

LENGUAJES SQL TRANSACCIONAL EN



FAVA - Formación en Ambientes Virtuales de Aprendizaje

SENA - Servicio Nacional de Aprendizaje.

Estructura de contenidos

	Pág.
Introducción	3
Mapa de contenidos	4
1. Generalidades	5
2. Procedimientos almacenados	6
2.1. Creación de procedimientos almacenados sin parámetros en MySQL.....	9
2.2. Creación de procedimientos almacenados con parámetros	12
2.2.1. Creación de procedimientos almacenados con parámetros en MySQL	13
2.2.2. Creación de procedimientos almacenados con parámetros tipo IN en Oracle	16
2.2.3. Creación de procedimientos almacenados con parámetros tipo OUT en Oracle	20
2.2.4. Creación de procedimientos almacenados con parámetros tipo IN y OUT en Oracle	22
2.2.5. Creación de procedimientos almacenados sin parámetros en SQL Server	23
2.2.6. Creación de procedimientos almacenados con parámetros en SQL Server.....	25
3. Funciones	29
3.1. Funciones en MySQL	29
3.2. Funciones en Oracle	30
3.3. Funciones en SQL Server	30
4. Triggers.....	33
4.1. Triggers en MySQL.....	33
4.2. Trigger en Oracle.....	38
4.3. Trigger en SQL Server.....	46
Glosario	50
Bibliografía.....	51
Control de documento	52

LENGUAJE TRANSACCIONAL SQL

Introducción

En el desarrollo de aplicaciones ya sea cliente-servidor, ambientes web o aplicaciones móviles, habrá interacción con Sistemas de Gestión de Bases de Datos SGBD, por esto la importancia de que los Analistas y Desarrolladores de Software conozcan las diferentes herramientas y sus características que permitan automatizar procesos o consultas a nivel de bases de datos.

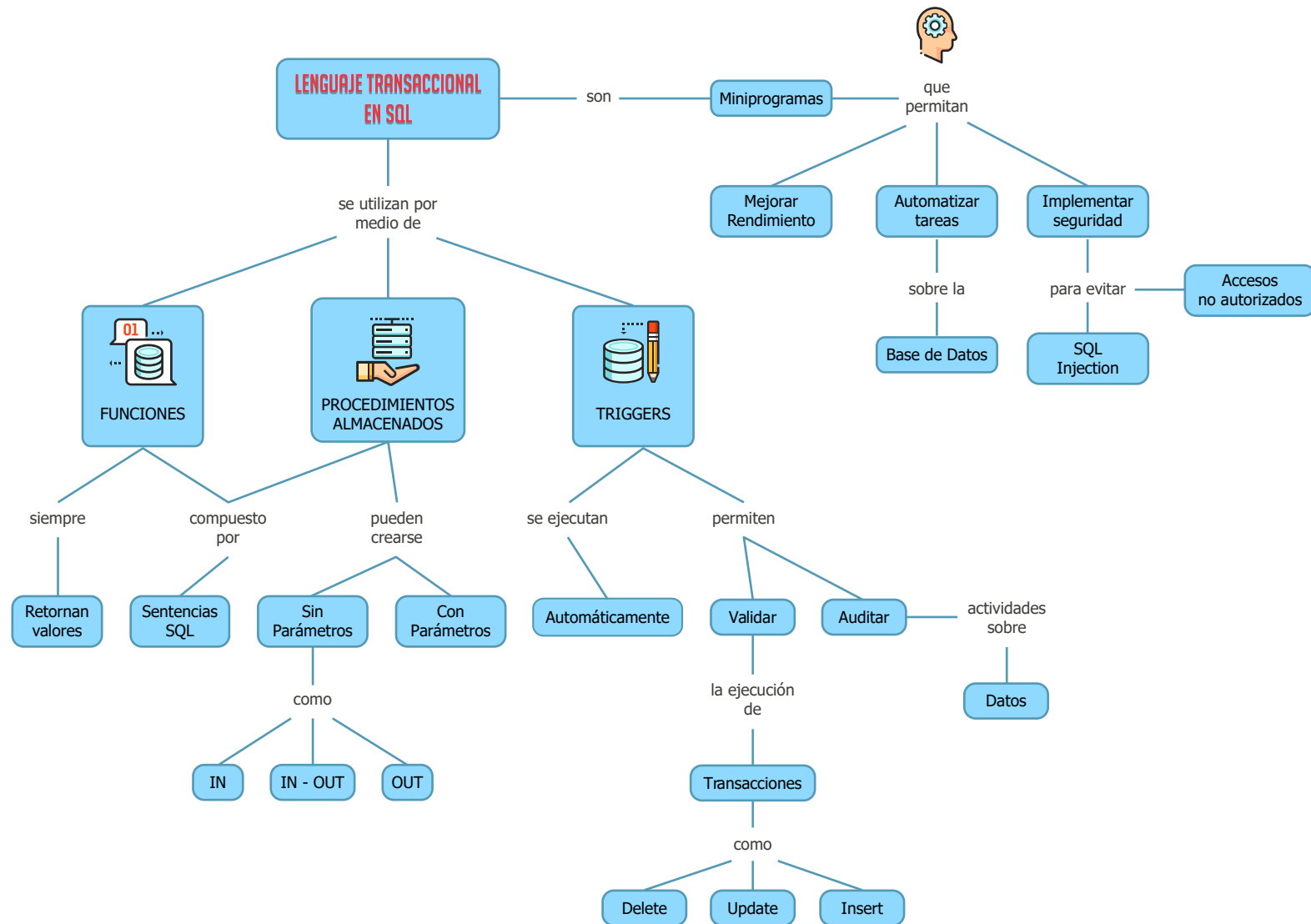
Existen tres herramientas básicas que son los Procedimientos Almacenados, Funciones y Triggers que son comunes en su conceptualización en los diferentes motores de bases de datos, pero que difieren en su codificación.

En este recurso didáctico se hará una introducción a cada una de estas herramientas, especificando para que sirven, en qué momento se deben de utilizar, su sintaxis, si reciben o no parámetros, cómo son ejecutadas y los valores devueltos. Igualmente, se presentan ejemplos prácticos de su utilización para una mejor comprensión de los temas tratados.

Es importante anotar que el conocimiento completo de estas herramientas y ante los avances de cada uno de los motores de bases de datos como MySQL, Oracle y SQL Server se requiere de una mayor dedicación al aprendizaje de estos temas. Debido a esto, una vez apropiado este recurso didáctico el aprendiz podrá profundizar estas temáticas mediante la investigación o consultas a la referencias bibliográficas al final de este material.



Mapa de contenidos



Desarrollo de contenidos

1. Generalidades

Después de haber realizado las respectivas fases de análisis y construcción de bases de datos, llega el momento de sistematizar algunos procesos para el manejo de las diferentes operaciones en una base de datos. En este objeto de aprendizaje se trabajará en los fundamentos para aplicar el lenguaje transaccional en la implementación de funcionalidades en el SGBD (Sistema de Gestión de Bases de Datos).

Se centrará la atención en la construcción de programas usando lenguaje transaccional SQL, con el fin de proporcionar funcionalidades que son implementadas en la base de datos y utilizadas por la capa de datos.

Las implementaciones más comunes son los procedimientos almacenados (stored procedures), las funciones (functions) y los desencadenadores (triggers), en este objeto de aprendizaje todos estos serán referenciados para los Sistemas de Gestión de Base de datos MySQL, Oracle y SQL Server.



2. Procedimientos almacenados

Un procedimiento almacenado es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez esté almacenado, los clientes no necesitan volver a ejecutar los comandos individuales y en vez de esto pueden ejecutar el procedimiento almacenado.

Los procedimientos almacenados son subprogramas que ejecutan una acción específica y no devuelven ningún valor. Tienen un nombre, un conjunto de parámetros (opcional) y un bloque de código. Dependiendo del Sistema de Gestión de Base de Datos (SGBD) la sintaxis y los parámetros pueden variar.

Una vez se crean los procedimientos almacenados, el sistema no tiene necesidad de verificar la sintaxis de la instrucción SQL, porque solo con utilizar el nombre del procedimiento, se ejecuta la instrucción.

Los procedimientos almacenados permiten implementar elementos de seguridad a las aplicaciones, porque evitan la manipulación directa de los datos, los usuarios de los procedimientos deben seguir procesos de autenticación y no importaría el lenguaje o plataforma de acceso de dónde son invocados.

Por otro lado, mejoran el rendimiento en los equipos clientes, debido a que envía menos información al servidor, aunque aumenta la carga del servidor.

La estructura de un procedimiento en general consta de un nombre, la lista de parámetros en caso de requerirlos y el cuerpo del procedimiento (definición).

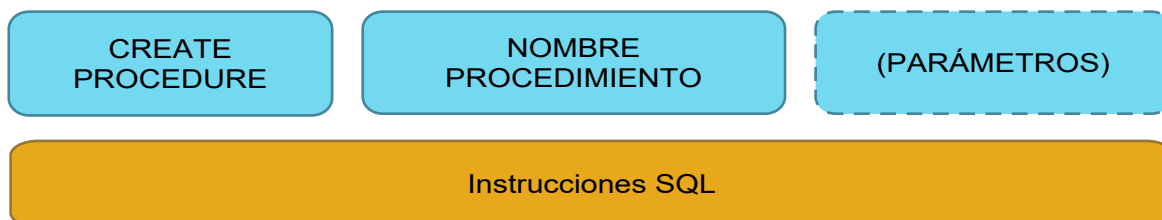


Figura 1. Estructura de un procedimiento almacenado.

Por cada uno de los parámetros que se requieran se debe hacer la siguiente especificación:

[IN | OUT | INOUT] Nombre_del_parametro tipo_de_dato

Algunas de las palabras reservadas a utilizar en la construcción del procedimiento almacenado pueden ser:

- **IN:** Indica que el parámetro será de entrada.
- **OUT:** Indica que el parámetro será de salida.
- **INTOUT:** Indica que el parámetro será tanto de Entrada como de salida.
- **BEGIN:** Limitador del procedimiento.
- **END:** Fin de nuestro procedimiento.
- **DELIMITER:** Restablece el punto y coma como delimitador.
- **CALL:** para llamar al procedimiento una vez creado.

La sintaxis y palabras reservadas pueden variar dependiendo del Sistema de Gestión de Base de Datos donde se implemente, en este material se expondrá su construcción para MySQL, Oracle y SQL Server, las instrucciones que van al interior del procedimiento son sentencias SQL acompañadas de algunos elementos del lenguaje de programación tales como variables, inicio y fin de bloques, estructuras de control y repetitivas.

Nota: Para los ejemplos presentados en esta plantilla se utilizará la base de datos del consultorio médico tratada en materiales anteriores. A continuación, se presenta el modelo relacional y el diccionario de datos.

