


PROYECTO FINAL

BASE DE DATOS

González Flores Diana Patricia

170812



Contenido

Introducción.....	2
Marco Teórico	3
Planteamiento	4
Requerimientos funcionales	4
Requerimientos no funcionales	4
Diagrama PgModeler	5
Implementación en PostgreSQL.....	5
Tablas.....	6
Relaciones	10
Funciones	10
Triggers.....	14
Vistas	18
Ingreso y visualización de datos utilizando las funciones, triggers y vistas implementadas	21
Agregar instructores	21
Vista instructores	22
Agregar alumnos	22
Vista alumnos	22
Agregar clases.....	23
Validación de horario de la clase con trigger	24
Vista clases.....	24
Inscribir alumno a clase	24
Validación de inscripción con trigger	25
Vista inscripciones	25
Conclusiones.....	26
Bibliografía	27

Introducción

Una base de datos es un sistema estructurado donde se pueden colocar datos imponiendo reglas y estructura, que puede cambiar de acuerdo a las necesidades que se tengan.

Al plantear un sistema de cualquier índole puede surgir la siguiente pregunta ¿Por qué necesitamos una base de datos?, esta interrogante es muy fácil de responder ya que cuando se tengan algunos datos, será de suma importancia almacenarlos en algún lugar. Estos datos pueden ser cualquier cosa, podría tratarse de clientes, productos, alumnos, pedidos, etc. Estos datos pueden estar en formato de texto, numérico, fechas, etc. Además será importante poder administrar estos datos, ya que se deseará poder agregar, editar, eliminar y consultar. Una base de datos nos permite realizar todo lo anterior de una forma segura, optima y eficiente.

Implementar una base de datos siempre tiene muchos beneficios, por ejemplo imaginemos que se tienen los datos de los empleados de una empresa, tal vez se pueda pensar que no sea necesario utilizar una base de datos y que estos datos se pueden almacenar, por ejemplo, en una hoja de cálculo de Excel. Aunque parezca que esta manera de almacenar datos es muy rápida y eficiente, ¿Qué pasaría si...

- hay un número muy grande de empleados?
- solo ciertas personas deben acceder a esta información?
- es importante validar la información ingresada?
- no debe haber información duplicada?
- varios usuarios modifican información al mismo tiempo?
- falla el programa?
- se debe acceder a los datos de forma rápida?

Al hacer este análisis, es fácil apreciar que una hoja de cálculo no es la forma más apropiada para guardar estos datos, ya que Excel o una aplicación similar no nos permitiría evitar o solucionar los errores que puedan surgir cuando pasen las situaciones planteadas anteriormente. En cambio una base de datos, por sus bases, reglas y conceptos, nos permite solucionar estos problemas e incluso más.

Debido a lo anterior, es de suma importancia conocer los conceptos de base de datos que nos permitan diseñar e implementar una base de datos que cumpla con los requerimientos y reglas necesarios.

El presente proyecto muestra el proceso de desarrollo e implementación de una base de datos para un sistema administrativo, donde se diseñó una base de datos acorde al levantamiento de requerimientos. Esta base de datos fue implementada en PostgreSQL, siguiendo los pasos necesarios y tomando como referencia toda la teoría y práctica aprendida en la materia Base de Datos de la Universidad Politécnica de San Luis Potosí.

Marco Teórico

Atributos : Las propiedades que posee cada miembro de un conjunto de entidades.

Base de datos: Una base de datos (BD) es una entidad en la cual se pueden almacenar datos de manera estructurada, con la menor redundancia posible

Comandos SQL: Especifican qué tipo de interacción se hará con la base de datos.

CREATE TABLE: Crea una nueva tabla

ALTER TABLE: Modifica una tabla

SELECT: Obtener datos de tablas de bases de datos

UPDATE: Modificar datos en una tabla de base de datos

INSERT INTO: Inserta nuevos datos en una tabla de base de datos

Entidad: Objeto en el mundo real que es distinguible de todos los demás objetos.

Función: Serie de sentencias SQL que se tratan como una unidad de un bloque funcional.

Integridad Referencial: Son reglas que se utilizan en la mayoría de las bases de datos, que permiten asegurar que los registros en tablas relacionadas sean válidos, no se borren o modifiquen datos que puedan producir errores de integridad. Se realiza a través de una clave foránea, ya que a partir de ahí se comprueba que las modificaciones que se lleguen a hacer no lleven a errores de integridad.

JOIN: Comando utilizado para consultar datos de más de una tabla.

Llave primaria: Campo o campos que se escogen para identificar registros de modo único.

Modelo Entidad-Relación: Modelo de datos de alto nivel, basado en una percepción de un mundo real que consiste en una colección de objetos básicos, denominados entidades, y de relaciones entre estos objetos.

Relación: Asociación entre diferentes entidades.

Secuencia: Elemento que genera valores secuenciales que se pueden utilizar en cualquier sentencia SQL.

Sistema Gestor de Bases de Datos (SGBD): Colección de archivos interrelacionados y un conjunto de programas que permitan a los usuarios acceder y modificar estos archivos

Trigger: Acción definida en una tabla de una base de datos que es ejecutada automáticamente por una función programada por nosotros. Esta acción se activa al realizar un INSERT, UPDATE o DELETE.

Vista: Tabla virtual con una estructura definida sin datos, se utiliza para guardar consultas de selección en la base de datos.

Planteamiento

Sistema para el control de una escuela de Danza.

Se plantea un sistema que cumpla con los requerimientos de una escuela de danza. Dicha escuela brinda clases en un horario de 8:00 a 20:00 hrs. Cada clase dura un hora y cada hora se inicia clase (Por ejemplo de 12:00 a 13:00, 15:00 a 16:00 ,19:00 a 20:00). Al ser una escuela pequeña solo cuenta con un salón, por lo que solo se puede dar una clase a la vez. El sistema debe llevar un control de los instructores, las clases y los alumnos, permitiendo registrar y visualizar esta información. A continuación se describen los requisitos funcionales y no funcionales del sistema.

Requerimientos funcionales

Requerimiento	Descripción
RF-01	Se requiere poder registrar Instructores, incluyendo su nombre y fecha de nacimiento, además de asignarle una clave única para su identificación
RF-02	Se requiere poder registrar clases, incluyendo la hora de la clase, el instructor que la impartirá y los lugares disponibles para los alumnos.
RF-03	Se requiere poder registrar alumnos, incluyendo su nombre y fecha de nacimiento, además se debe asignar una clave única para su identificación.
RF-04	Se requiere poder inscribir alumnos a las clases, verificando los lugares disponibles, ya que si estos se agotaron ya no se pueden inscribir más alumnos a la clase.
RF-05	Visualizar reporte de los instructores registrados.
RF-06	Visualizar reporte de los alumnos registrados.
RF-07	Visualizar reporte de las clases registradas.
RF-08	Visualizar reporte de los alumnos que se han inscrito a las clases, mostrando los datos del alumno y de la clase. En este reporte se podrá ver la información de todos los alumnos o de alguno en particular.

Requerimientos no funcionales

Requerimiento	Descripción
RNF-01	Base de Datos PostgreSQL 10 o superior
RNF-02	Los datos de la base de datos deben tener integridad y seguridad
RNF-03	Los nombres de las columnas deben tener un nombre significativo, para mayor comprensión

RNF-04	A través de mensajes, se debe informar al usuario si ocurrió un error (ej. No hay lugares disponibles para inscripción, hora de clase fuera de rango, etc.)
RNF-05	Reporte instructores
RNF-06	Reporte Alumnos
RNF-07	Reporte clases
RNF-08	Reporte inscripciones

