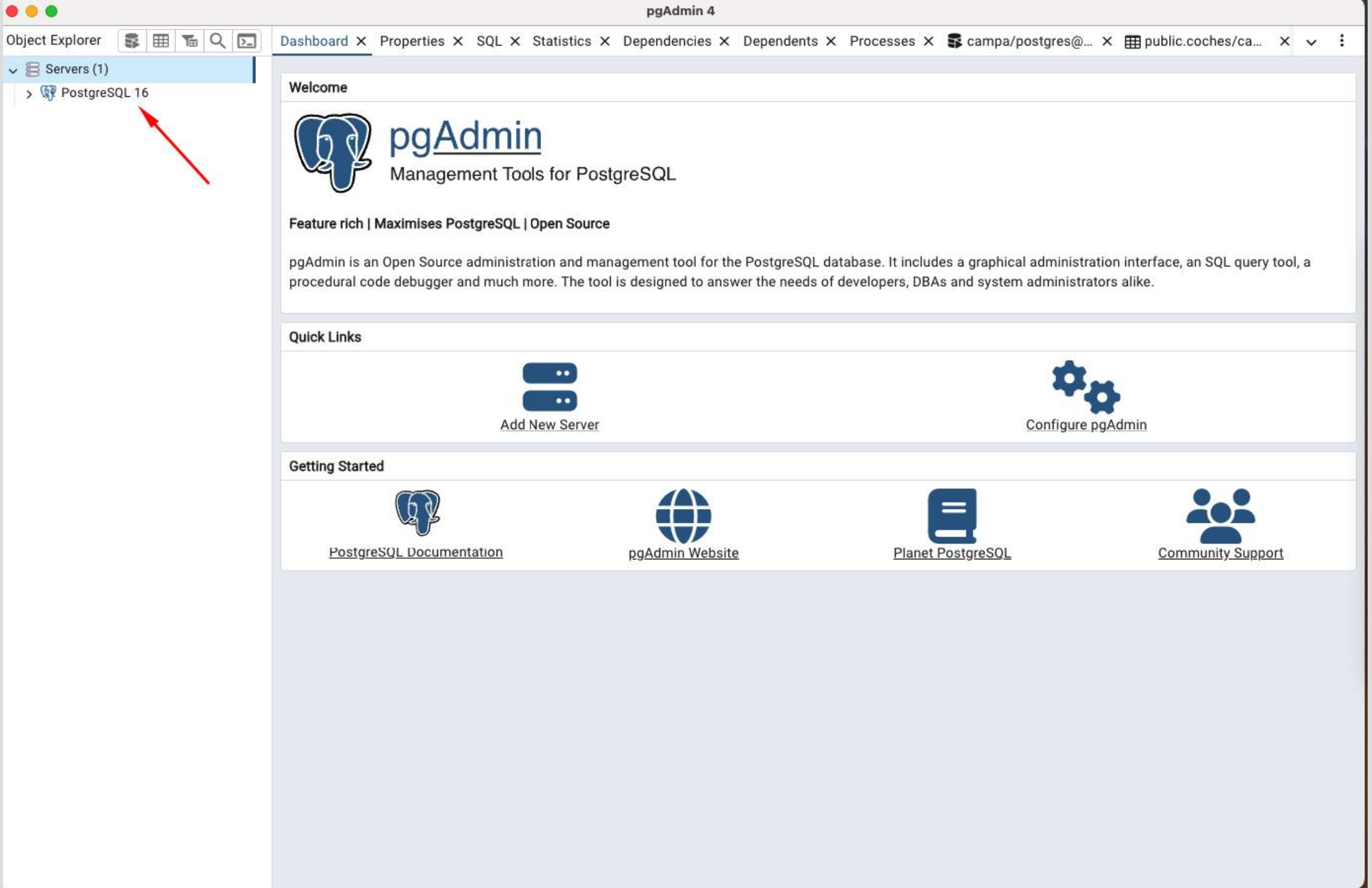
EDB EDB Postgres AI Services Resources Company Sign In Contact Us