Modelo Relacional:

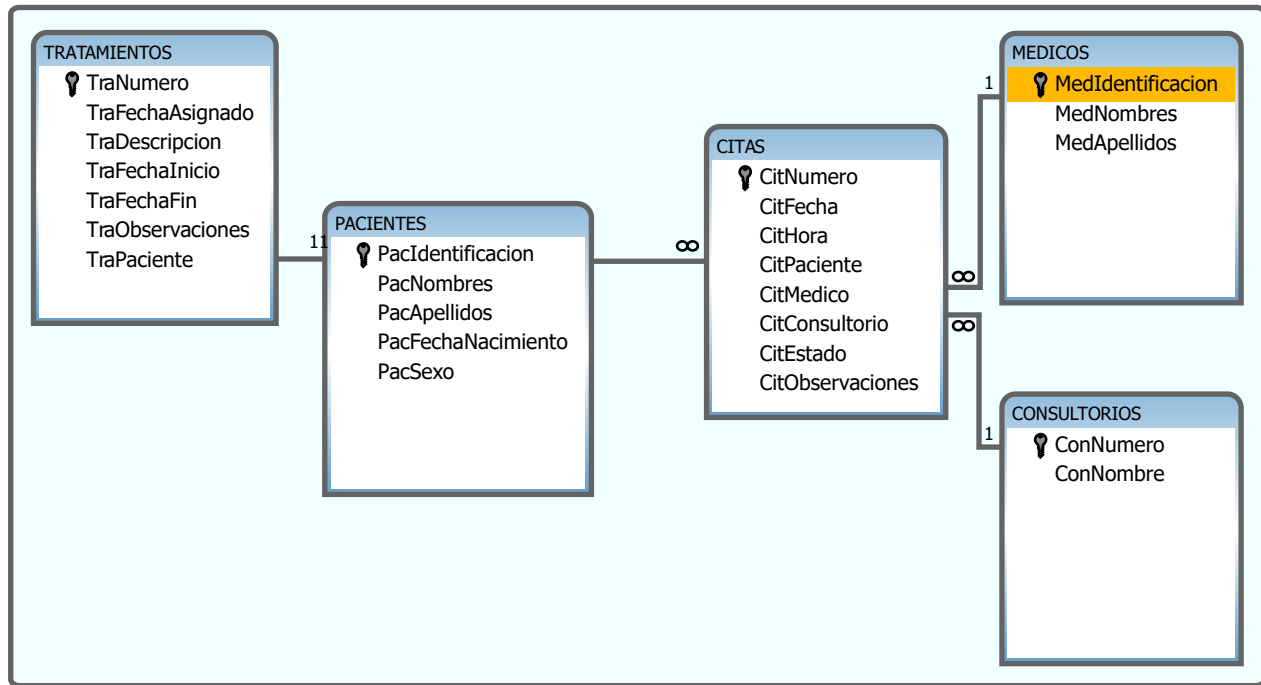


Figura 2. Modelo entidad relación de la base de datos CITAS.

Diccionario de Datos:

Tabla	Atributo – Campo	Tipo de Dato	Longitud	Modificador	Tabla y campo foránea
Pacientes	PacIdentificacion	char	10	primary key, not null	
Pacientes	PacNombres	Varchar	50	not null	
Pacientes	PacApellidos	Varchar	50	not null	
Pacientes	PacFechaNacimiento	Date		not null	
Pacientes	PacSexo	Por tener el Modificador ENUM, no se declara		(ENUM('M', 'F'))	
Medicos	MedIdentificacion	Char	10	primary key, not null	
Medicos	MedNombres	Varchar	50	not null	
Medicos	MedApellidos	Varchar	50	not null	
Consultorios	ConNumero	Int	3	primary key, not null	
Consultorios	ConNombre	Varchar	50	not null	
Tratamientos	TraNumero	Int		primary key, auto_increment	
Tratamientos	TraFechaAsignado	Date		not null	

Tratamientos	TraDescripcion	Text		not null	
Tratamientos	TraFechaInicio	Date		not null	
Tratamientos	TraFechaFin	Date		not null	
Tratamientos	TraObservaciones	Text		not null	
Tratamientos	TraPaciente	Char	10	not null	Pacientes (PacIdentificacion)

Figura 3. Diccionario de datos de la base de datos CITAS.

2.1. Creación de procedimientos almacenados sin parámetros en MySQL

Los procedimientos almacenados y funciones son nuevas funcionalidades de la versión de MySQL 5.0 sigue la sintaxis de SQL (Structured Query Language) para procedimientos almacenados.

Los procedimientos quedan implementados como objetos de la base de datos, por eso para su creación y ejecución se debe estar dentro del contexto de la base de datos y tener permisos para su creación.

El siguiente ejemplo, ilustra un procedimiento básico, que consta de una sola sentencia SQL. El procedimiento almacenado llamado listarconsultorios cuyo objetivo es listar todos los datos de la tabla consultorios se presenta a continuación:

Use citas	Se informa la base de datos en la que se va a trabajar.
create procedure listaconsultorios();	Se informa que se va a crear el procedimiento, Llamado listaconsultorios.
select * from consultorios;	Instrucción SQL, para la consulta.

Al dar enter en el punto y coma de la instrucción, el procedure queda almacenado automáticamente en el esquema de la base de datos.

Una vez creado el procedimiento almacenado listaconsultorios, se puede ejecutar, cada vez que se necesite visualizar los consultorios, la ventaja de llamar (ejecutar) el procedimiento, es que el sistema omite la revisión de la sintaxis SQL, situación que no sucede cuando se tiene el código SQL sin procedimiento almacenado.

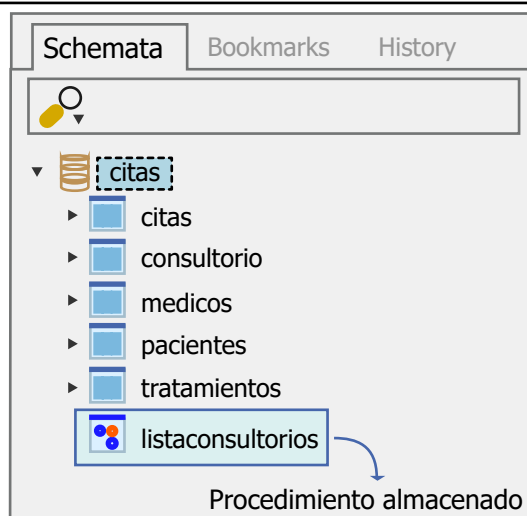


Figura 4. Creación del procedimiento almacenado.

Para llamar el procedimiento almacenado, se utiliza la instrucción:

```
call <nombre_procedimiento>();
```

Para el ejemplo presentado: call listaconsultorios;

La respuesta del sistema presentará el listado de los consultorios incluidos en la tabla “consultorios”.

```
mysql> use citas;
Database changed
mysql> call listaconsultorios();
```

ConNumero	ConNombre
1	Laboratorio Dental
2	Consultorio 1
3	Toma de muestras

```
3 rows in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)

mysql>
```

Figura 5. Ejecución del procedimiento almacenado.

Ahora, para construir un procedimiento que liste el documento de identidad, nombres, apellidos y fecha de nacimiento de los pacientes, registrados en la base de Datos se utilizarían las siguientes sentencias:

Use citas	Se informa la base de datos en la que se va a trabajar.
create procedure listapacientes();	Se informa que se va a crear el procedimiento, Llamado listapacientes.
select pacIdentificacion, PacNombres, PacApellidos, PacFechaNacimiento from pacientes;	Instrucción SQL, para la consulta.

A continuación, se invoca el procedure y el resultado visualizado es:

```
call listapacientes();
```

La respuesta del sistema:

```
mysql>
mysql> call listaconsultorios();
```

pacIdentificacion	ConNombre	PacApellidos	PacFechaNacimiento
1098765678	Carolina	Rojas Zabala	1984-06-28
37821200	Evelia	Arias Mendoza	1970-03-25
37821203	Mari Fernanda	Rodriguez Perez	1970-07-28
63502720	Maria Carolina	Rojas Perez	1980-04-12
63502730	María Alejandra	Torres Cañas	1972-10-15
77191950	Carlos Jose	Arias Rojas	1980-04-12
77191957	Carlos Jose	Arias Rojas	1980-04-12

```
7 rows in set (0.00 sec)

Query OK, 0 rows affected (0.07 sec)
```

Figura 6. Ejecución del procedimiento almacenado.

Para generar el listado de los pacientes ordenados alfabéticamente por apellidos, la sintaxis para el procedure sería:

Use citas	Se informa la base de datos en la que se va a trabajar.
create procedure ordenamientoapellidos();	Se informa que se va a crear el procedimiento, Llamado ordenamientoapellidos.
select pacIdentificacion, PacNombres, PacApellidos, PacFechaNacimiento from pacientes Orderby pacApellidos;	Instrucción SQL, para la consulta.

```
mysql> call ordenamientoapellidos();
```

pacIdentificacion	ConNombre	PacApellidos	PacFechaNacimiento
37821200	Evelia	Arias Mendoza	1970-03-25
77191950	Carlos Jose	Arias Rojas	1980-04-12
77191957	Carlos Jose	Arias Rojas	1980-04-12
37821203	Mari Fernanda	Rodriguez Perez	1970-07-28
63502720	Carolina	Rojas Perez	1980-04-12
1098765678	Maria Carolina	Rojas Zabala	1984-06-28
63502730	María Alejandra	Torres Cañas	1972-10-15

7 rows in set (0.04 sec)

Figura 7. Ejecución del procedimiento almacenado.

Toda instrucción SQL que se construya, puede ser almacenada en un procedure.

2.2. Creación de procedimientos almacenados con parámetros

El parámetro es fundamental, al momento de utilizar criterios de selección en la cláusula where.

En los ejemplos anteriores después del nombre del procedure se incluían unos paréntesis, los cuales estaban vacíos; para los ejemplos de esta sección los paréntesis llevan información, y esta información se convertirá en el parámetro del procedimiento.

Para este proceso, iniciar con listar los datos de todas las pacientes mujeres que están registradas en la entidad pacientes, para este caso el criterio de búsqueda es el campo sexo, que para este ejemplo se llama PacSexo y la condición que es que sea mujer. Es importante aclarar que esta condición es sensible a mayúsculas y minúsculas.

La consulta SQL sería:

```
PacIdentificacion, PacNombres, PacApellidos, PacFechaNacimiento, PacSexo
from pacientes
Where PacSexo='F ';
```

Es una buena práctica especificar los campos que la consulta devuelve, de esa manera se tiene un mejor control de los datos devueltos por la consulta.