Diagrama PgModeler

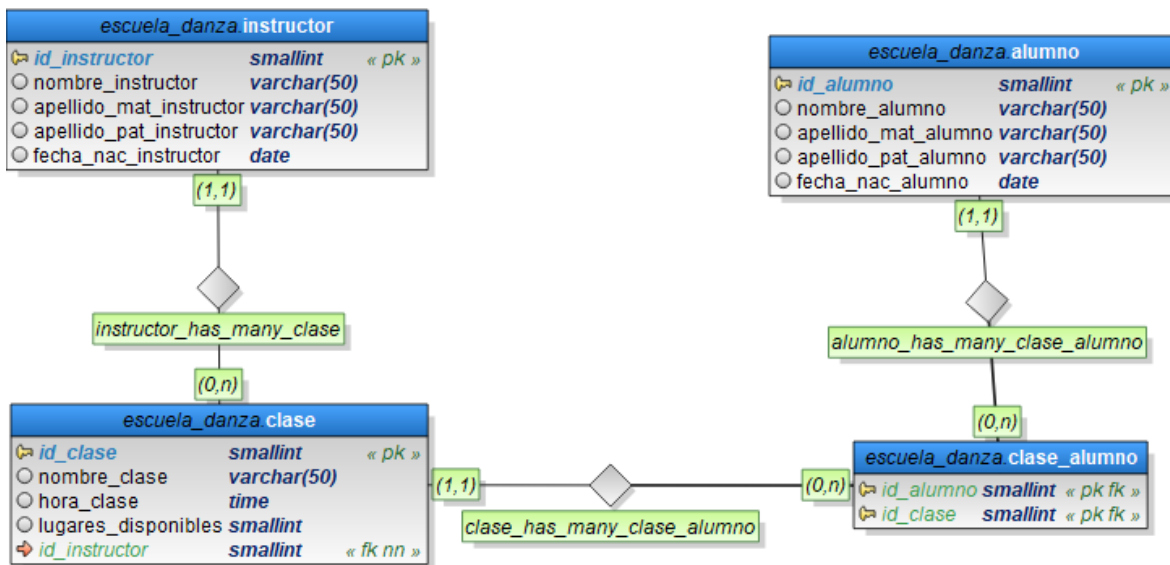


Ilustración 1 Diagrama en PgModeler

Implementación en PostgreSQL

1. Crear la base de datos BD_EscuelaDanza

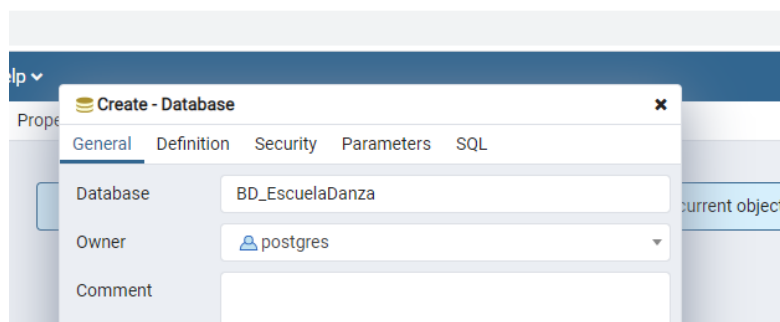


Ilustración 2 BD_EscuelaDanza

2. Crear el esquema escuela_danza

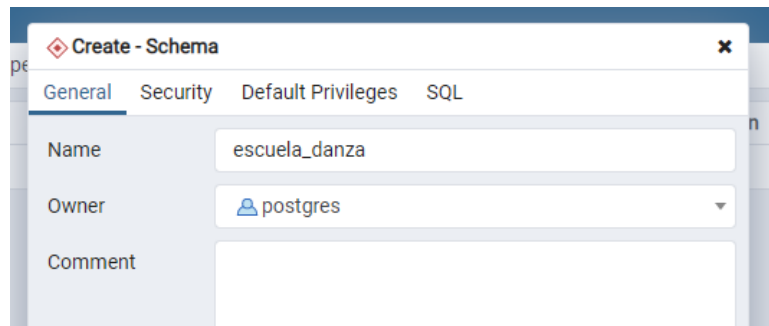


Ilustración 3 Esquema escuela_danza

Tablas

Tabla instructor

```
CREATE SEQUENCE escuela_danza.escuela_danza_id_instructor_seq;  
CREATE TABLE escuela_danza.instructor(  
    id_instructor smallint NOT NULL DEFAULT  
        nextval('escuela_danza.escuela_danza_id_instructor_seq'::regclass),  
    nombre_instructor varchar(50),  
    apellido_mat_instructor varchar(50),  
    apellido_pat_instructor varchar(50),  
    fecha_nac_instructor date,  
    CONSTRAINT pk_id_instructor PRIMARY KEY (id_instructor)  
    WITH (FILLFACTOR = 10)  
    USING INDEX TABLESPACE pg_default  
)  
TABLESPACE pg_default;  
ALTER TABLE escuela_danza.instructor OWNER TO postgres;
```

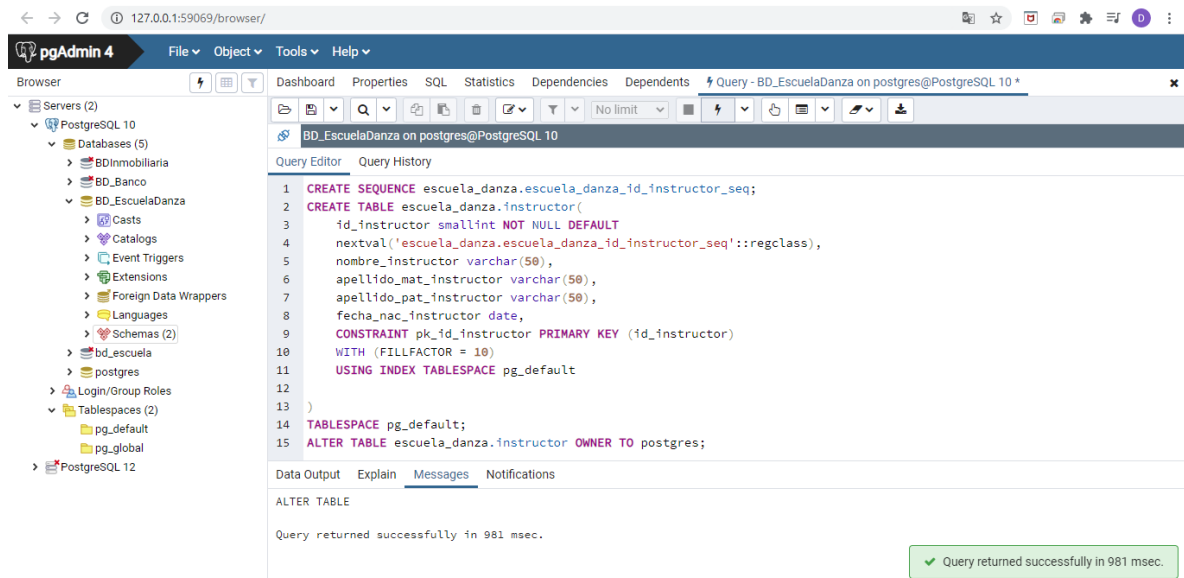


Ilustración 4 Tabla instructor

Tabla alumno

```
CREATE SEQUENCE escuela_danza.escuela_danza_id_alumno_seq;
CREATE TABLE escuela_danza.alumno(
    id_alumno smallint NOT NULL DEFAULT
    nextval('escuela_danza.escuela_danza_id_alumno_seq'::regclass),
    nombre_alumno varchar(50),
    apellido_mat_alumno varchar(50),
    apellido_pat_alumno varchar(50),
    fecha_nac_alumno date,
    CONSTRAINT pk_id_alumno PRIMARY KEY (id_alumno)
)
TABLESPACE pg_default;
ALTER TABLE escuela_danza.alumno OWNER TO postgres;
```

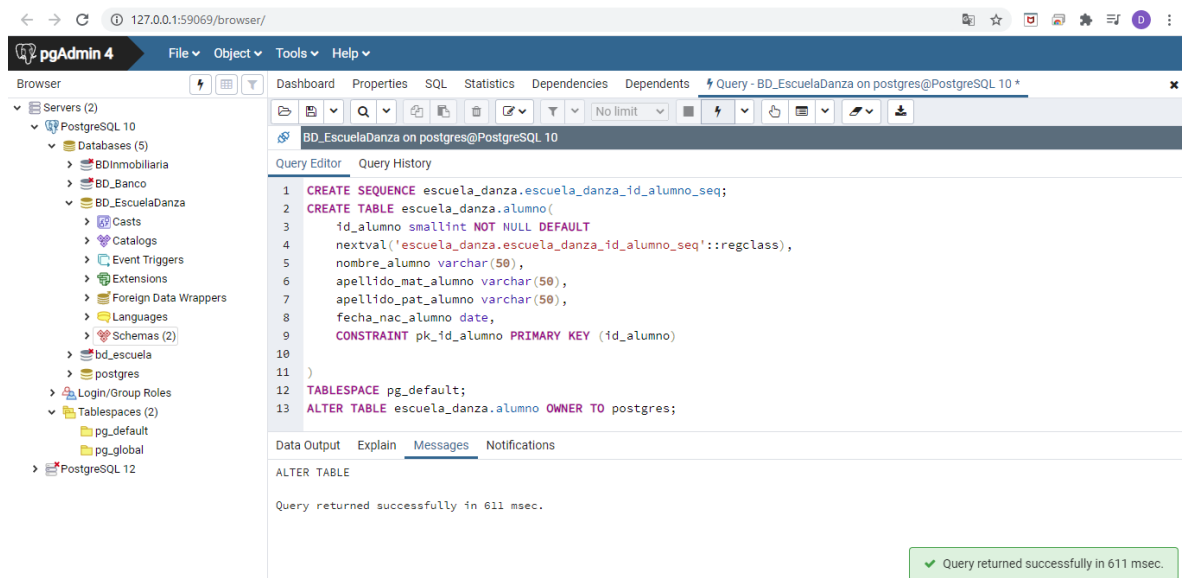



Ilustración 5 Tabla alumno

Tabla clase

```

CREATE SEQUENCE escuela_danza.escuela_danza_id_clase_seq;
CREATE TABLE escuela_danza.clase(
    id_clase smallint NOT NULL DEFAULT
    nextval('escuela_danza.escuela_danza_id_clase_seq'::regclass),
    nombre_clase varchar(50),
    hora_clase time,
    lugares_disponibles smallint,
    id_instructor smallint NOT NULL,
    CONSTRAINT pk_id_clase PRIMARY KEY (id_clase)
    WITH (FILLFACTOR = 10)
    USING INDEX TABLESPACE pg_default
)
TABLESPACE pg_default;
ALTER TABLE escuela_danza.clase OWNER TO postgres;