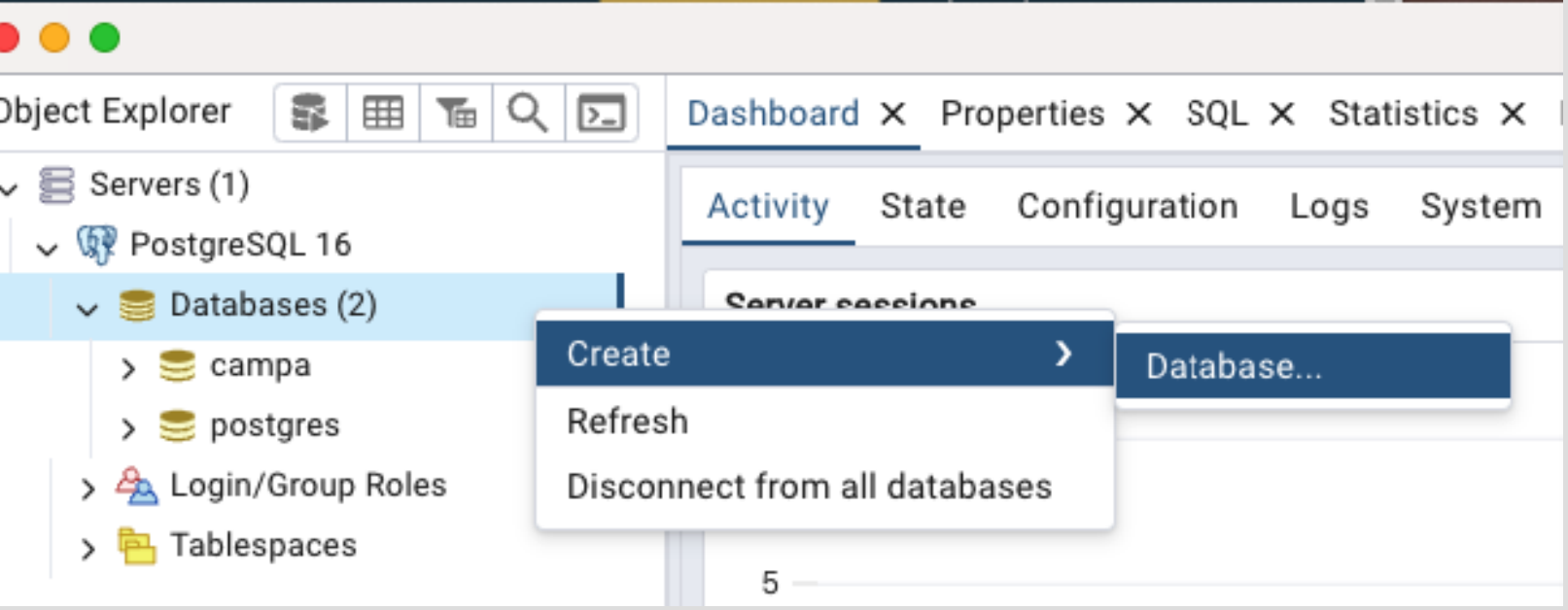
# Download PostgreSQL

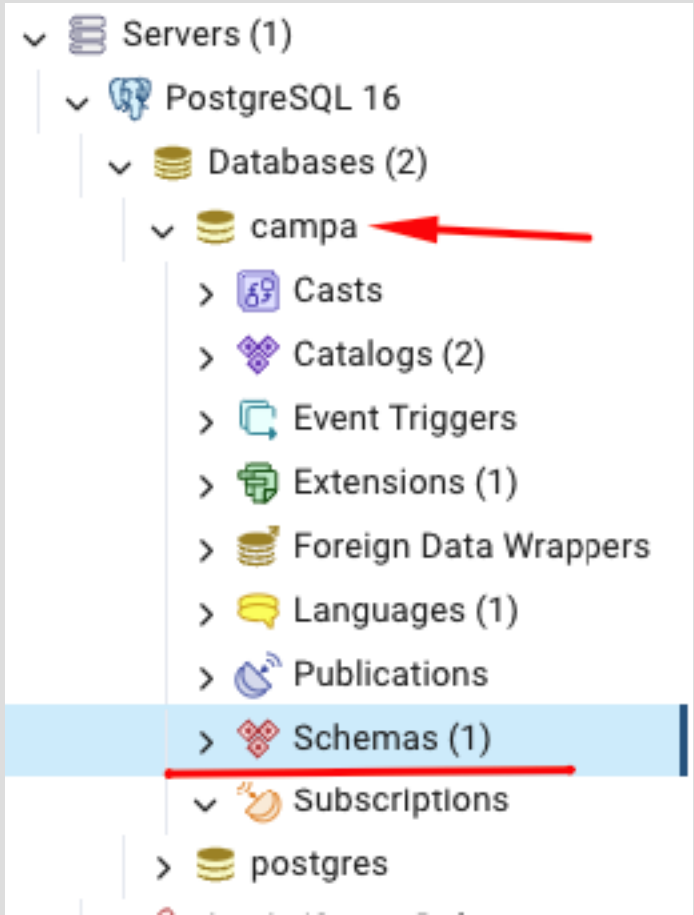
Open source PostgreSQL packages and installers from EDB