Existe otra forma de realizar la consulta:

```
Select *
from pacientes
Where PacSexo='F ';
```

Donde el simbolo * determina que se devuelvan todos los campos de la tabla. Cuando una tabla tiene muchos campos, la opción de select * no es recomendable porque hace que la consulta demore más en su ejecución debido a que trae todos los campos de la tabla. En este caso, es mejor especificar los campos a devolver en la consulta.

Es así que, al aplicar esta instrucción a un procedure, la condición se convierte en un parámetro.

2.2.1. Creación de procedimientos almacenados con parámetros en MySQL

Tomando como referencia la instrucción construida en SQL anteriormente, aplicar este concepto para construir el procedure en MySQL; al momento de construir el procedure se debe tener claro lo siguiente:

```
Create procedure <nombreprocedimiento> ( <nombreparámetro> <tipo_
de_dato_del_parámetro>)
```

Donde, <nombreparámetro> será el nombre que se va a utilizar como variable al momento de construir el where y tipo_de_dato_del_parámetro, será EXACTAMENTE EL MISMO TIPO DE DATO usado en el campo con el que se establece la condición.

En ese orden de ideas, como el campo que se tiene para la consulta es PacSexo y este campo solo recibe F o M, el tipo_de_dato_del_parámetro, es de tipo varchar(1). La sentencia para construir el procedure de este ejemplo es:

Use citas	Se informa la base de datos en la que se va a trabajar.
create procedure listamujeres(vsexvarchar(1));	Vsex, nombre de a variable a utilizar y, varchar(1), donde vsex es la variable utilizada al crear el procedure.
SELECT*FROM pacientes WHERE PacSexo=vsex;	Instrucción SQL, para la consulta, NOTE que PacSexo=vsex, donde vsex es la variable utilizada al crear el procedure.

Para ejecutar el procedure, se utiliza la misma palabra reservada `call` nombre_del_procedimiento y entre paréntesis el argumento a buscar, para este ejemplo la letra F, porque necesitamos listar los pacientes de sexo femenino. La sintaxis quedaría:

`Call listarjueves('F');` Note que dentro del paréntesis va la F, de femenino, y entre comilla sencilla, por ser un campo tipo `varchar`.

Ahora visualizar los nombres, apellidos y documento de identidad de los pacientes que se han realizado como tratamiento, un blanqueamiento dental. Recordemos que los datos básicos de los pacientes se encuentran en la entidad “pacientes”, mientras que el tratamiento se encuentra en la entidad “tratamientos”, la consulta SQL sería de la siguiente manera:

```
select pac.PacIdentificacion, pac.PacNombres, pac.PacApellidos, tra.TraDescripcion
from pacientes pac, tratamientos tra
where pac.pacIdentificacion=tra.TraPaciente and tra.TraDescripcion='Blanqueamiento
Dental';
```

Ahora incluir esta instrucción SQL en un procedure.

Use citas	Se informa la base de datos en la que se va a trabajar.
<code>create procedure listarblanqueamientos(tipo text);</code>	tipo, nombre de la variable a utilizar y, <code>text</code> , porque el campo <code>TraDescripcion</code> en la tabla es de ese tipo, va sin ancho, tal cual está en la tabla.
<code>select pac.PacIdentificacion, pac.Pacnombres, pac.PacApellidos, tra.TraDescripcion from pacientes as pac inner join tratamientos as tra on ac.pacIdentificacion=tra.TraPaciente Where pac.pacIdentificacion=tra.TraPaciente and tra.TraDescripcion=tipo;</code>	Instrucción SQL, para la consulta, NOTE que <code>TraDescripcion=tipo</code> , donde <code>tipo</code> es la variable utilizada al crear el procedure.

La llamada al procedure,

```
call listarblanqueamientos('Blanqueamiento Dental');
```

Note que dentro del paréntesis va Blanqueamiento Dental entre comilla sencilla, por ser un campo tipo `text` y por ser el tipo de tratamiento que deseamos listar.

La respuesta del sistema:

```
mysql> call listarblanqueamientos<'Blanqueamiento Dental'>;
```

pacIdentificacion	ConNombre	PacApellidos	PacFechaNacimiento
77191957	Carlos Jose	Arias Rojas	Blanqueamiento Dental
37821203	Mari Fernanda	Rodriguez Perez	Blanqueamiento Dental
37821203	Mari Fernanda	Rodriguez Perez	Blanqueamiento Dental

```
3 rows in set (0.01 sec)
```

```
Query OK, 0 rows affected (0.05 sec)
```

Figura 8. Ejecución del procedimiento almacenado con parámetros.

También se pueden construir procedimientos almacenados para sentencias de inserción, modificación o eliminación de un registro a la tabla, la instrucción SQL para incluir un registro a la tabla “médicos” sería:

```
insert into medicos(MedIdentificacion, MedNombres, MedApellidos)
values ('63456789', 'Ramón', 'Arenas');
```

La codificación del procedimiento sería de la siguiente manera:

Use citas	Se informa la base de datos en la que se va a trabajar.
create procedure insertarmedico (identificacion char(10), nombre varchar(50), apellidos varchar(50));	
Los argumentos del procedimiento corresponden a los campos de la table, los tipos deben ser iguales a los tipos de datos en la table. OJO deben llamarse diferente a los campos de la tabla.	
insert into medicos(MedIdentificacion, MedNombres, MedApellidos) values (identificacion, nombre, apellidos);	
Instrucción SQL, NOTE que en los valores se referencian los argumentos del procedimiento.	

Para insertar un registro mediante el procedimiento insertarmedico, se realiza la siguiente instrucción:

```
call insertarmedico('234567', 'Carlos', 'Pedraza');
```

Valores a incluir en la tabla

Nombre del procedimiento

2.2.2. Creación de procedimientos almacenados con parámetros tipo IN en Oracle

En Oracle, TODOS los procedimientos almacenados, para manejo de bases de Datos, se manejan con argumentos, la sintaxis para la construcción o modificación de un procedimiento es:

```
CREATE [OR REPLACE]
PROCEDURE <procedure_name> [(<param1> [IN|OUT|IN OUT] <type>,
<param2> [IN|OUT|IN OUT] <type>, ...)]
IS -- Declaración de variables locales
BEGIN
-- Sentencias
[EXCEPTION]
-- Sentencias control de excepcion
END [<procedure_name>];
```

El uso de la palabra reservada OR REPLACE permite sobrescribir un procedimiento existente. Si se omite, y el procedimiento ya existe, el sistema producirá un error.

Inicialmente se va a realizar un procedimiento almacenado con argumentos tipo IN(entrada) para “ingresar” datos a la tabla “médicos”. Recuerde que la instrucción SQL para incluir registros a la tabla médicos es:

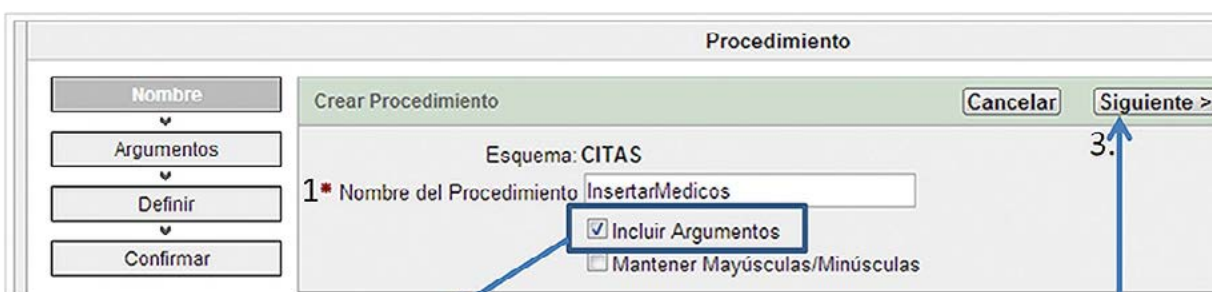
```
insert into medicos(MedIdentificacion, MedNombres, MedApellidos)
values ( '1', 'nombre', 'apellidos');
```

En el entorno de trabajo de Oracle10g, se pueden codificar los procedimientos almacenados, haciendo clic en el icono Explorador de Objetos, opción crear, sub-opción Procedimiento.



Figura 9. Opción Crear Procedimiento en Oracle.

Al hacer clic en la opción procedimiento, se visualiza la siguiente pantalla, para que se asigne el nombre del procedimiento y se continúe con el asistente, haciendo clic en el icono denominado siguiente:



2. Es muy importante activar esta opción para el manejo de los Argumentos (in, out o in out) del procedimiento

3. Después de asignar el nombre, y activar la opción "incluir argumentos, se hace clic en Siguiente

Figura 10. Asistente - Crear Procedimiento en Oracle.

El siguiente paso en el asistente es configurar los argumentos del procedimiento, el número de argumentos depende de la cantidad de campos a manipular en la tabla, en este ejemplo se tienen tres (3) argumentos, porque la tabla médicos tiene tres campos, así:

NOMBRE	TIPO	ANCHO
MedIdentificacion	Char	10
MedNombres	Varchar	50
MedApellidos	Varchar	50

A continuación, la presentación del sistema:

Procedimiento

Nombre

Argumentos

Definir

Confirmar

Nombre

MedIdentificacion

MedNombres

MedApellidos

Tipo

Char

Varchar

Varchar

Ancho

10

50

50

Crear Procedimiento

Cancelar

< Anterior

Siguiente >

Esquema: CITAS

Nombre del Procedimiento: INSERTARMEDICOS

Nombre del Argumento	Entrada/Salida	Tipo de Argumento	Valor por Defecto	Mover
IdentificacionMed	IN	CHAR		▼
NombreMed	IN	VARCHAR2		▼ ▲
ApellidoMed	IN	VARCHAR2		▼ ▲
	IN	VARCHAR2		▼ ▲
	OUT	VARCHAR2		▼ ▲
	IN OUT	VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲

Agregar Argumento

Identifique los argumentos que desea incluir en el procedimiento. Los argumentos son parámetros que se transfieren o devuelven de los procedimientos. Si desea que el argumento sea nulo, escriba Nulo como valor por defecto.

Figura 11. Argumentos - Crear Procedimiento en Oracle.

Una vez se configura los parámetros del procedimiento, se hace clic en el icono de "siguiente"; el asistente presenta la pantalla para que se digite la respectiva instrucción SQL, en este caso con la cláusula insert.

Procedimiento

Nombre: Esquema: CITAS

Argumentos: Nombre del Procedimiento: INSERTARMEDICOS

Definir: * Cuerpo del Procedimiento:

```
insert into medicos (MedIdentificacion, MedNombres, MedApellidos)
values (IdentificacionMed, NombreMed, ApellidoMed);
```

1. Los campos deben llamarse igual y estar en el mismo orden que en la tabla.

2. Estos nombres corresponden a los argumentos creados en el paso anterior y deben estar en el mismo orden que en el insert

Utilice esta página para introducir el bloque PL/SQL que desee utilizar como cuerpo del procedimiento. El cuerpo del procedimiento es todo lo incluido entre BEGIN y END;

Por ejemplo, si define un parámetro IN del tipo VARCHAR2 denominado p_name en el paso anterior, podría introducir lo siguiente para el cuerpo del procedimiento:

```
htp.p('Hola '||p_name);
```

Figura 12. Cuerpo del procedimiento - Crear Procedimiento en Oracle.