```

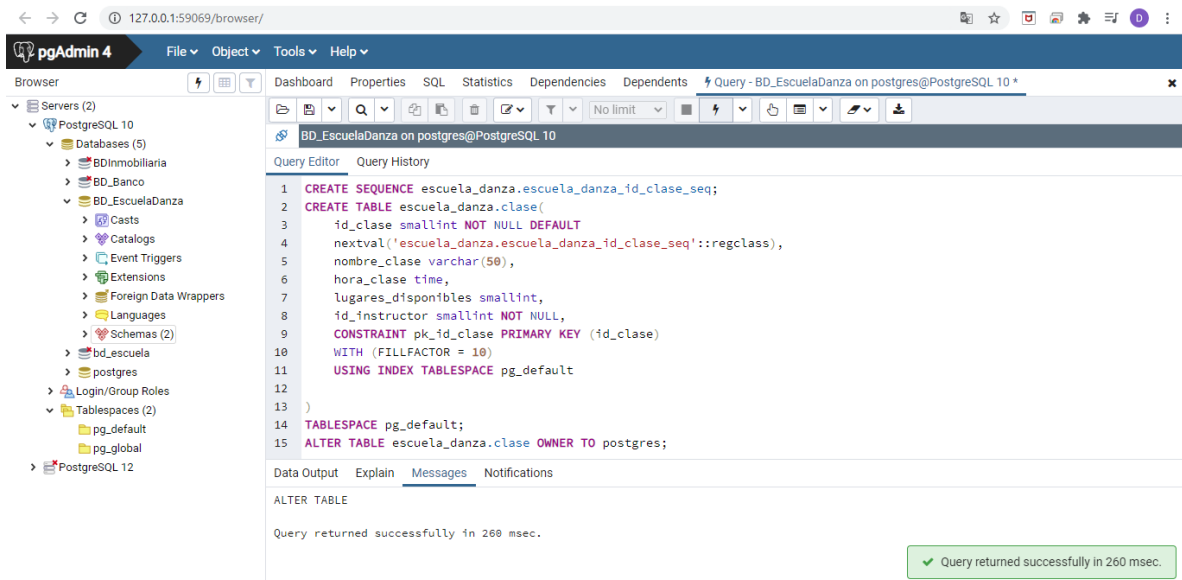


Ilustración 6 Tabla clase

Tabla clase_alumno

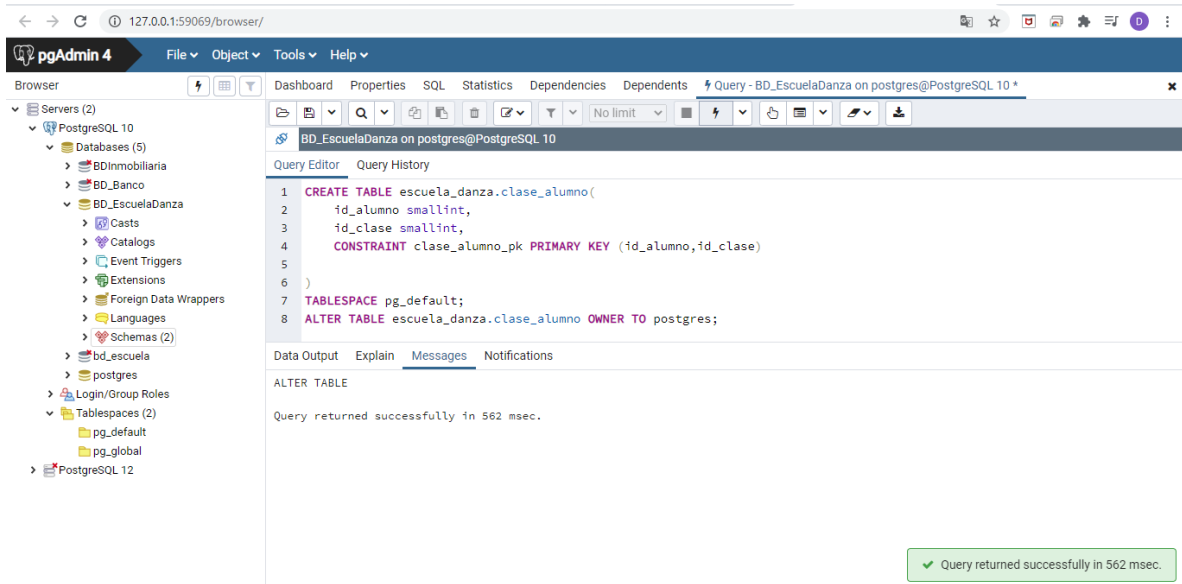
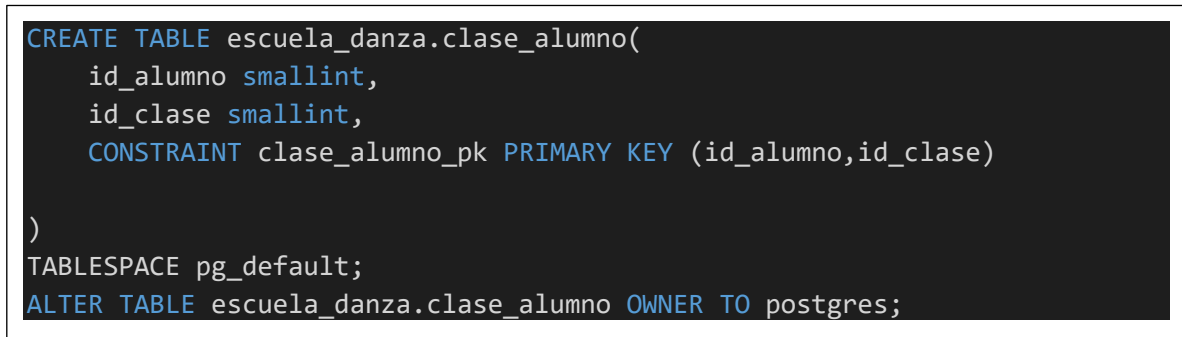


Ilustración 7 Tabla clase_alumno

Relaciones

```
--relacion maestro clase-
ALTER TABLE escuela_danza.clase ADD CONSTRAINT instructor FOREIGN KEY (id_instructor)
REFERENCES escuela_danza.instructor (id_instructor) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

--relacion clase clase_alumno--
ALTER TABLE escuela_danza.clase_alumno ADD CONSTRAINT clase_fk FOREIGN KEY (id_clase)
REFERENCES escuela_danza.clase (id_clase) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;

--relacion alumno clase_alumno--
ALTER TABLE escuela_danza.clase_alumno ADD CONSTRAINT alumno FOREIGN KEY (id_alumno)
REFERENCES escuela_danza.alumno (id_alumno) MATCH FULL
ON DELETE CASCADE ON UPDATE CASCADE;
```

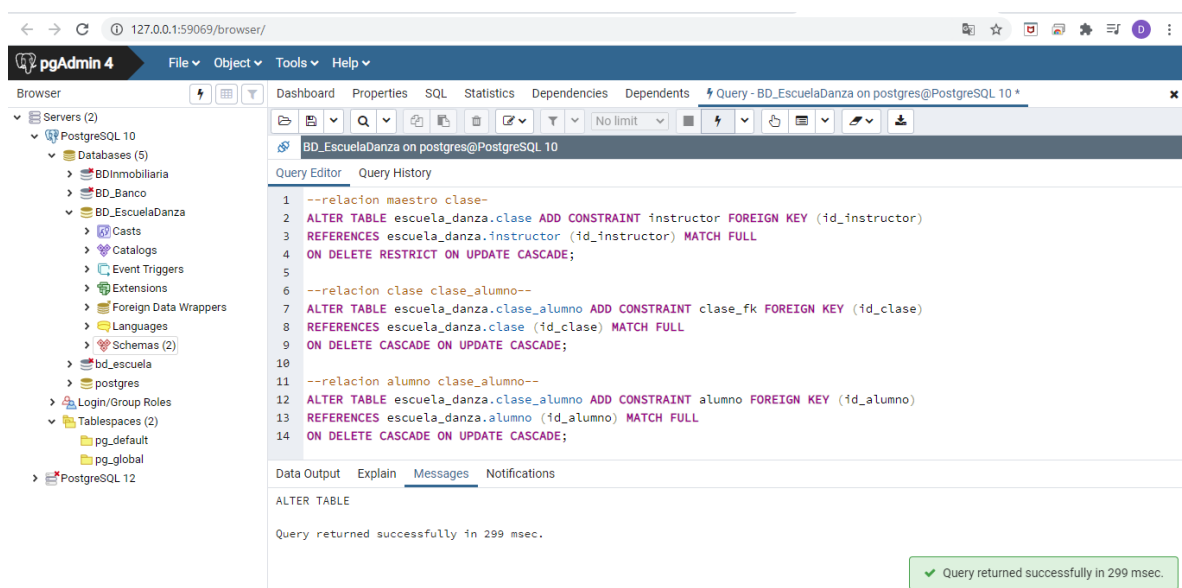


Ilustración 8 Relaciones

Funciones

Agregar instructor

```
CREATE OR REPLACE FUNCTION escuela_danza.agregar_instructor(
inombre VARCHAR(50),iapellidomat VARCHAR (50),iapellidopat VARCHAR (50),i
fechanac DATE) RETURNS void AS $$
BEGIN
    INSERT INTO escuela_danza.instructor(
        nombre_instructor, apellido_mat_instructor, apellido_pat_instructor,
        fecha_nac_instructor)
        VALUES (inombre, iapellidomat, iapellidopat, ifechanac);
END
$$ LANGUAGE plpgsql;
```

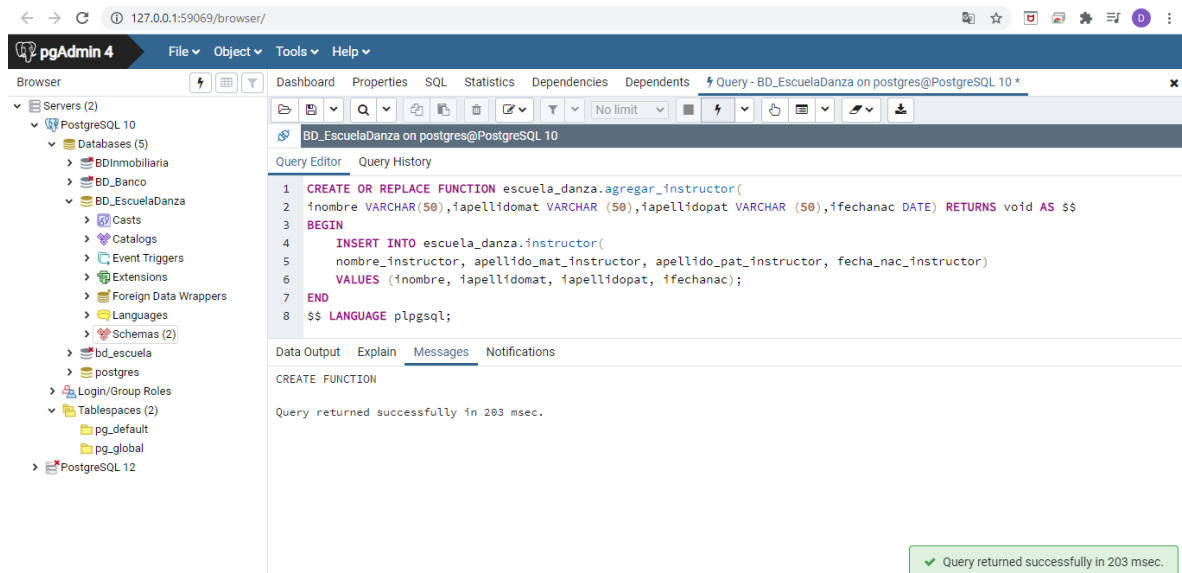


Ilustración 9 Función agregar instructor

Agregar alumno

```
CREATE OR REPLACE FUNCTION escuela_danza.agregar_alumno(
anombre VARCHAR(50),aapellidomat VARCHAR (50),aapellidopat VARCHAR (50),a
fechanac DATE) RETURNS void AS $$
BEGIN
    INSERT INTO escuela_danza.alumno(
        nombre_alumno, apellido_mat_alumno, apellido_pat_alumno, fecha_nac_al
        umno)
        VALUES (anombre, aapellidomat,aapellidopat,afechanac);
END
$$ LANGUAGE plpgsql;
```