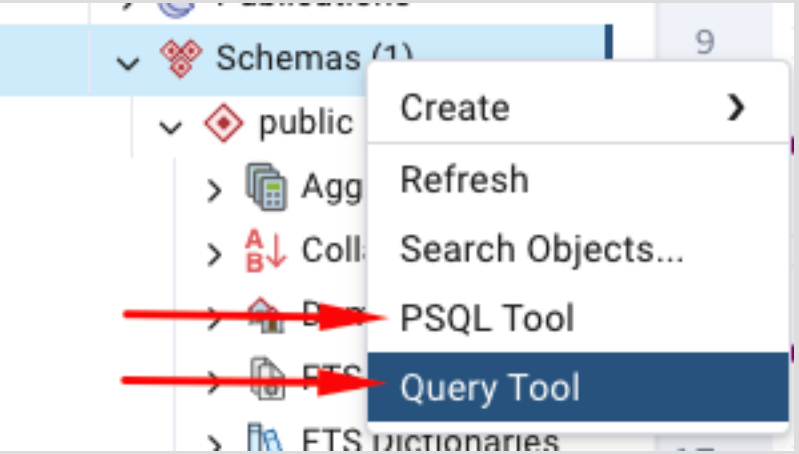
| PostgreSQL Version | Linux x86-64 | Linux x86-32 | Mac OS X | Windows x86-64 | Windows x86-32 |
|---|---|---|---|---|---|
| 17.2 | postgresql.org | postgresql.org | | | Not supported |
| 16.6 | postgresql.org | postgresql.org | | | Not supported |
| 15.10 | postgresql.org | postgresql.org | | | Not supported |




pgAdmin 4
Object Explorer
Servers (1)
PostgreSQL 16
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Processes
campa/postgres@...
public.coches/ca...
Welcome
pgAdmin
Management Tools for PostgreSQL
Feature rich | Maximises PostgreSQL | Open Source
pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.
Quick Links
Add New Server
Configure pgAdmin
Getting Started
PostgreSQL Documentation
pgAdmin Website
Planet PostgreSQL
Community Support


Object Explorer
Servers (1)
PostgreSQL 16
Databases (2)
campa
postgres
Login/Group Roles
Tablespaces
Dashboard
Properties
SQL
Statistics
Activity
State
Configuration
Logs
System
Create
Refresh
Disconnect from all databases
Database...
5


Servers (1)
PostgreSQL 16
Databases (2)
campa
Casts
Catalogs (2)
Event Triggers
Extensions (1)
Foreign Data Wrappers
Languages (1)
Publications
Schemas (1)
Subscriptions
postgres