Después de digitar la respectiva instrucción SQL y al hacer clic en siguiente, el sistema presenta la información del procedimiento que se acabó de construir para que se dé por terminada la construcción del procedimiento, haciendo clic en el botón terminar. Si es necesario realizar alguna corrección entonces se hace clic en el botón anterior.

Procedimiento

Crear Procedimiento

Esquema: CITAS

Tipo de Objeto: Procedimiento

Objeto: INSERTARMEDICOS

SQL ☐ ☒ SQL

Por defecto aparece este icono, debo hacer clic en él, para visualizar el código SQL, automáticamente el icono cambia

```
create or replace procedure "INSERTARMEDICOS"
(identificacionmed IN CHAR,
nombremed IN VARCHAR2,
apellidomed IN VARCHAR2)
is
begin
insert into medicos (MedIdentificacion, MedNombres, MedApellidos)
values ( IdentificacionMed, NombreMed, ApellidoMed);
end;
```

Figura 13. Confirmar - Crear Procedimiento en Oracle.

Al hacer clic en terminar, automáticamente aparece el procedimiento creado en el panel izquierdo del explorador de objetos, por seguridad se debe verificar si se tiene errores o no; esto se realiza con el icono errores.

Cuando el procedimiento tiene errores aparece con una franja vertical roja antes del nombre de dicho procedimiento.

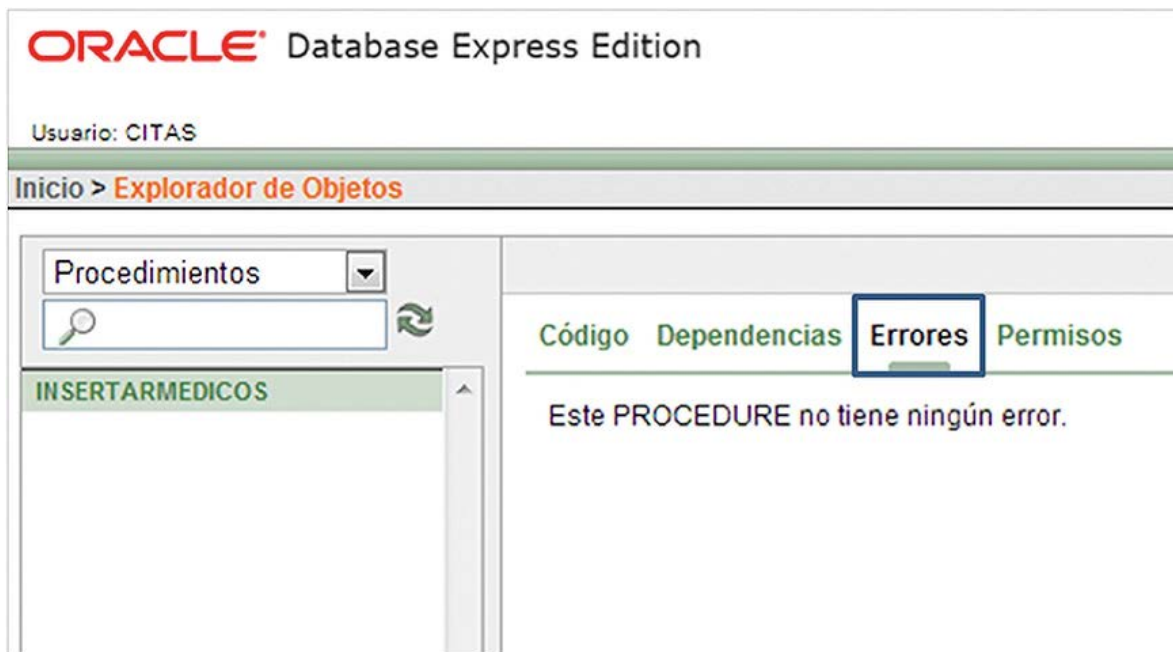


Figura 14. Validación del procedimiento almacenado creado.

2.2.3. Creación de procedimientos almacenados con parámetros tipo OUT en Oracle

Los procedimientos almacenados con argumentos tipo OUT, son los utilizados para generar las salidas del sistema, esto significa que se han trabajado las instrucciones con la cláusula select.

La cláusula select puede ser utilizada de manera simple, es decir sin condiciones, o de manera compuesta, con condiciones mediante la utilización del where. El manejo del select, se ha trabajado en el objeto de aprendizaje anterior denominado “Construir sentencias SQL para la definición y manipulación del modelo de base de datos”; con base en el ejemplo del listado de los consultorios que se realizó en MySQL y ahora trabajarlo desde ORACLE.

Oracle para la presentación de datos, mediante la cláusula select, utiliza el argumento OUT asociado al tipo de datos especial denominado “cursor”, notará que en la instrucción se trabaja una nueva palabra reservada.

“SYS_REFCURSOR”, este tipo de dato es el que permitirá generar los datos que trae la consulta.

En este momento se van a listar los consultorios

NOMBRE	TIPO	ANCHO
ConNumero	Integer	3
ConNombre	Varchar	50

Los pasos a seguir en el asistente para la construcción de procedimientos almacenados en Oracle son:

a. Clic en el icono Explorador de Objetos, opción SQL, donde aparecerá el espacio para digitar la instrucción SQL correspondiente a la creación del procedimiento:

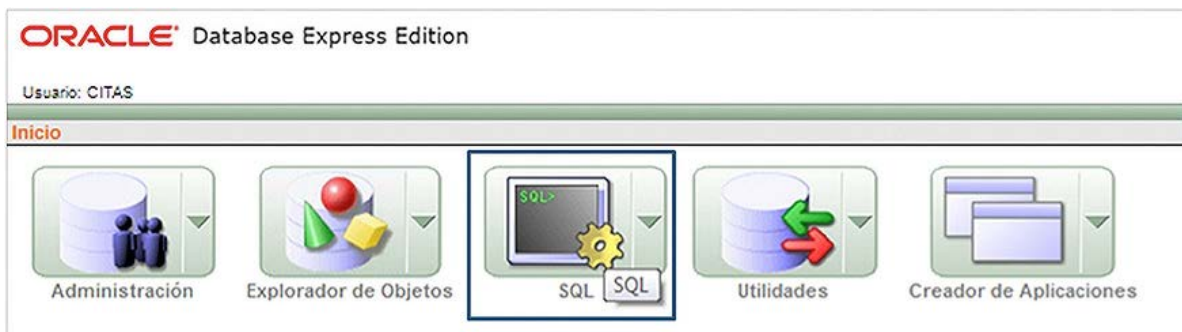
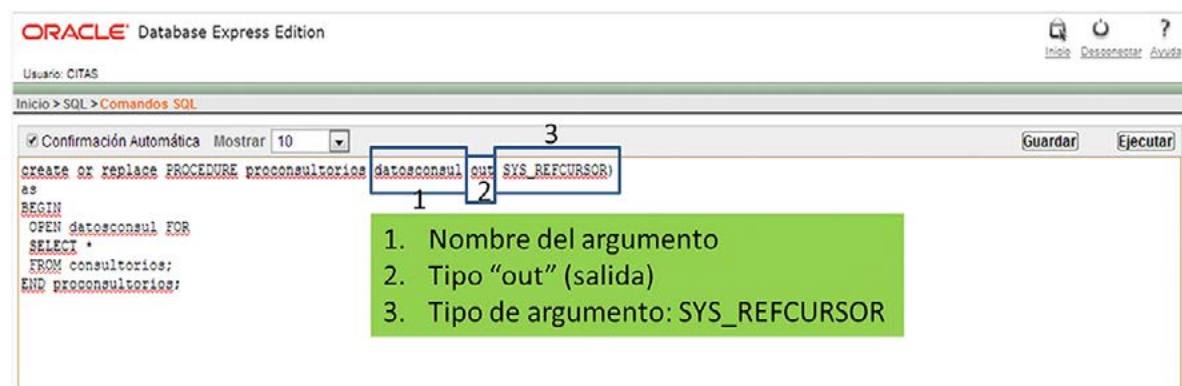


Figura 15. Explorador de objetos de Oracle.

b. Codificación del procedure:



1. IMPORTANTE: Al final de la consulta y del procedure debe estar el respectivo punto y coma

Figura 16. Codificación del procedimiento almacenado.

Importante: Al final de la consulta y del procedimiento almacenado debe estar el respectivo punto y coma.

c. Al final clic en el botón ejecutar para que el procedure quede grabado en el sistema.

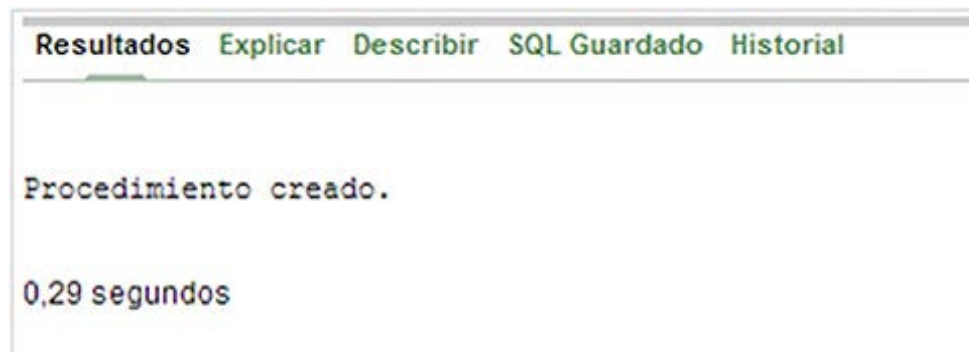


Figura 17. Resultado al crear el procedimiento almacenado.

2.2.4. Creación de procedimientos almacenados con parámetros tipo IN y OUT en Oracle

Listar los pacientes de tipo Femenino.