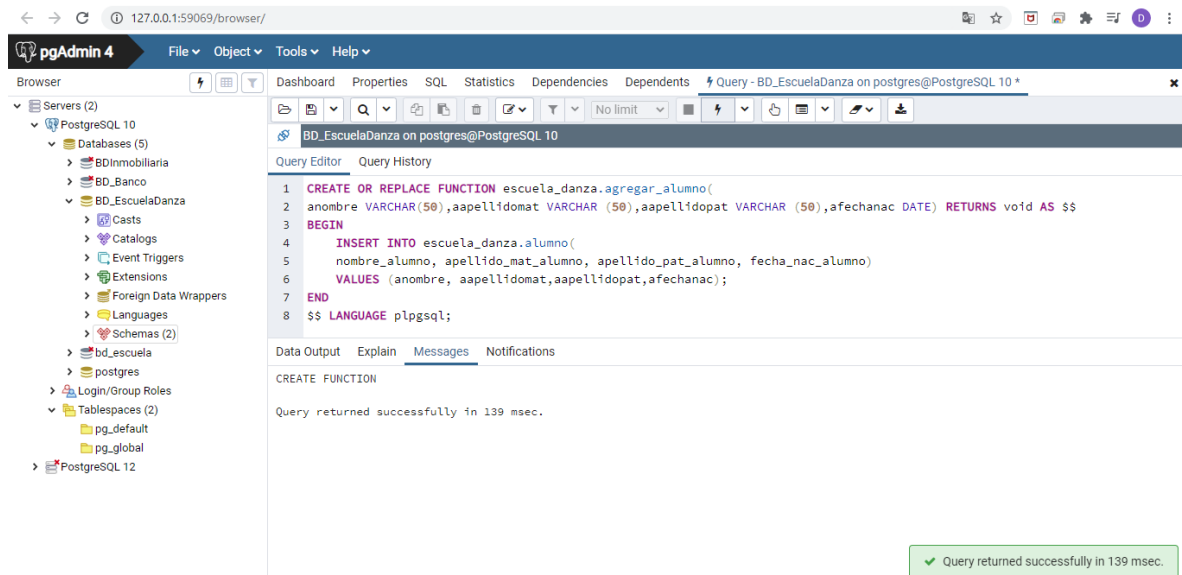


Ilustración 10 Función agregar alumno

Agregar clase

```

CREATE OR REPLACE FUNCTION escuela_danza.agregar_clase(
cnombre VARCHAR(50),hora TIME,lugares_dis SMALLINT, instructor SMALLINT )
  RETURNS void AS $$
BEGIN
  INSERT INTO escuela_danza.clase(
    nombre_clase, hora_clase, lugares_disponibles, id_instructor)
    VALUES (cnombre, hora, lugares_dis, instructor);
END
$$ LANGUAGE plpgsql;

```

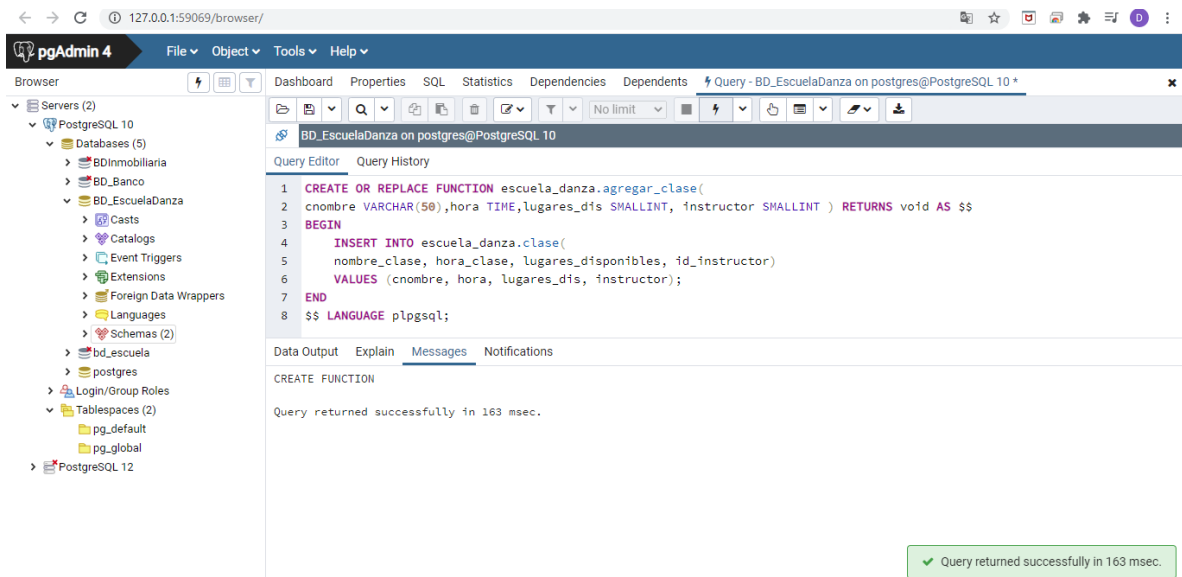


Ilustración 11 Función agregar clase

Inscribir alumno a clase

```
CREATE OR REPLACE FUNCTION escuela_danza.inscribir_alumnoAclase(
id_alumnoi SMALLINT,id_clasei SMALLINT ) RETURNS void AS $$
BEGIN
  INSERT INTO escuela_danza.clase_alumno(
    id_alumno, id_clase)
  VALUES (id_alumnoi,id_clasei);
END
$$ LANGUAGE plpgsql;
```

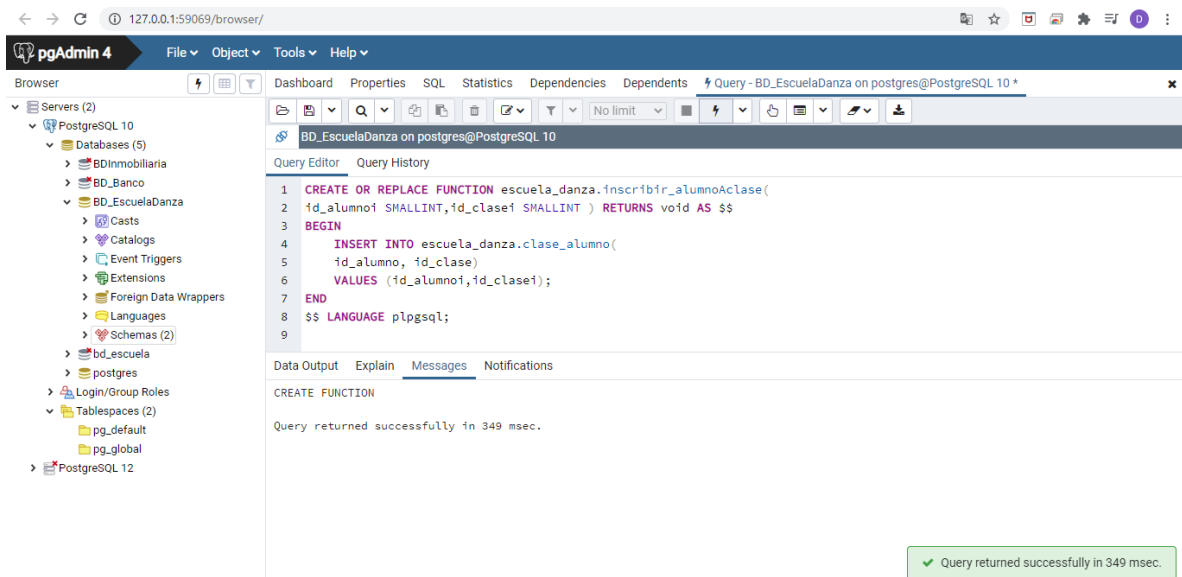


Ilustración 12 Función inscripción

Triggers

Trigger para validar hora en tabla clase

```
CREATE OR REPLACE FUNCTION escuela_danza.valida_hora()
RETURNS trigger
LANGUAGE 'plpgsql'
COST 100
VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
vhora TIME:='00:00';
reg RECORD;
bandera_disp BIT:=0;
bandera_horario BIT:=0;
bandera_hExacta BIT:=0;
minutos integer:=0;
BEGIN
    IF(TG_OP='INSERT') THEN
        FOR REG IN SELECT hora_clase FROM escuela_danza.clase LOOP
            vhora:= reg.hora_clase;
            IF (vhora=NEW.hora_clase) THEN
                bandera_disp:=1;
            END IF;
        END LOOP;

        IF (bandera_disp=1::BIT) THEN
            raise notice 'La hora esta ocupada por otra clase';
        END IF;

        IF (NEW.hora_clase < '08:00' OR NEW.hora_clase > '19:00' ) THEN
            bandera_horario:=1;
        END IF;

        IF (bandera_horario=1::BIT) THEN
            raise notice 'La hora se encuentra fuera del horario de la escuela';
        END IF;

        minutos:=extract (minute from NEW.hora_clase);

        IF minutos!=0 THEN
            bandera_hExacta:=1;
        END IF;

        IF (bandera_hExacta=1::BIT) THEN
```

```

        raise notice 'La hora debe ser exacta, sin minutos';
    END IF;

    IF (bandera_disp=0::BIT AND bandera_horario=0::BIT AND bandera_hExacta=0::BIT) THEN
        RETURN NEW;
    END IF;
END IF;
RETURN NULL;
END;$BODY$;