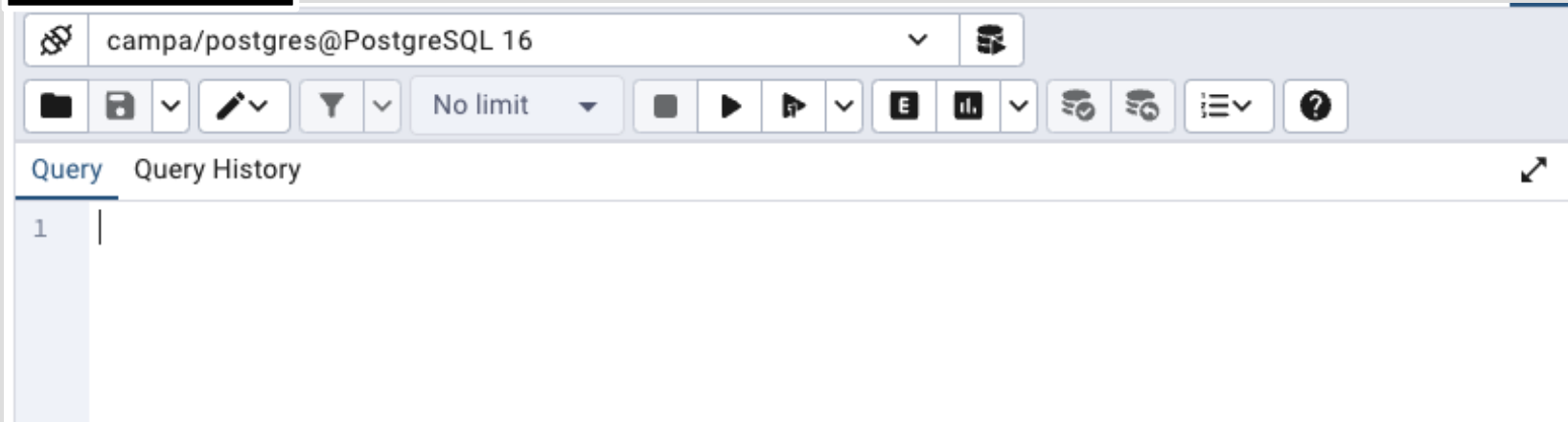



## PSQL Tool

```
psql (16.6)
Type "help" for help.

campa=#
```

## Query Tool



```
CREATE TYPE vehiculo AS (
  marca TEXT,
  modelo TEXT
);
```


```
CREATE TABLE vehiculos (
  id SERIAL PRIMARY KEY,
  info vehiculo
);
```

```
CREATE TABLE coches (
  puertas INT
) INHERITS (vehiculos);
```

```
CREATE TABLE motos (
  tipo_manillar TEXT
) INHERITS (vehiculos);
```

```sql
INSERT INTO coches (info, puertas)
VALUES (ROW('Toyota', 'Corolla'), 4);
```

Query | Query History

```sql
SELECT * FROM coches
```

Data Output | Messages | Notifications

| | id integer | info vehiculo | puertas integer |
|---|---|---|---|
| 1 | 1 | (Toyota,Corolla) | 4 |

```sql
INSERT INTO motos (info, tipo_manillar)
VALUES (ROW('Yamaha', 'MT-07'), 'Deportivo');
```

Query | Query History

```sql
SELECT * FROM motos
```

| | id integer | info vehiculo | tipo_manillar text |
|---|---|---|---|
| 1 | 2 | (Yamaha,MT-07) | Deportivo |

```sql
SELECT * FROM vehiculos
```