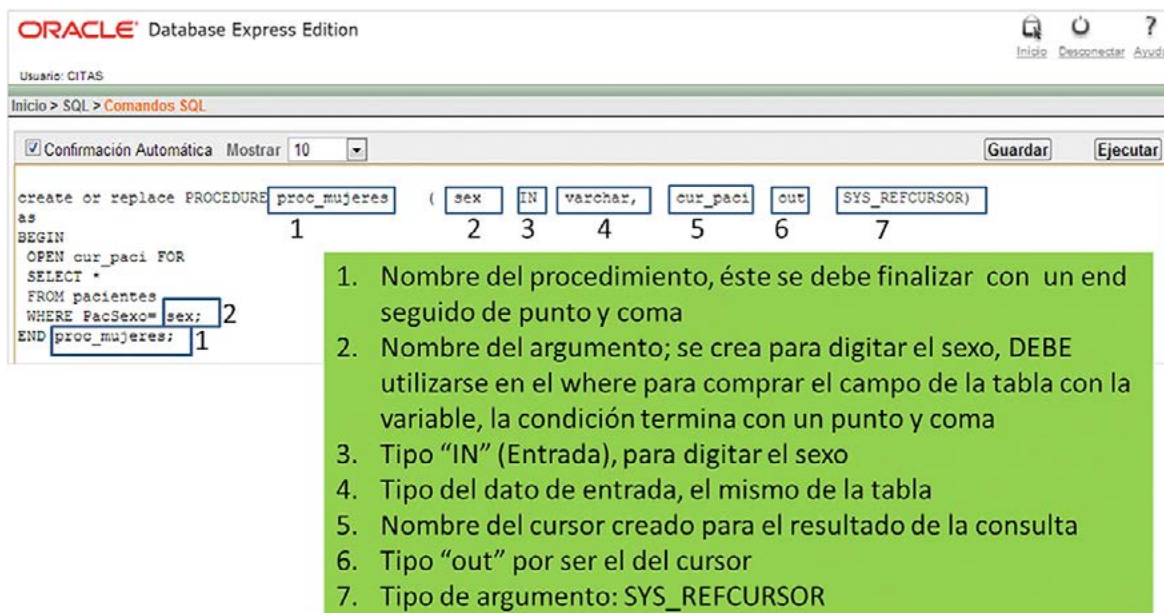
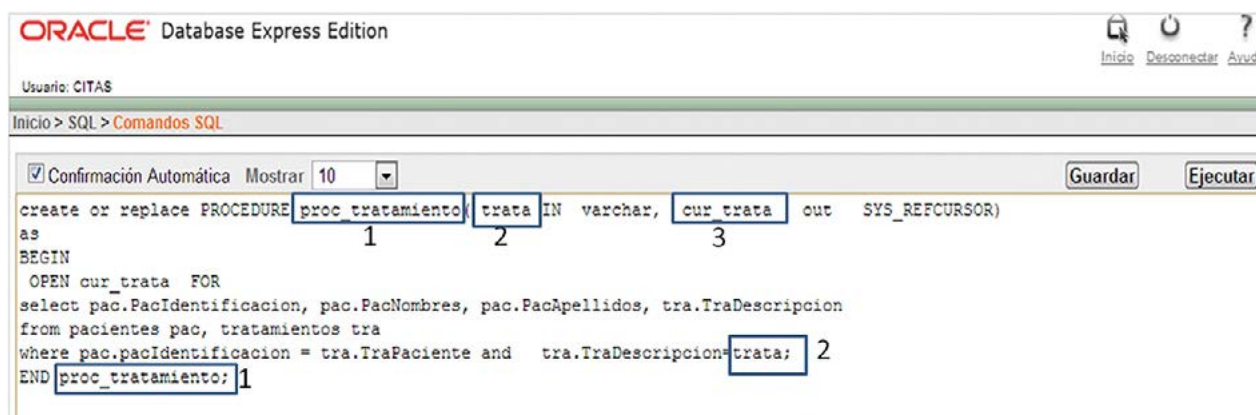


Figura 18. Creación de procedimientos almacenados con parámetros.

Listar los datos de los pacientes que se han realizado como tratamiento un "Blanqueamiento Dental".



1. Nombre del procedimiento, éste se debe finalizar con un end seguido de punto y coma
2. Nombre del argumento; se crea para digitar el nombre del tratamiento, DEBE utilizarse en el where para comprar el campo de la tabla con la variable, la condición termina con un punto y coma
3. Nombre del cursor creado para el resultado de la consulta

Figura 19. Creación de procedimientos almacenados con parámetros.

2.2.5. Creación de procedimientos almacenados sin parámetros en SQL Server

Los siguientes comandos permiten administrar los procedimientos almacenados en SQL Server:

CREATE PROCEDURE	Permite crear un procedimiento almacenado.
ALTER PROCEDURE	Permite editar o modificar un procedimiento almacenado ya existente.
DROP PROCEDURE	Permite eliminar un procedimiento almacenado existente.

En cualquiera de los casos, el usuario de la conexión actual al motor de base de datos debe de tener los permisos suficientes para su ejecución.

En SQL Server se puede utilizar su ambiente gráfico (SQL Server Management Studio o "SSMS") para crear procedimientos almacenados. Este ambiente gráfico, está compuesto básicamente por dos secciones:

En la sección de la izquierda se encuentra:

- Información de conexión del servidor (nombre del servidor, nombre de la instancia y el usuario que está conectado).
- Las bases de datos del servidor.
- Los objetos que componen la base de datos (Diagramas, Tablas, Vistas, Programación, Seguridad, entre otros).
- Los objetos del servidor como (Seguridad, Replicación, Administración, entre otros).

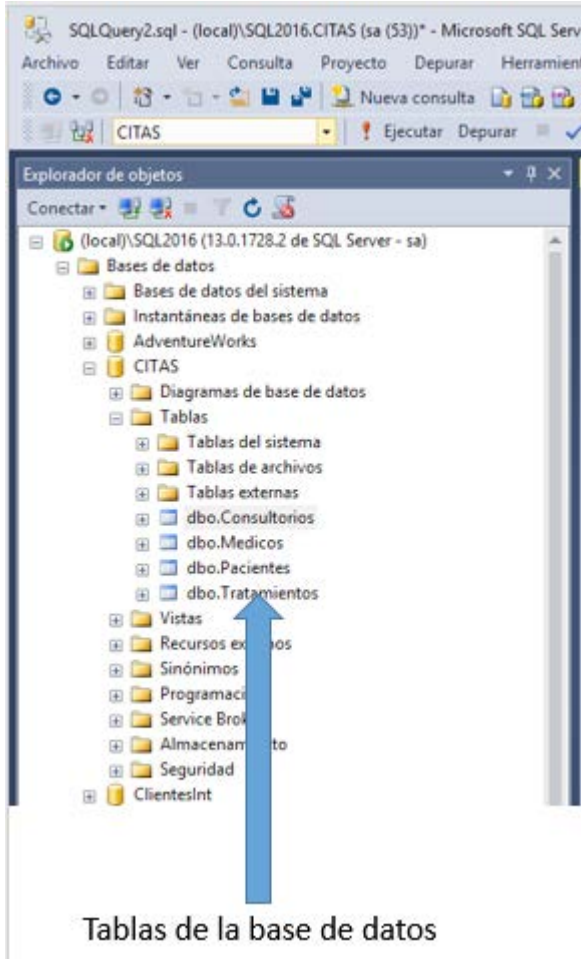


Figura 20. Lista de tablas de la base de datos CITAS.

En la sección de la derecha se encuentra el área de trabajo.

Pasos para crear un nuevo procedimiento almacenado en SSMS:

- Ir a la base de datos, por ejemplo, CITAS.
- Luego Programación – Procedimientos almacenados. El sistema despliega todos los procedimientos almacenados creados para la base de datos.

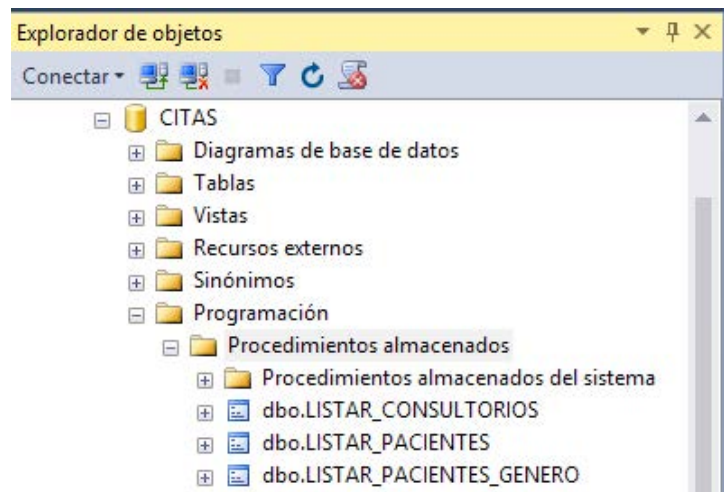


Figura 21. Lista de procedimientos almacenados de la base de datos CITAS.

- Sobre la opción de Procedimientos almacenados, botón derecho, Nuevo procedimiento almacenado. El sistema despliega una plantilla con la sintaxis del comando Create Procedure.

En la siguiente imagen se muestran los objetos que componen un procedimiento almacenado sin parámetros y su respectiva consulta:

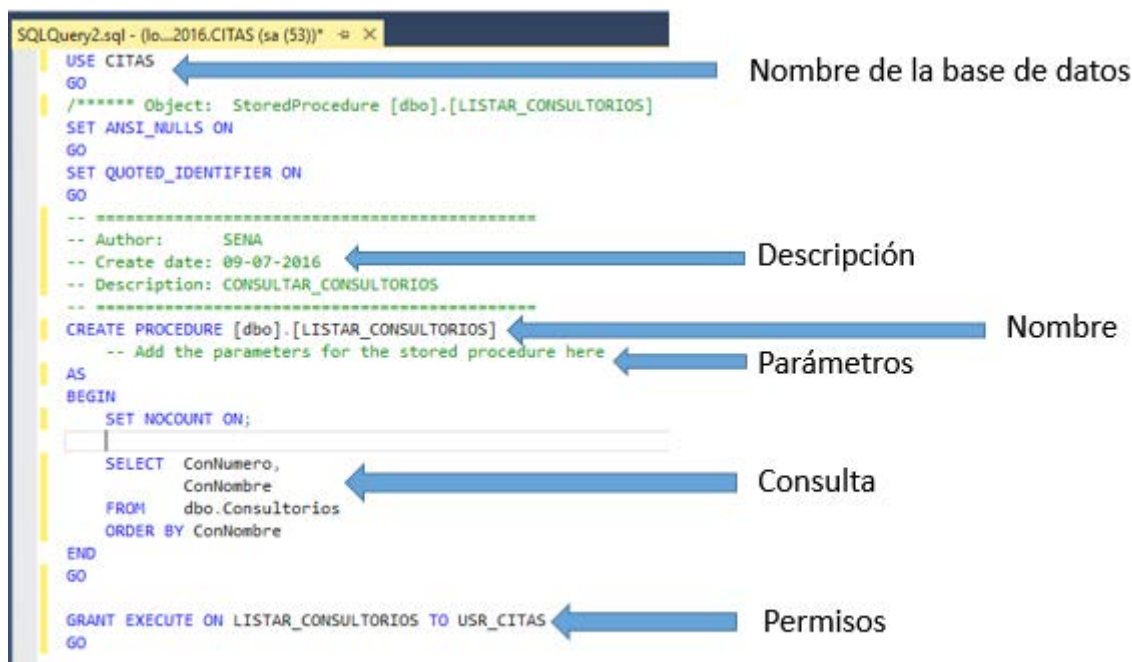


Figura 22. Estructura de un procedimiento almacenado sin parámetros en SQL Server.