```

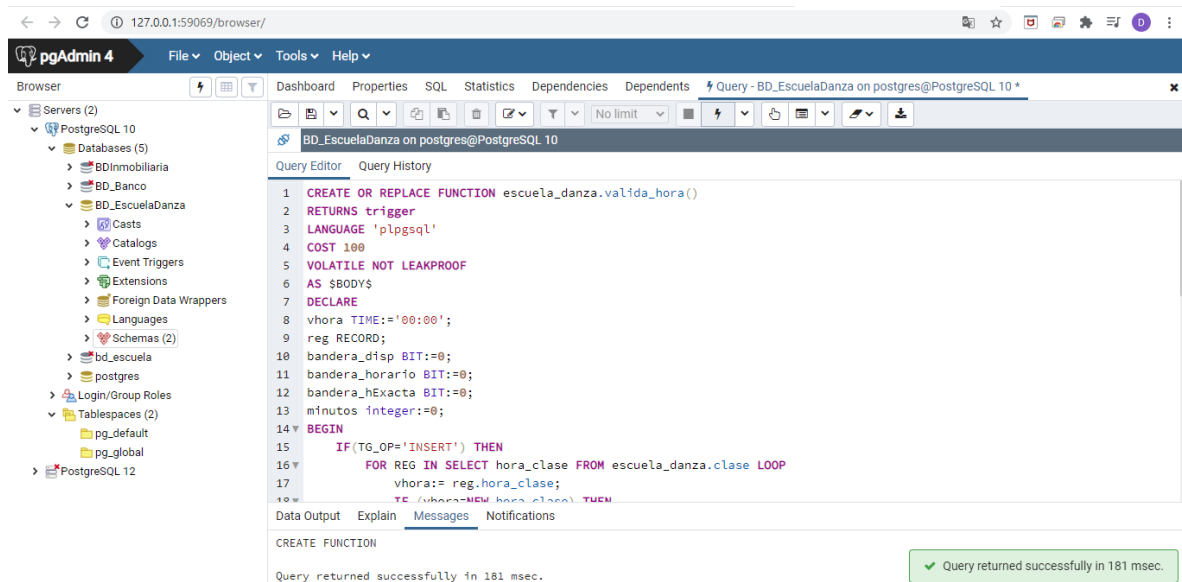


Ilustración 13 Trigger para validar hora de clase

Asociación de Trigger en la tabla clase

```

CREATE TRIGGER vhoraClase
BEFORE INSERT
ON escuela_danza.clase
FOR EACH ROW
EXECUTE PROCEDURE escuela_danza.valida_hora();

COMMENT ON TRIGGER vhoraClase ON escuela_danza.clase
IS 'Trigger que valida que la hora de la clase este disponible';

```

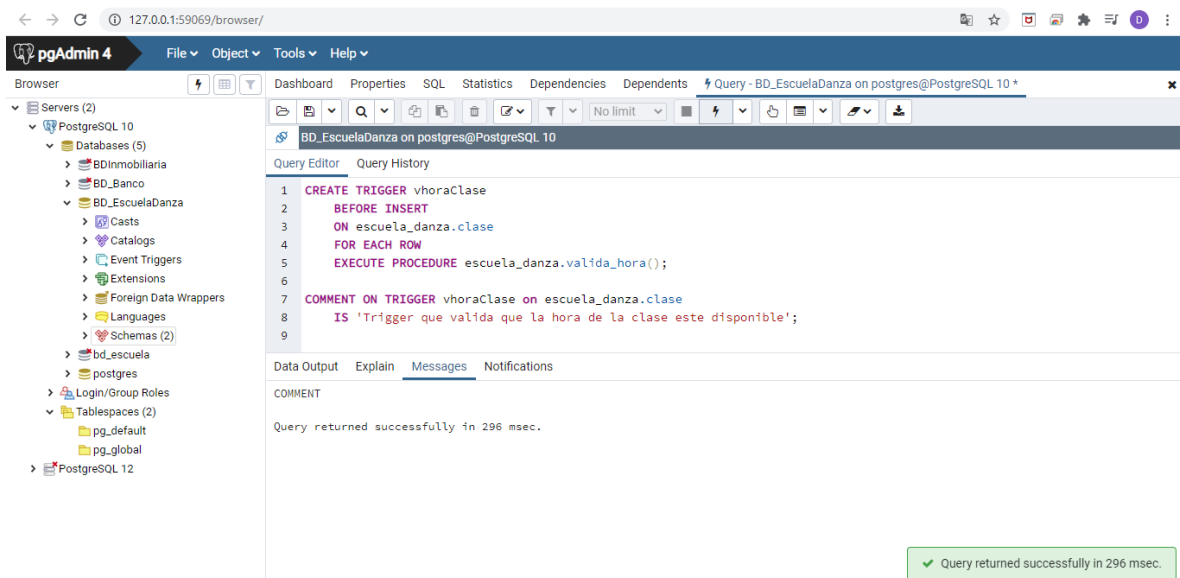



Ilustración 14 Asociación de trigger a la tabla clase

Trigger para validar inscripciones

```
CREATE OR REPLACE FUNCTION escuela_danza.valida_inscripcion()
RETURNS trigger
LANGUAGE 'plpgsql'
COST 100
VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
lugares integer:=0;
BEGIN
    IF(TG_OP='INSERT') THEN
        SELECT lugares_disponibles into lugares FROM escuela_danza.clase
        WHERE id_clase = NEW.id_clase;
        IF lugares>0 THEN
            UPDATE escuela_danza.clase
            SET lugares_disponibles=lugares_disponibles-1
            WHERE id_clase=NEW.id_clase;
            RETURN NEW;
        ELSE
            raise notice 'No hay lugares disponibles en esta clase';
        END IF;
    END IF;
    RETURN NULL;
END;$BODY$;
```

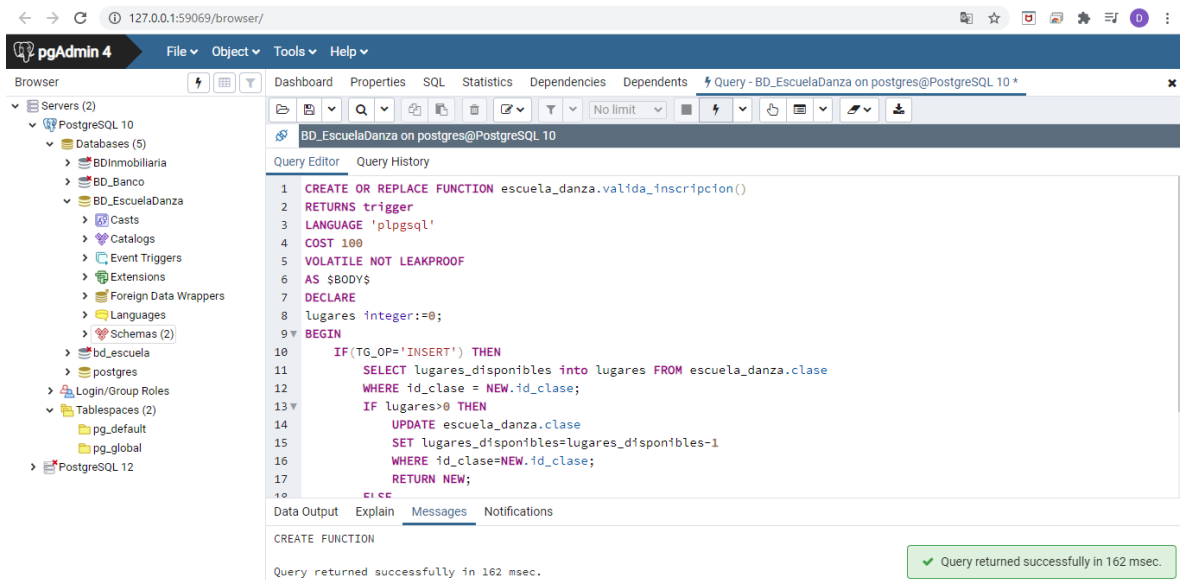


Ilustración 15 Trigger para validar inscripción

Asociación de Trigger en la tabla clase_alumno

```

CREATE TRIGGER vinscripcion
  BEFORE INSERT
  ON escuela_danza.clase_alumno
  FOR EACH ROW
  EXECUTE PROCEDURE escuela_danza.valida_inscripcion();

COMMENT ON TRIGGER vinscripcion ON escuela_danza.clase_alumno
  IS 'Trigger que valida que la inscripción de alumnos a las clases';

```

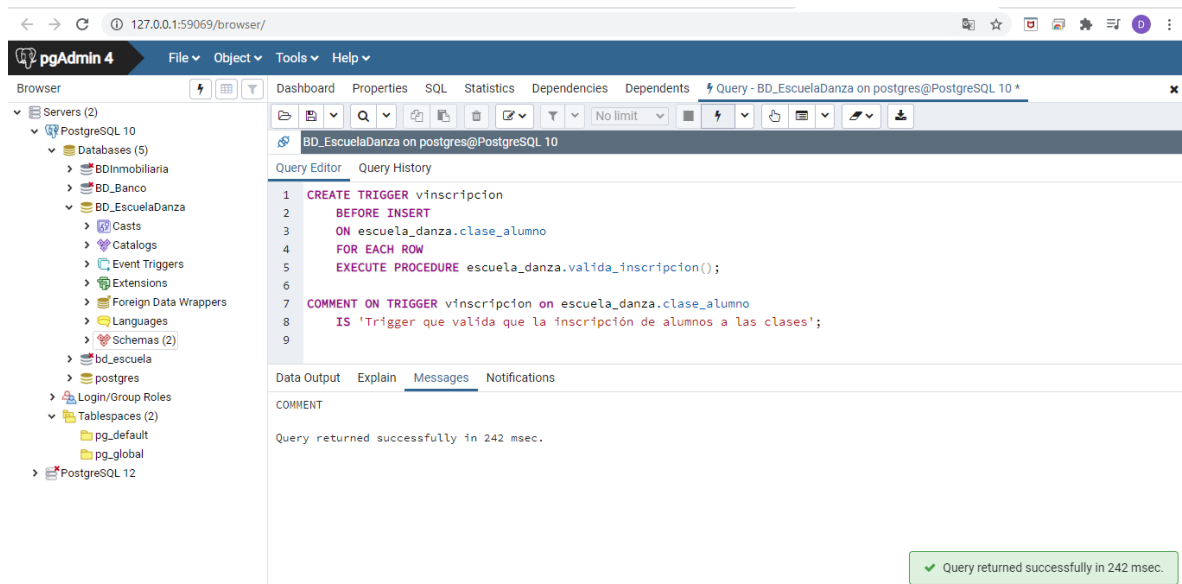


Ilustración 16 Asociación de trigger a tabla clase_alumno

Vistas

Vista instructores

```

CREATE OR REPLACE VIEW escuela_danza.v_instructores
AS
SELECT id_instructor as "Id Instructor",
CONCAT(nombre_instructor, ' ', apellido_mat_instructor, ' ', apellido_pat_instructor) as "Nombre Completo",
fecha_nac_instructor as "Fecha de nacimiento"
FROM escuela_danza.instructor;