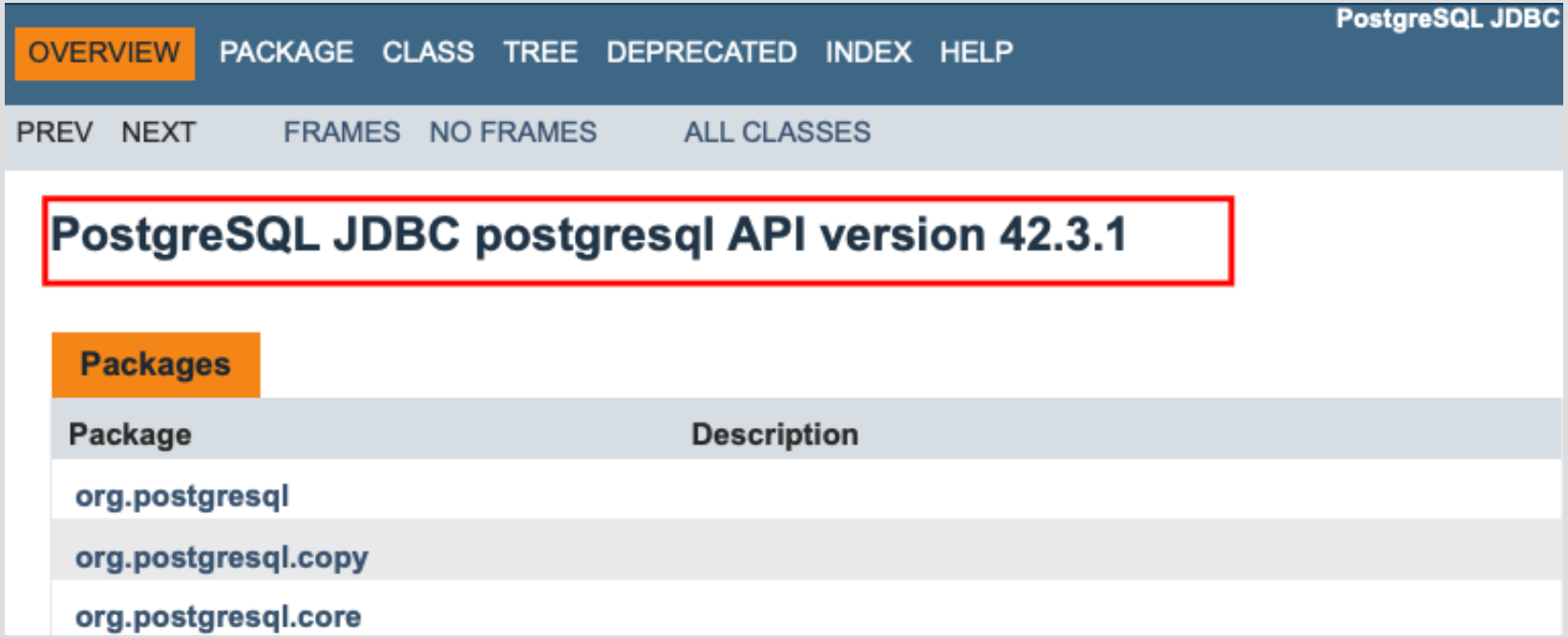
Data Output | Messages | Notifications

| | id [PK] integer | info vehiculo |
|---|---|---|
| 1 | 1 | (Toyota,Corolla) |
| 2 | 2 | (Yamaha,MT-07) |