Como buenas prácticas se recomienda:

- Generar la consulta y validar su resultado antes de colocarla dentro de un procedimiento almacenado.
- Otorgar permisos de ejecución a un usuario específico de la base de datos
- Evitar nombrar los procedimientos almacenados con el prefijo sp_ debido a que así comienzan los procedimientos almacenados del sistema, evitando conflictos de ejecución.

Una vez se ejecute la consulta de creación del procedimiento almacenado, el motor de la base de datos verifica que no existan errores de sintaxis en sus sentencias o código interno y procede a crear el procedimiento en la base de datos respectiva.

En la próxima sección se mostrará cómo ejecutar un procedimiento almacenado.

2.2.6. Creación de procedimientos almacenados con parámetros en SQL Server

SQL Server permite generar procedimientos almacenados con parámetros de entrada IN y con parámetros de salida OUTPUT.

Los parámetros deben de comenzar con el símbolo @ y deben de tener un tipo de datos válido. Cuando el procedimiento almacenado tiene parámetros, estos deben de ser provistos en el momento de su ejecución. Cuando el parámetro no tiene implícito la palabra

reservada OUTPUT, el motor de base de datos interpreta este parámetro como de entrada (IN).

En el siguiente ejemplo, se utiliza un parámetro de entrada, para un procedimiento almacenado que tendrá como resultado los pacientes de género Masculino 'M' o género Femenino 'F':

Figura 23. Creación de un procedimiento almacenado con parámetros en SQL Server.

```
SQLQuery17.sql - (L...2016.CITAS (sa (66))    SQLQuery16.sql - (L...2016.CITAS (sa (64))*    SQLQuery15.sql - (L...
USE [CITAS]
GO
/***** Object: StoredProcedure [dbo].[LISTAR_PACIENTES_GENERO]    Script Date: 10/07
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:        SENA
-- Create date: 10-07-2016
-- Description: CONSULTAR PACIENTES POR GENERO ORDENADOS POR APELLIDOS
-- =====
CREATE PROCEDURE [dbo].[LISTAR_PACIENTES_GENERO]
    @GENERO CHAR(1)
    -- Add the parameters for the stored procedure here
AS
BEGIN
    SET NOCOUNT ON;


    SELECT PacIdentificacion, PacNombres, PacApellidos, PacFechaNacimiento, PacSexo
    FROM dbo.Pacientes
    WHERE PacSexo = @GENERO
    ORDER BY PacApellidos
END
GRANT EXECUTE ON LISTAR_PACIENTES_GENERO TO USR_CITAS
```

Una vez creado el código del procedimiento almacenado, se puede utilizar los siguientes comandos:



Ejecutar ó F5: valida la sintaxis y si es correcta, procede a crear el procedimiento en la base de datos, generando mensaje “Comandos completados correctamente”.

Depurar: permite recorrer cada una de las sentencias del procedimiento y validar sus resultados.

 : permite validar si existe errores de sintaxis en el código del procedimiento, mostrando el resultado en la ventana de Resultados. Si el resultado es incorrecto, el motor genera mensajes similares al siguiente:

Resultados

```
Mensaje 156, nivel 15, estado 1, procedimiento LISTAR_PACIENTES_GENERO, línea 9 [línea de inicio de lote 7]
Sintaxis incorrecta cerca de la palabra clave 'AS'.
Mensaje 137, nivel 15, estado 2, procedimiento LISTAR_PACIENTES_GENERO, línea 15 [línea de inicio de lote 7]
Debe declarar la variable escalar "@GENERO".
```

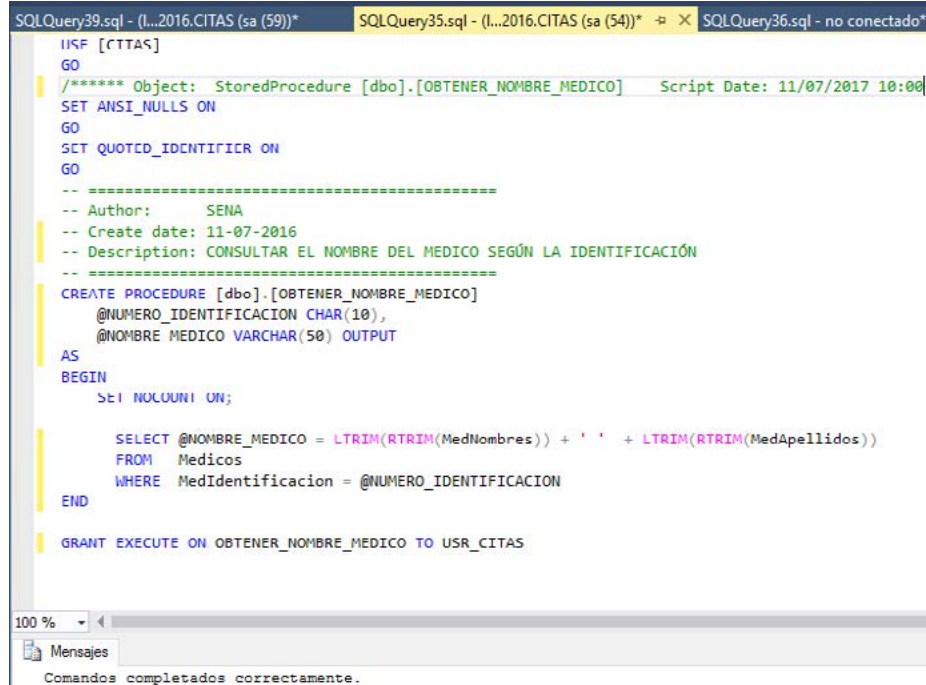
Parámetros de Salida OUTPUT: los procedimientos almacenados en SQL Server, devuelven valores bien sea a través de su valor de retorno, o mediante los parámetros OUTPUT.

La palabra reservada OUTPUT es requerida junto al parámetro cuando el procedimiento es creado y en la ejecución del mismo se debe de especificar junto al parámetro que va a contener el valor devuelto. Dentro del procedimiento el parámetro de salida aparece

como una variable que tendrá algún valor cuando se ejecute el procedimiento.

Los parámetros de salida son útiles para devolver unidades únicas de datos cuando no se requiere un conjunto de registros. La siguiente imagen muestra ejemplo de procedimiento almacenado con parámetro de entrada @NUMERO_IDENTIFICACION y parámetro de salida @NOMBRE_MEDICO.

Figura 24. Creación de un procedimiento almacenado con parámetros OUTPUT en SQL Server.



```

USE [CITAS]
GO

/***** Object: StoredProcedure [dbo].[OBTENER_NOMBRE_MEDICO] Script Date: 11/07/2017 10:00 */
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- Author: SENA
-- Create date: 11-07-2016
-- Description: CONSULTAR EL NOMBRE DEL MEDICO SEGUN LA IDENTIFICACION
--

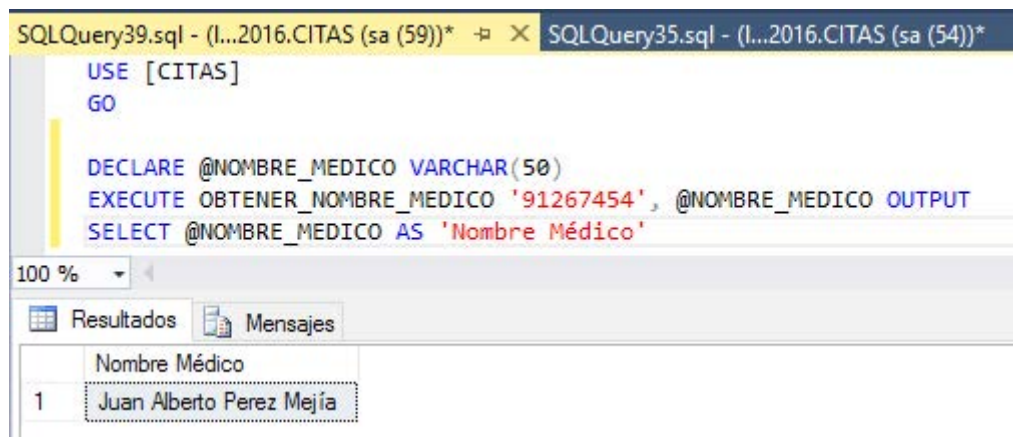
CREATE PROCEDURE [dbo].[OBTENER_NOMBRE_MEDICO]
    @NUMERO_IDENTIFICACION CHAR(10),
    @NOMBRE_MEDICO VARCHAR(50) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT @NOMBRE_MEDICO = LTRIM(RTRIM(MedNombres)) + ' ' + LTRIM(RTRIM(MedApellidos))
    FROM Medicos
    WHERE MedIdentificacion = @NUMERO_IDENTIFICACION
END

GRANT EXECUTE ON OBTENER_NOMBRE_MEDICO TO USR_CITAS
    
```

Mensajes
Comandos completados correctamente.

En la imagen anterior, se creó el procedimiento para consultar el nombre del médico según el número de identificación ingresado, utilizando parámetros de entrada y salida. A continuación, el resultado de la ejecución:



```

USE [CITAS]
GO

DECLARE @NOMBRE_MEDICO VARCHAR(50)
EXECUTE OBTENER_NOMBRE_MEDICO '91267454', @NOMBRE_MEDICO OUTPUT
SELECT @NOMBRE_MEDICO AS 'Nombre Médico'
    
```

Resultados

	Nombre Médico
1	Juan Alberto Perez Mejía

Figura 25. Ejecución de un procedimiento almacenado con parámetros OUTPUT en SQL Server.

Para ejecutar un procedimiento almacenado se puede realizar de dos maneras:

- Directamente en el editor de consultas, con el comando Execute.