```

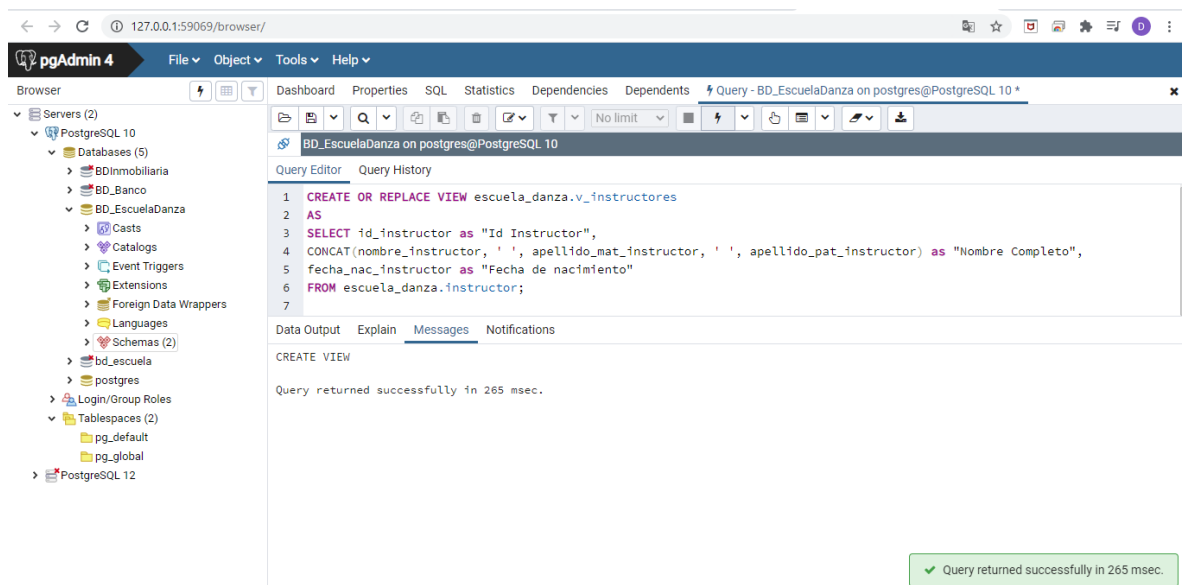


Ilustración 17 Vista instructores

Vista alumnos

```
CREATE OR REPLACE VIEW escuela_danza.v_alumnos
AS
SELECT id_alumno as "Id Alumno",
CONCAT(nombre_alumno, ' ', apellido_mat_alumno, ' ', apellido_pat_alumno)
  as "Nombre Completo",
fecha_nac_alumno as "Fecha de nacimiento"
FROM escuela_danza.alumno;
```

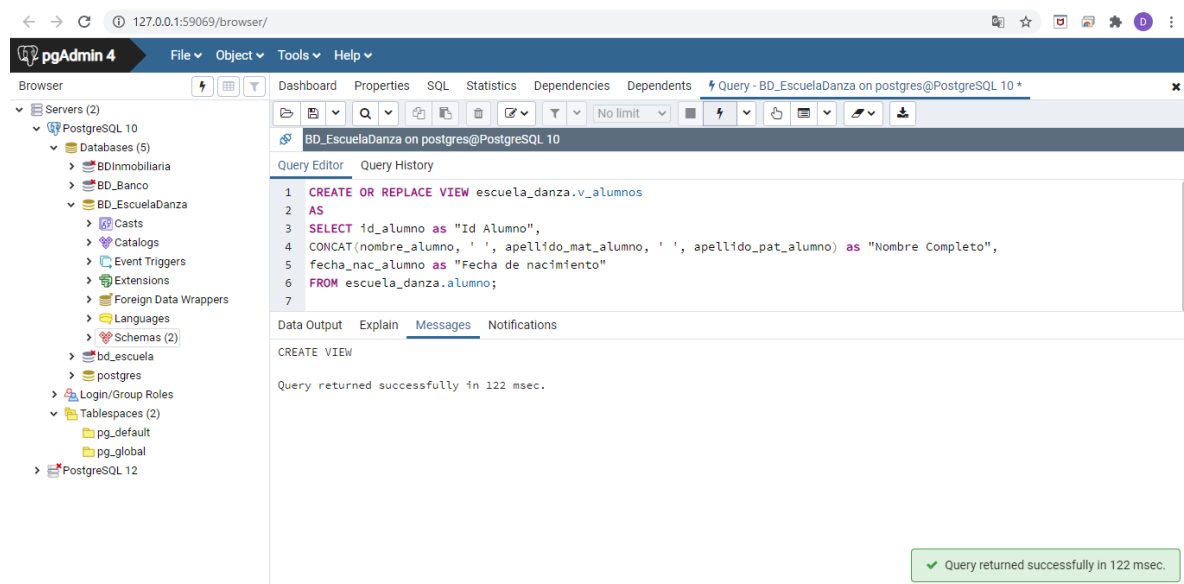


Ilustración 18 Vista alumnos

Vista clases

```
CREATE OR REPLACE VIEW escuela_danza.v_clases
AS
SELECT nombre_clase as "Clase",
hora_clase as "Hora",
CONCAT(instructor.nombre_instructor, ' ', instructor.apellido_mat_instructor, ' ', instructor.apellido_pat_instructor) as "Instructor",
lugares_disponibles as "Lugares Disponibles"
FROM escuela_danza.clase
INNER JOIN escuela_danza.instructor
ON clase.id_instructor = instructor.id_instructor
ORDER BY hora_clase;
```

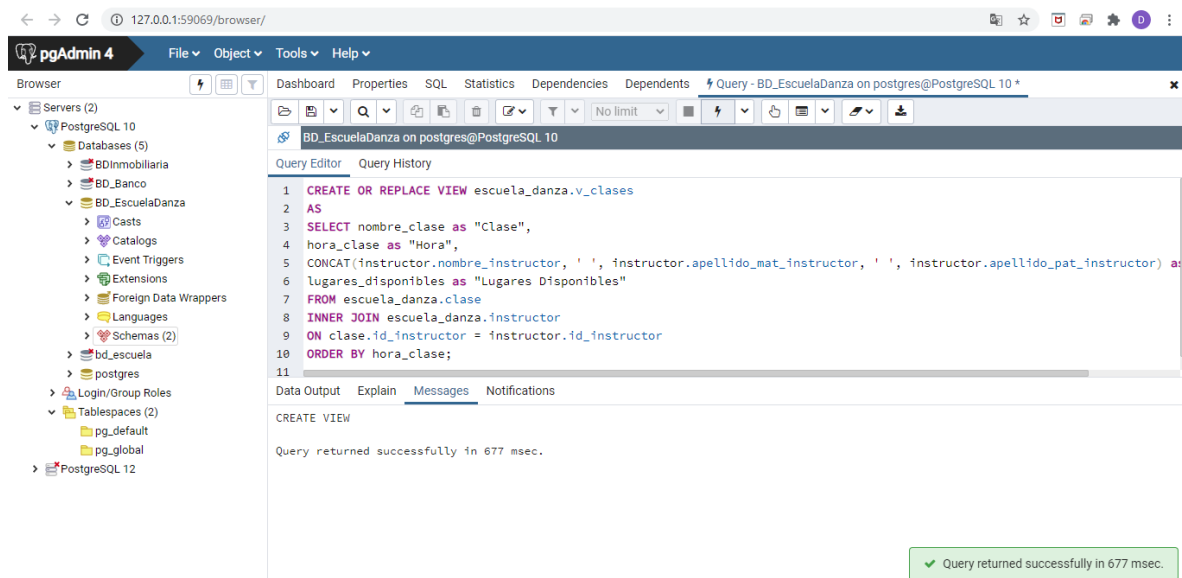


Ilustración 19 Vista clases

Vista inscripciones

```
CREATE OR REPLACE VIEW escuela_danza.v_clasesInscritasAlumno
AS
SELECT clase_alumno.id_alumno as "Id Alumno",
CONCAT(alumno.nombre_alumno, ' ', alumno.apellido_mat_alumno, ' ', alumno
.apellido_pat_alumno) as "Nombre Completo",
clase.nombre_clase as "Clase",
clase.hora_clase as "Hora",
CONCAT(instructor.nombre_instructor, ' ', instructor.apellido_mat_instruc
tor, ' ', instructor.apellido_pat_instructor) as "Instructor"
FROM escuela_danza.clase_alumno
INNER JOIN escuela_danza.alumno
ON clase_alumno.id_alumno = alumno.id_alumno
INNER JOIN escuela_danza.clase
ON clase_alumno.id_clase = clase.id_clase
INNER JOIN escuela_danza.instructor
ON clase.id_instructor = instructor.id_instructor
ORDER BY clase_alumno.id_alumno,clase.hora_clase;
```

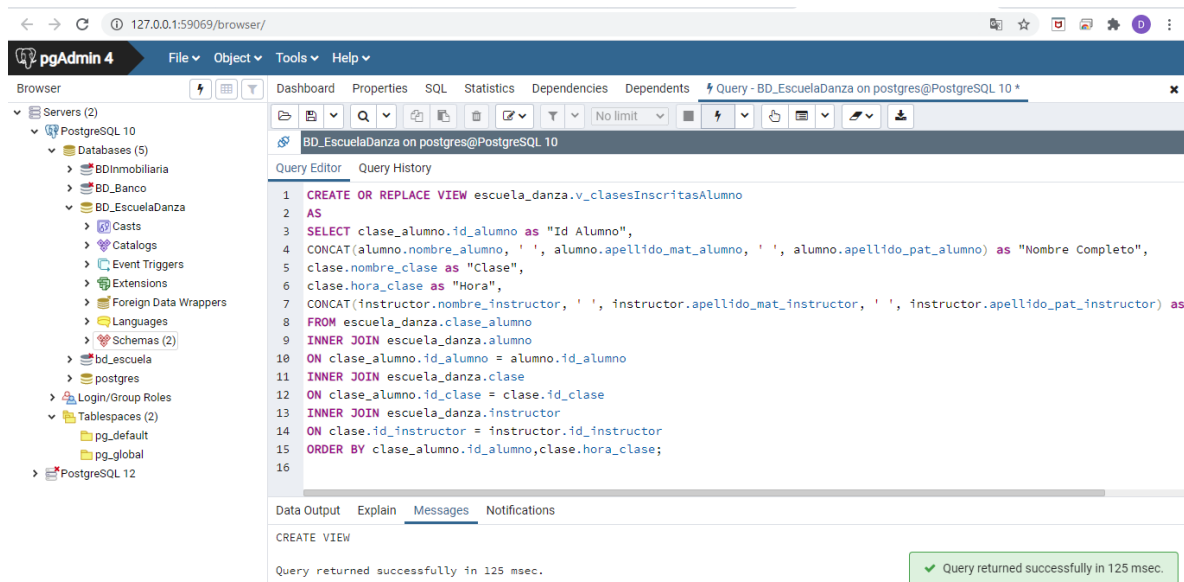


Ilustración 20 Vista inscripciones

Ingreso y visualización de datos utilizando las funciones, triggers y vistas implementadas