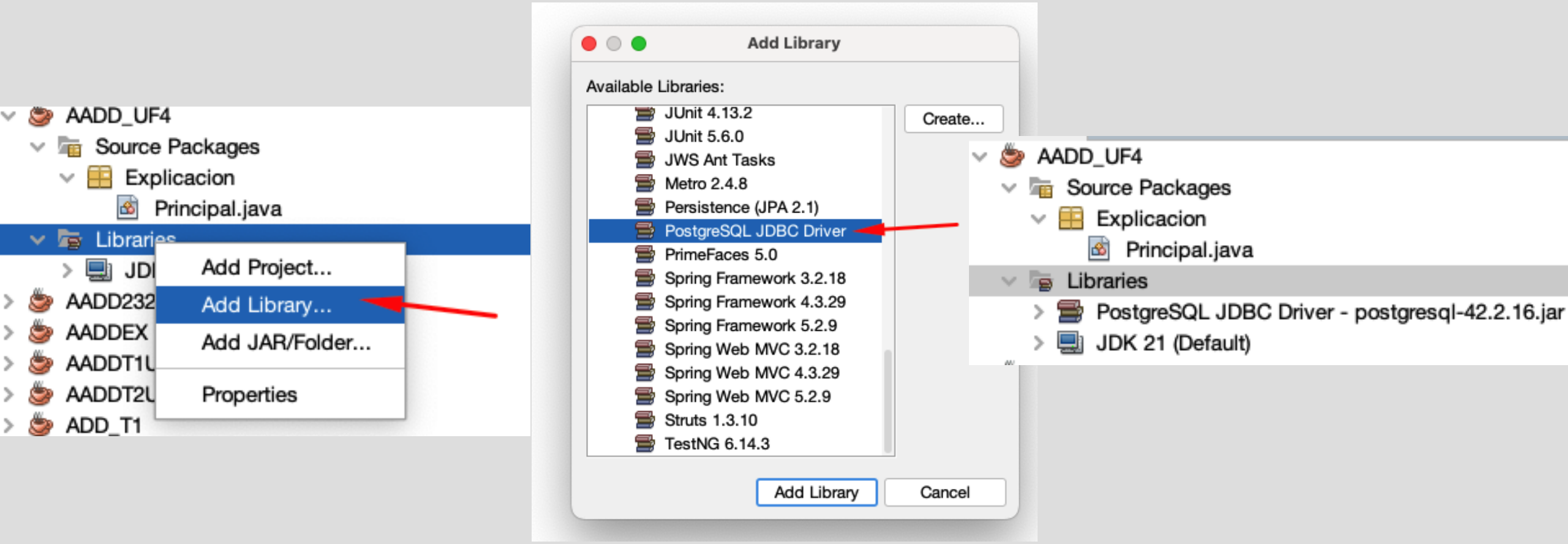

JDBC
Java

# API DE POSTGRESQL (JDBC)

- https://jdbc.postgresql.org/documentation/publicapi/overview-summary.html



# CONEXIÓN POR JDBC



# PGOBJECT… PURO

```
public static void main(String[] args) {
    // Configurar la conexión
    String url = "jdbc:postgresql://localhost:5432/campa";//nombre de la BBDD
    String usuario = "postgres"; //usuario por defecto
    String pass = "root";
    //try-with-resources
    try (Connection conn= DriverManager.getConnection(url, user:usuario, password:pass);
         Statement stmt = conn.createStatement();) {
        // Consulta para leer objetos con PGobject
        String sql = "SELECT id, info FROM vehiculos";
        ResultSet rs = stmt.executeQuery( string:sql);
        // Procesar los resultados y el objeto lo gestionamos como un String
        while (rs.next()) {
            int id = rs.getInt( string:"id");
            PGobject obj = (PGobject) rs.getObject( string:"info");
            System.out.println("ID: " + id);
            System.out.println("Información del Vehículo: " + obj.getValue());
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

| | id [PK] integer | info vehiculo |
|---|---|---|
| 1 | 1 | (Toyota,Corolla) |
| 2 | 2 | (Yamaha,MT-07) |

### getValue

```
public @Nullable String getValue()
```

This must be overidden, to return the value of the object, in the form required by org.postgresql.

**Returns:**

the value of this object

### setValue

```
public void setValue(@Nullable String value)
              throws SQLException
```

This method sets the value of this object. It must be overridden.

**Parameters:**

value - a string representation of the value of the object

**Throws:**

SQLException - thrown if value is invalid for this type

Creamos la clase Vehiculos que es la que queremos trabajar en el programa principal, con la misma estructura con la que se ha creado en Postgresql

```
public class Vehiculos {

    private String marca;
    private String modelo;


    public Vehiculos(String marca, String modelo) {...4 lines }
    public String getMarca() {...3 lines }
    public void setMarca(String marca) {...3 lines }
    public String getModelo() {...3 lines }
    public void setModelo(String modelo) {...3 lines }

    @Override
    public String toString() {...6 lines }
}
```

Sobrescribimos la clase PGOBject y adaptamos el String que nos da al objeto que queremos

```java
import org.postgresql.util.PGobject;
import java.sql.SQLException;

//Vamos a sobreescribir la clase PGObject y lo adaptamos a Vehiculos
//El string los separamos y construimos un objeto con cada una de sus partes

public class PGOVehiculos extends PGobject{
    private Vehiculos vehiculo;

    public PGOVehiculos() {...3 lines }

    @Override
    public void setValue(String value) throws SQLException {...10 lines }

    public Vehiculos getVehiculo() {...3 lines }
}
```

```java
public PGOVehiculos() {
    setType( type:"vehiculo"); // Define el tipo PostgreSQL
}
```

```sql
CREATE TYPE vehiculo AS (
  marca TEXT,
  modelo TEXT
);
```

```java
@Override
public void setValue(String value) throws SQLException {
    super.setValue(value);

    // Parsear el valor desde PostgreSQL (formato típico: "(marca,modelo)")
    if (value != null) {
        String[] parte = value.replace( target:"(",  replacement:"")
                .replace( target:")",  replacement:"").split( regex:",");
        //Construimos el vehículo con las dos partes separadas
        this.vehiculo = new Vehiculos(parte[0], parte[1]);
    }
}
```

```java
public Vehiculos getVehiculo() {
    return vehiculo;
}
```

Modificamos el recorrido recogiendo el PGObject y metiéndolo en nuestro PGOVehiculos.

Ya podremos hacer el setter para poner el objeto Vehiculos y el getter para trabajar con él

```
while (rs.next()) {
    PGobject pgObject = (PGobject) rs.getObject( string:"info");
    PGOVehiculos pgVehiculo = new PGOVehiculos();
    //Lanzamos el setter
    //Se podría construir el objetos directamente ahí modificando el constructor
    pgVehiculo.setValue( value:pgObject.getValue());
    //Lo recogemos un getter
    Vehiculos vehiculo = pgVehiculo.getVehiculo();
    //Ya podemos trabajar con él
    System.out.println( x:vehiculo.toString());
}
```