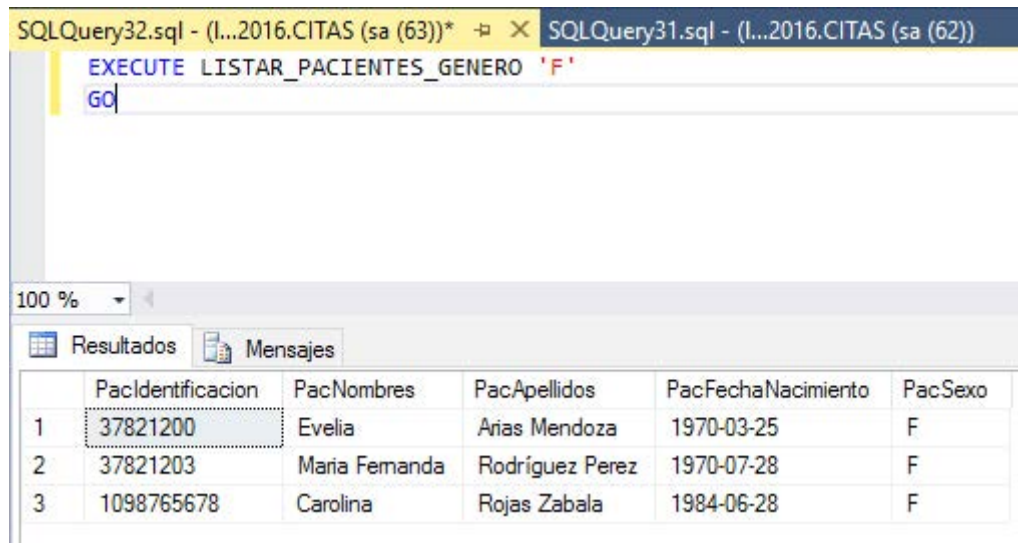


Figura 26. Comando EXECUTE para ejecutar procedimientos almacenados en SQL Server.

En caso de que el procedimiento almacenado tenga varios parámetros, estos se deben ir separados por comas.

- Utilizar el asistente de ejecución. Ubicar el procedimiento almacenado, botón derecho y Ejecutar procedimiento almacenado:

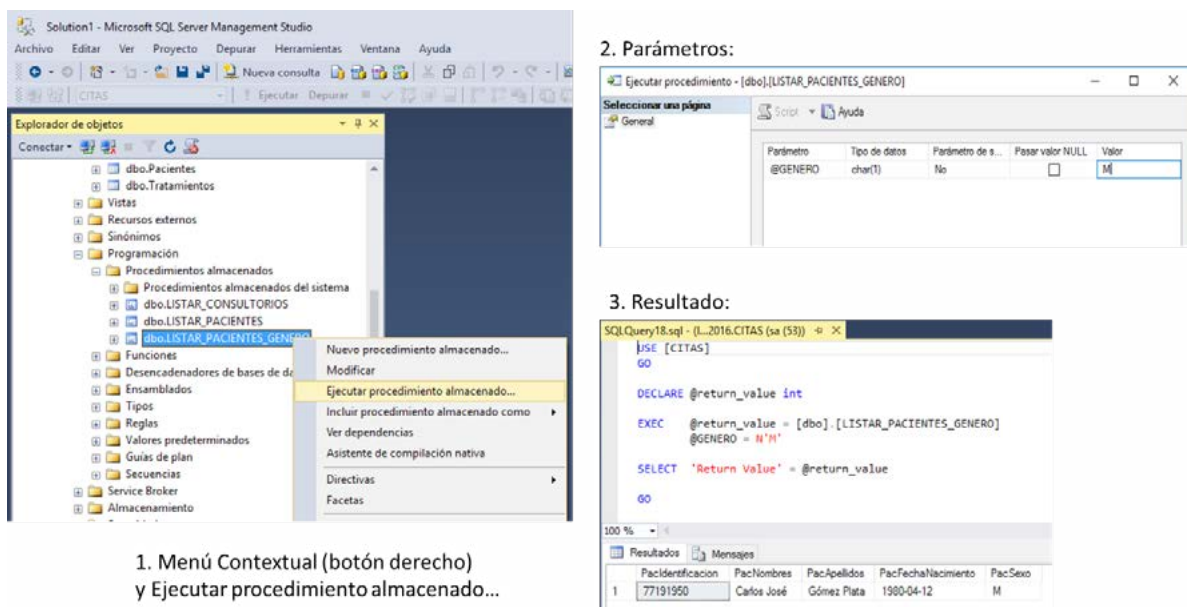


Figura 27. Ejecución de procedimientos almacenados por el asistente en SQL Server.

3. Funciones

3.1. Funciones en MySQL



MySQL, para crear funciones ofrece la directiva `CREATE FUNCTION`, una función se diferencia de un procedimiento, porque devuelve valores.

Estos valores pueden ser utilizados como argumentos para instrucciones SQL, un ejemplo podría ser las funciones `MAX()` o `COUNT()`.

Debido a que en este proceso se devuelven valores, es obligatorio utilizar la cláusula `RETURNS`, al momento de definir una función y sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato).

```
Sintaxis:
delimiter//
CREATE FUNCTION nombre_funcion (parámetro)
Returns tipo
BEGIN
[características] definición
END
//
```

Como ejemplo, se va a construir una función para que cuente los pacientes que están registrados en la tabla `pacientes`, la sintaxis sería la siguiente:

<code>delimiter//</code>	Es muy importante realizar esta definición, porque con él termina la declaración de la función.
<code>CREATE FUNCTION contarpacientes() RETURNS int</code>	Se informa que esta función va a retornar, un valor de tipo numérico (int)
<code>BEGIN</code>	Inicio del bloque de código para la función.
<code>DECLARE cantidad int;</code>	Se declara la variable que se va a retornar, OJO debe ser del mismo tipo de returns.
<code>select count(*) INTO cantidad FROM pacientes;</code>	Instrucción SQL para la consulta, note que se utilizó la palabra reservada <code>INTO</code> para asignar la variable declarada anteriormente.
<code>RETURN cantidad</code>	Se retorna el resultado generado por el count.
<code>END</code>	Fin de la programación de la función.
<code>//</code>	Cierres del delimitador.

3.2. Funciones en Oracle

Una función en Oracle, al igual que en MYSQL, devuelve valores, esa es la diferencia respecto a un procedimiento almacenado.

En esta función se va a contar cuantos pacientes se tienen registrados en la tabla “pacientes”,

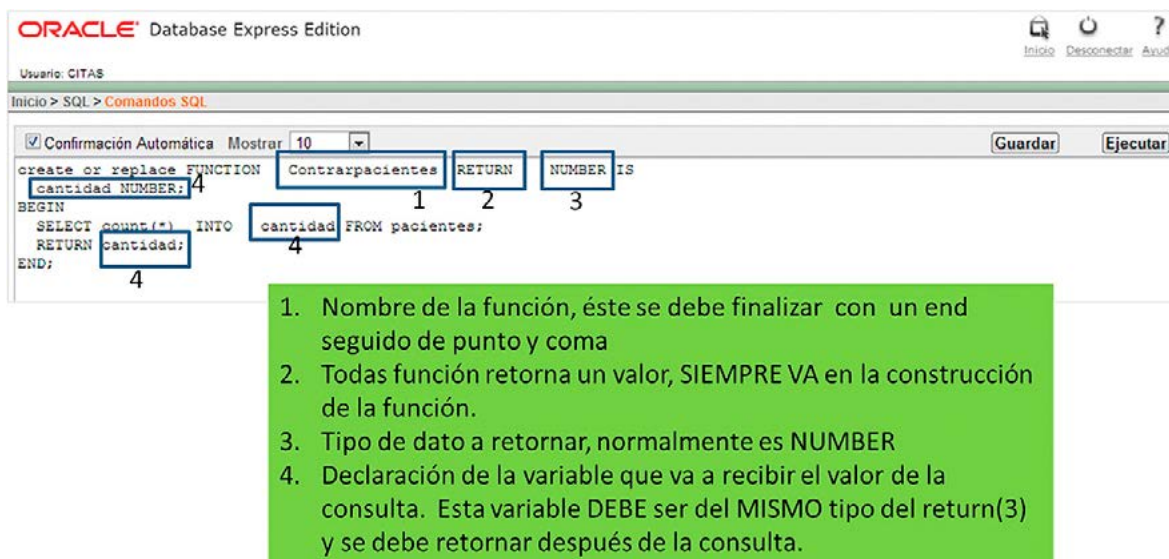


Figura 28. Creación de funciones en Oracle.

Al finalizar la codificación se hace clic en el botón ejecutar, automáticamente el sistema reporta la creación de la función.

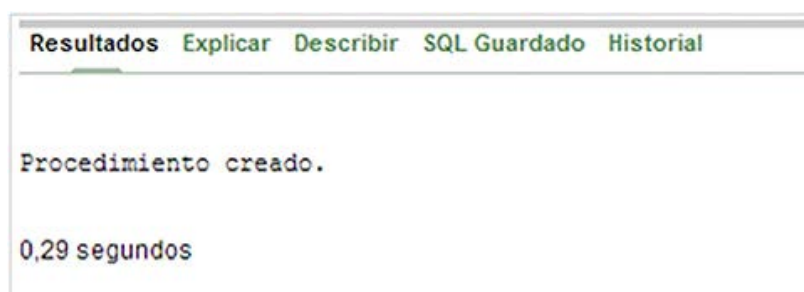
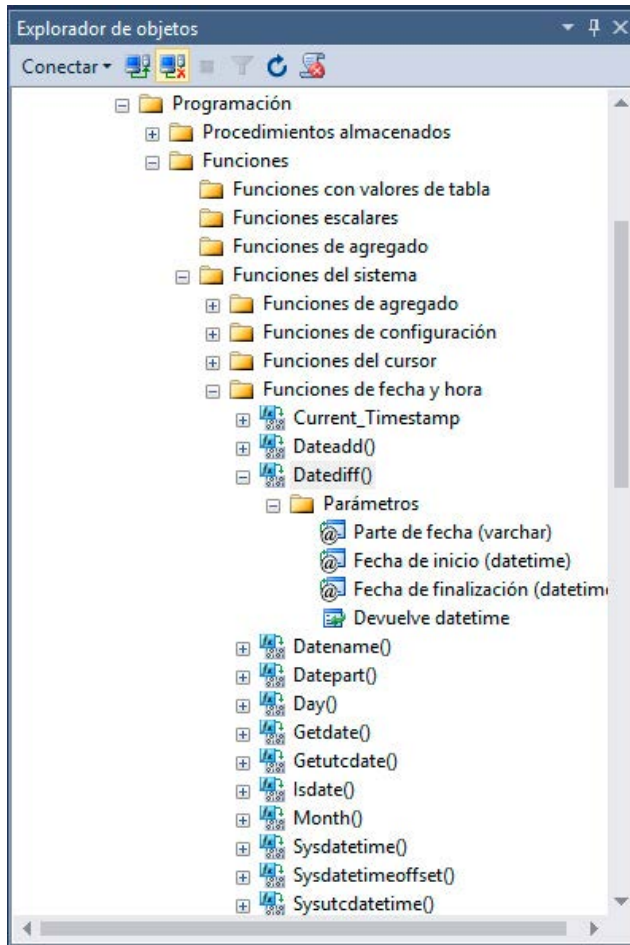


Figura 29. Resultado al crear función en Oracle.