Agregar instructores

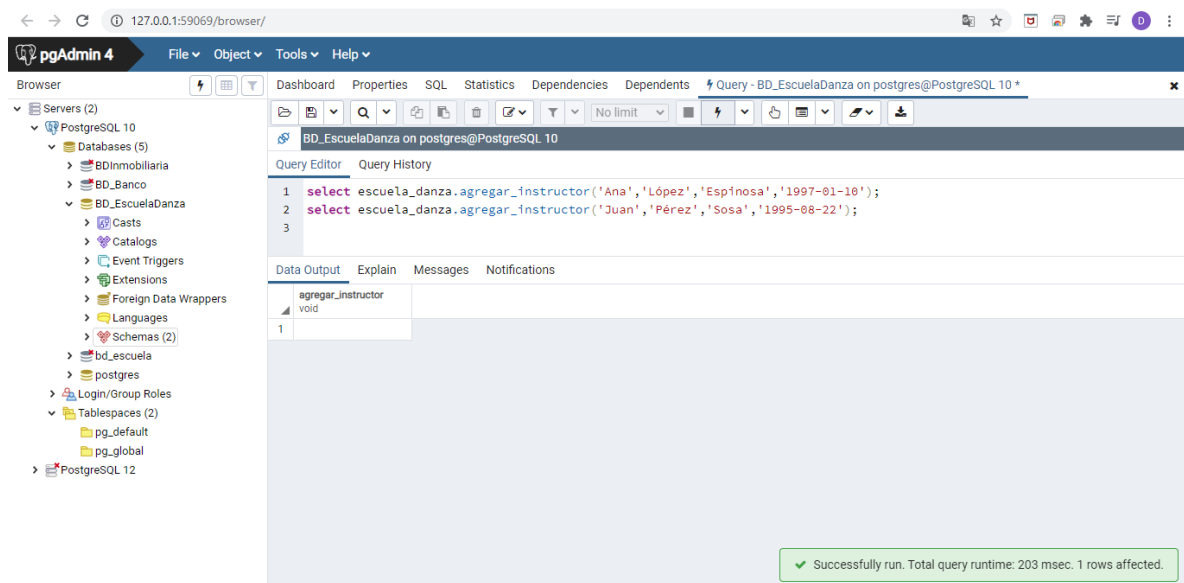
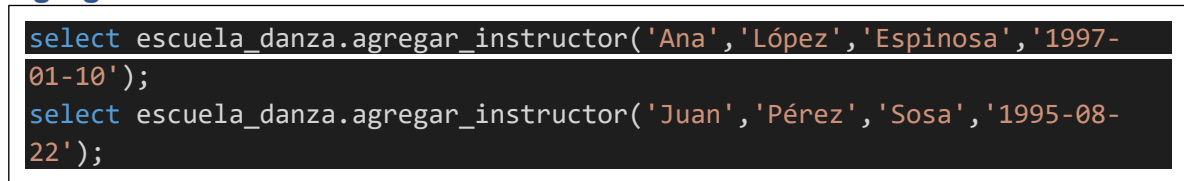
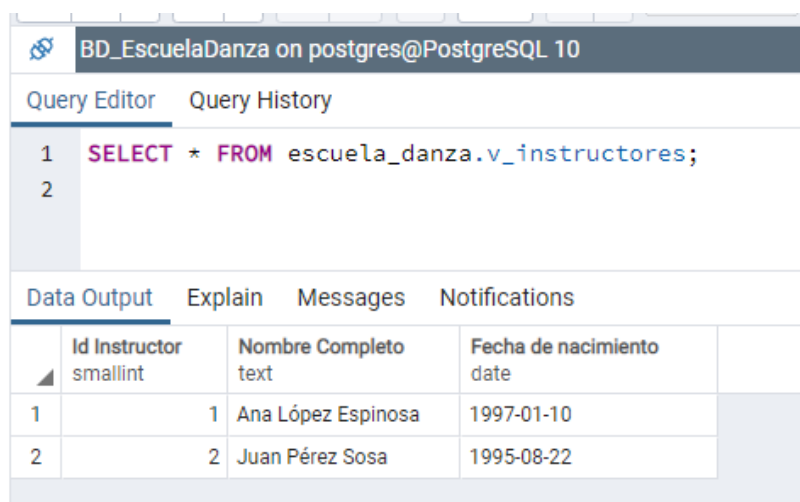


Ilustración 21 Agregar instructores

Vista instructores

```
SELECT * FROM escuela_danza.v_instructores;
```



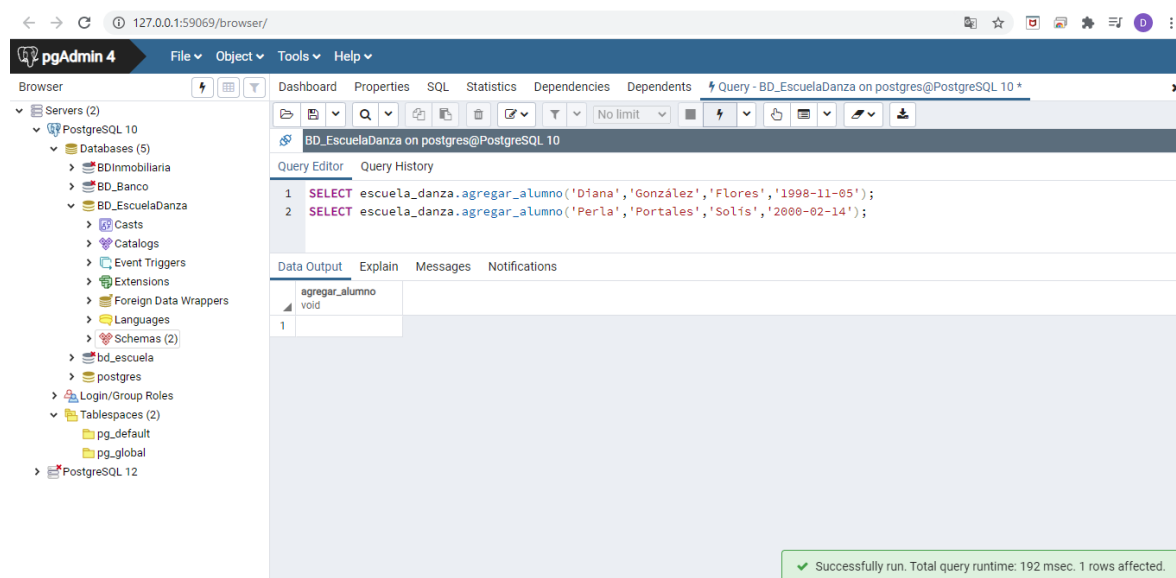
The screenshot shows the pgAdmin 4 interface. The top bar indicates the connection is 'BD_EscuelaDanza on postgres@PostgreSQL 10'. Below this, there are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing the SQL query: `SELECT * FROM escuela_danza.v_instructores;`. Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table with the following data:

	Id Instructor smallint	Nombre Completo text	Fecha de nacimiento date
1	1	Ana López Espinosa	1997-01-10
2	2	Juan Pérez Sosa	1995-08-22

Ilustración 22 Instructores registrados

Agregar alumnos

```
SELECT escuela_danza.agregar_alumno('Diana','González','Flores','1998-11-05');  
SELECT escuela_danza.agregar_alumno('Perla','Portales','Solís','2000-02-14');
```



The screenshot shows the pgAdmin 4 interface. The top bar indicates the connection is 'BD_EscuelaDanza on postgres@PostgreSQL 10'. Below this, there are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing the SQL queries: `SELECT escuela_danza.agregar_alumno('Diana','González','Flores','1998-11-05');` and `SELECT escuela_danza.agregar_alumno('Perla','Portales','Solís','2000-02-14');`. Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table with the following data:

	agregar_alumno void
1	

At the bottom of the interface, a green status bar indicates: 'Successfully run. Total query runtime: 192 msec. 1 rows affected.'

Ilustración 23 Agregar alumnos

Vista alumnos

```
SELECT * FROM escuela_danza.v_alumnos;
```

BD_EscuelaDanza on postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM escuela_danza.v_alumnos;

Data Output

Explain

Messages

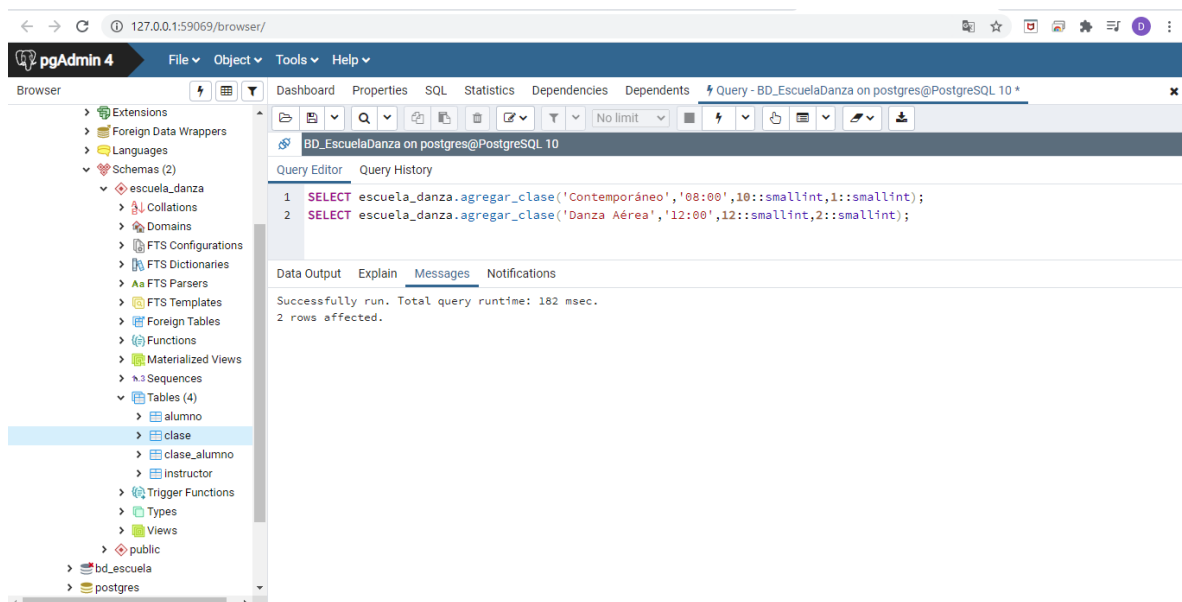
Notifications

	Id Alumno smallint	Nombre Completo text	Fecha de nacimiento date
1	1	Diana González Flores	1998-11-05
2	2	Perla Portales Solís	2000-02-14

Ilustración 24 Alumnos registrados

Agregar clases

```
SELECT escuela_danza.agregar_clase('Contemporáneo','08:00',10::smallint,1::smallint);
SELECT escuela_danza.agregar_clase('Danza Aérea','12:00',12::smallint,2::smallint);
```



The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays the database structure, with 'escuela_danza' expanded. The 'Query Editor' pane on the right contains two SQL queries:


```
1 SELECT escuela_danza.agregar_clase('Contemporáneo','08:00',10::smallint,1::smallint);
2 SELECT escuela_danza.agregar_clase('Danza Aérea','12:00',12::smallint,2::smallint);
```

 The 'Messages' pane at the bottom indicates that the queries were executed successfully, with a total runtime of 182 msec and 2 rows affected.

Ilustración 25 Agregar clases

Validación de horario de la clase con trigger



Ilustración 26 Validación de hora de clase

Vista clases

```
SELECT * FROM escuela_danza.v_clases;
```

The screenshot shows a PostgreSQL query editor window titled "BD_EscuelaDanza on postgres@PostgreSQL 10". The "Query Editor" tab is active, displaying a SQL query: `1 SELECT * FROM escuela_danza.v_clases;`. Below the query, the "Data Output" tab is selected, showing a table with 5 columns: "Clase", "Hora", "Instructor", and "Lugares Disponibles". The table contains two rows of data.

	Clase character varying (50)	Hora time without time zone	Instructor text	Lugares Disponibles smallint
1	Contemporáneo	08:00:00	Ana López Espinosa	10
2	Danza Aérea	12:00:00	Juan Pérez Sosa	12

Ilustración 27 Clases registradas

Inscribir alumno a clase

```
select escuela_danza.inscribir_alumnoAclase(1::SMALLINT,1::SMALLINT);  
select escuela_danza.inscribir_alumnoAclase(2::SMALLINT,2::SMALLINT);
```

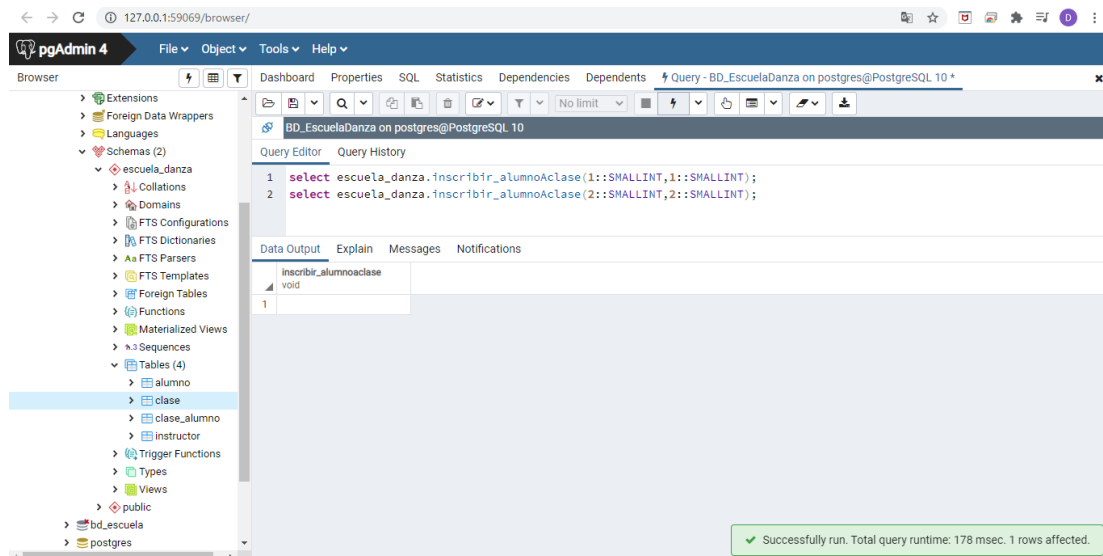


Ilustración 28 Inscribir alumnos a clases

Validación de inscripción con trigger

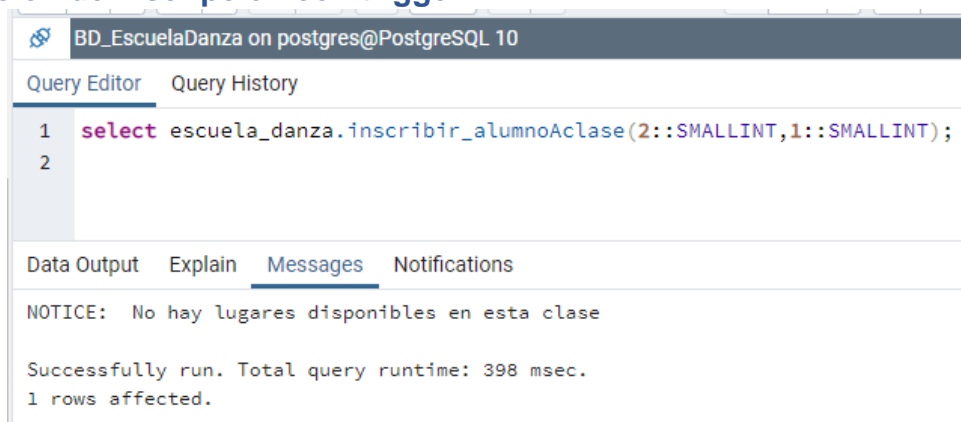


Ilustración 29 Validación de inscripción, si no hay lugares disponibles, no se puede inscribir el alumno

Vista inscripciones

```
SELECT * FROM escuela_danza.v_clasesInscritasAlumno;
```

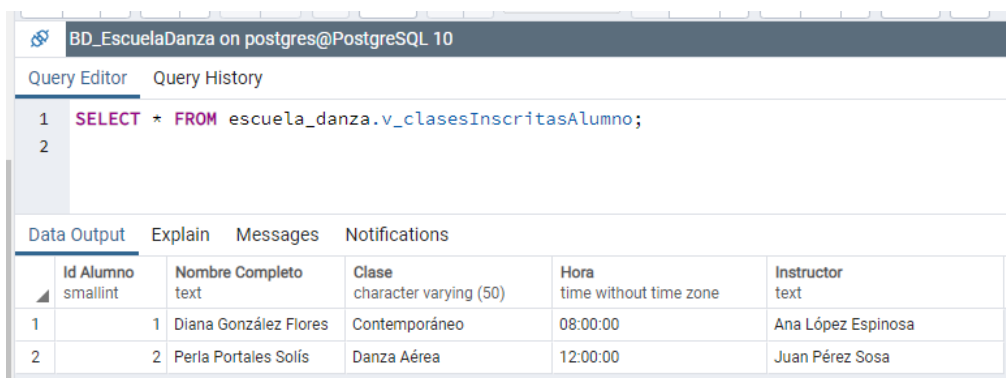


Ilustración 30 Clases inscritas por los alumnos

Conclusiones

Con la realización de este proyecto se pudo aplicar todo el conocimiento adquirido en clase y considero que lo podré seguir aplicando de ahora en adelante, ya que las bases de datos se utilizan en un sinnúmero de lugares y situaciones.

Al ser el desarrollo de software (específicamente desarrollo web) una de las cosas que más me interesa en el campo de las Tecnologías de la Información, siento que me será muy útil todo lo visto en clase, ya que como se puede observar en el presente proyecto, considero que aprendí lo suficiente para diseñar e implementar bases de datos aptas para los requerimientos que se tengan.

Me pareció muy interesante el uso de funciones y triggers en una base de datos ya que esto permitió que directamente en ésta se realicen procedimientos que validen y aseguren los datos que se van ingresando, además de facilitar la visualización de los mismos.

Anteriormente había diseñado bases de datos, pero para nada se parece a lo que logré en este proyecto, ya que antes no tenía muy claros algunos conceptos y otros nunca los había escuchado, siento que aun me falta mucho por aprender, pero con estos conocimientos tengo una base para poder investigar más y comprender más información.

Incluso no sabía bien los tipos de datos disponibles en una base de datos, si tenía que guardar un número entero siempre lo guardaba como un Integer, aunque el rango de los números a guardar fuera pequeño y no necesitara tantas cifras, ahora puedo realizar un análisis para saber que tipo de dato es el más apropiado para un campo, ya que es importante no desperdiciar espacio de almacenamiento.

Para mí es fácil apreciar que hay un antes y después de esta materia, ya que en semestres anteriores me era muy difícil realizar un SELECT con un JOIN y ahora los puedo realizar incluso con varios JOIN, como se puede observar en la vista inscripciones.

Por otra parte, cuando por ejemplo, estaba desarrollando un sitio web, realizaba muchas cosas en el código de la aplicación, lo cual no es lo más óptimo. Por ejemplo hacia selects y con los datos obtenidos hacia validaciones, realizaba consultas extensas, cargaba muchos datos directamente de la base de datos, etc. Esto traía consigo varios problemas, pues me tardaba más tiempo en detectar errores, las consultas eran más tardadas e incluso podría haber problemas de seguridad. Antes no tomaba tanto en cuenta la magnitud que podrían tener estos problemas, ya que se trataban de sistemas pequeños, pero si se tratara de un sistema más grande, estos problemas se multiplicarían y podrían tener efectos negativos en el producto final.

Al unir estos conceptos con los de otras materias podré desarrollar proyectos más complejos, eficientes y óptimos, que incluso me podrán ayudar en el campo laboral en el caso que me encuentre con retos de este tipo.

Bibliografía

Elgabry, O. (14 de Septiembre de 2016). *Database — Fundamentals* . Obtenido de Omar Elgabry's Blog: <https://medium.com/omarelgabrys-blog/database-fundamentals-part-2-b841032243ac>

Silberschatz, A., Korth, H., & Sudarshan, S. (2005). *Database System Concepts*. McGraw Hill.

w3schools. (2012). *SQL Tutorial*. Obtenido de w3schools: <https://www.w3schools.in/sql/>