3.3. Funciones en SQL Server

Una función definida por el usuario (User-defined functions “UDF”) es una rutina de Transact-SQL que puede aceptar parámetros, realizar una acción, como un cálculo complejo, y retornar el resultado de esa acción como un valor. El valor devuelto puede ser un valor escalar (único) o una tabla.



Por defecto, SQL Server tiene diferentes tipos de funciones a nivel de funciones del sistema que pueden ser utilizadas en las consultas. Dentro de estas funciones del sistema se tienen Funciones matemáticas, de fecha y hora, de cadenas, de texto e imágenes entre otras.

Por ejemplo, se puede utilizar la función del sistema Datediff para calcular los días entre dos fechas, que recibe tres parámetros y devuelve un valor.

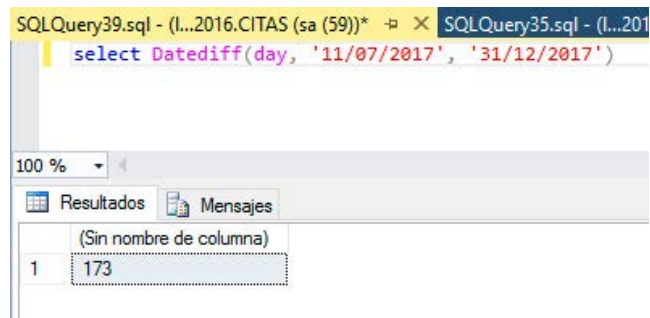


Figura 30. Resultado al ejecutar función del sistema en SQL Server.

Para mayor información de los tipos de funciones del sistema, consultar:

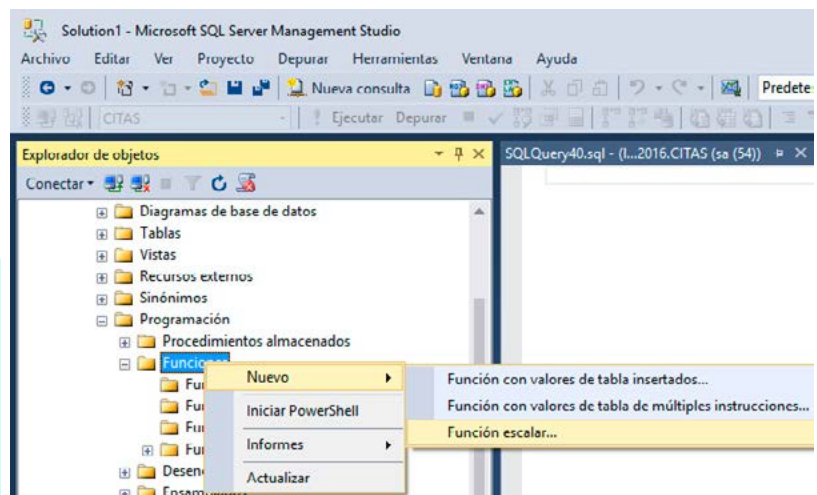
[https://msdn.microsoft.com/es-es/library/ms174318\(v=sql.120\).aspx](https://msdn.microsoft.com/es-es/library/ms174318(v=sql.120).aspx)

En SQL Server al igual que los procedimientos almacenados, se puede utilizar el ambiente gráfico “SSMS” (Sql Server Management Studio) para crear funciones definidas por el usuario.

Opción Programación – Funciones, botón derecho y Nuevo y se selecciona el tipo de función dependiendo del valor a devolver:

Figura 31. Creación de función en SQL Server.

Por ejemplo, siguiendo con la base de datos de CITAS médicas, se necesita consultar la cantidad de pacientes que están en un determinado tratamiento médico. En este caso se va a utilizar una función escalar que devuelve un valor único.



```

USE [CITAS]
GO
/***** Object: UserDefinedFunction [dbo].[FN_CALCULO_PACIENTES_TRATAMIENTO]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
--
-- Author: SENA
-- Create date: 11-07-2017
-- Description: Función que calcula cantidad de pacientes de un tratamiento
--
CREATE FUNCTION [dbo].[FN_CALCULO_PACIENTES_TRATAMIENTO]
(
    @TIPO_TRATAMIENTO VARCHAR(50)
)
RETURNS INT
AS
BEGIN
    DECLARE @CANTIDAD_PACIENTES INT
    SET @CANTIDAD_PACIENTES = 0

    SELECT @CANTIDAD_PACIENTES = COUNT(*)
    FROM TRATAMIENTOS
    WHERE TRATAMIENTO LIKE @TIPO_TRATAMIENTO

    RETURN @CANTIDAD_PACIENTES
END

```

Nombre de la base de datos
 Descripción
 Nombre de la función
 Parámetros
 Variable
 Consulta
 Valor a retornar

Figura 32. Estructura de una función en SQL Server.

Una vez creada la función en el editor, se procede a verificar su sintaxis y a su generación en la base de datos respectiva. Para esto se oprime el botón F5 (Ejecutar) para realizar estas tareas.

En la siguiente imagen se muestra el resultado al ejecutar la función creada anteriormente.

```

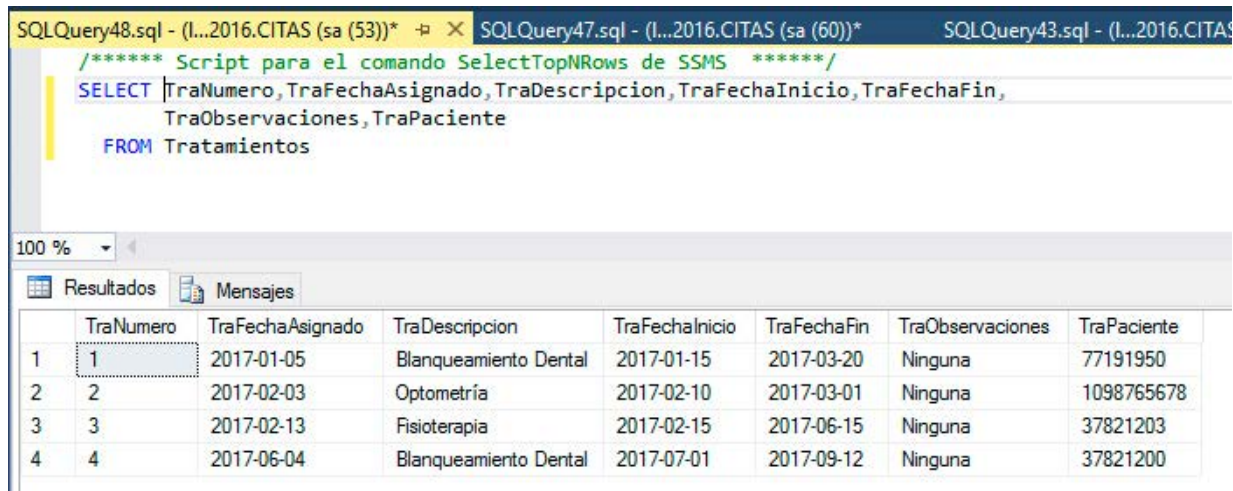
SELECT Cantidad_Pacientes = dbo.FN_CALCULO_PACIENTES_TRATAMIENTO('Blanqueamiento Dental')
GO

```

Cantidad_Pacientes
2

Figura 33. Ejecución de una función en SQL Server.

Al validar el resultado concuerda con los datos almacenados en la tabla de Tratamientos.



	TraNumero	TraFechaAsignado	TraDescripcion	TraFechaInicio	TraFechaFin	TraObservaciones	TraPaciente
1	1	2017-01-05	Blanqueamiento Dental	2017-01-15	2017-03-20	Ninguna	77191950
2	2	2017-02-03	Optometría	2017-02-10	2017-03-01	Ninguna	1098765678
3	3	2017-02-13	Fisioterapia	2017-02-15	2017-06-15	Ninguna	37821203
4	4	2017-06-04	Blanqueamiento Dental	2017-07-01	2017-09-12	Ninguna	37821200

Figura 34. Tabla de Tratamientos.

4. Triggers

Los triggers, también conocidos como disparadores o desencadenadores, son subrutinas que se ejecutan de manera automática, cuando sucede algún evento sobre las tablas asociadas a la base de datos. Los triggers, normalmente son utilizados para realizar auditorías a la base de datos.

Los eventos que activan un trigger pueden ser las sentencias INSERT, DELETE, UPDATE que pueden modificar los datos de una tabla. Los triggers se pueden ejecutar antes (BEFORE) y/o después (AFTER) de que sean modificados los datos.

Los triggers tienen dos palabras clave, OLD y NEW que se refieren a los valores que tienen las columnas antes y después de la modificación. Los INSERT permiten NEW, los DELETE sólo OLD y los UPDATE ambas.

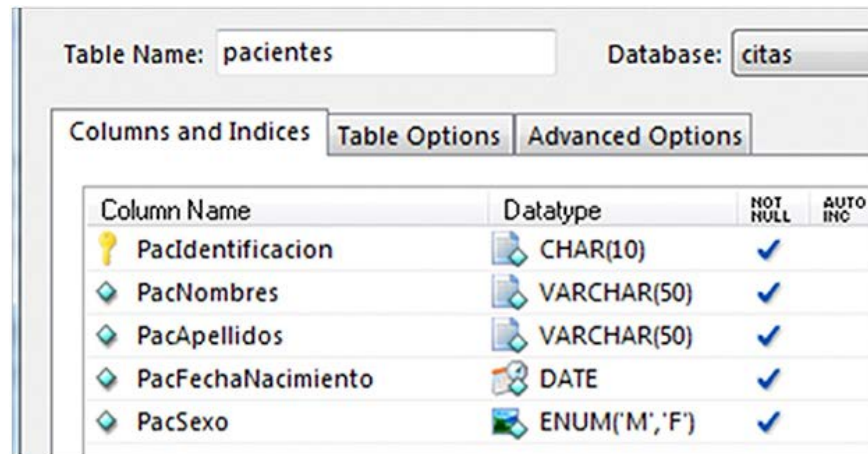
4.1. Triggers en MySQL

Se puede definir un disparador para que se active antes de borrar un registro o después que el registro sea actualizado.

Sintaxis:

```
CREATE TRIGGER <NombreTrigger>
BEFORE AFTER
INSERT UPDATE DELETE
ON
<nombre de la tabla>
FOR EACH ROW
BEGIN
<sentencias SQL>
END;
```

Para trabajar el tema de disparadores desde MySQL, se va a realizar auditoria en la manipulación de los datos para la tabla “pacientes”, esta tabla tiene la siguiente estructura:



Column Name	Datatype	NOT NULL	AUTO INC
PacIdentificacion	CHAR(10)	✓	
PacNombres	VARCHAR(50)	✓	
PacApellidos	VARCHAR(50)	✓	
PacFechaNacimiento	DATE	✓	
PacSexo	ENUM('M','F')	✓	

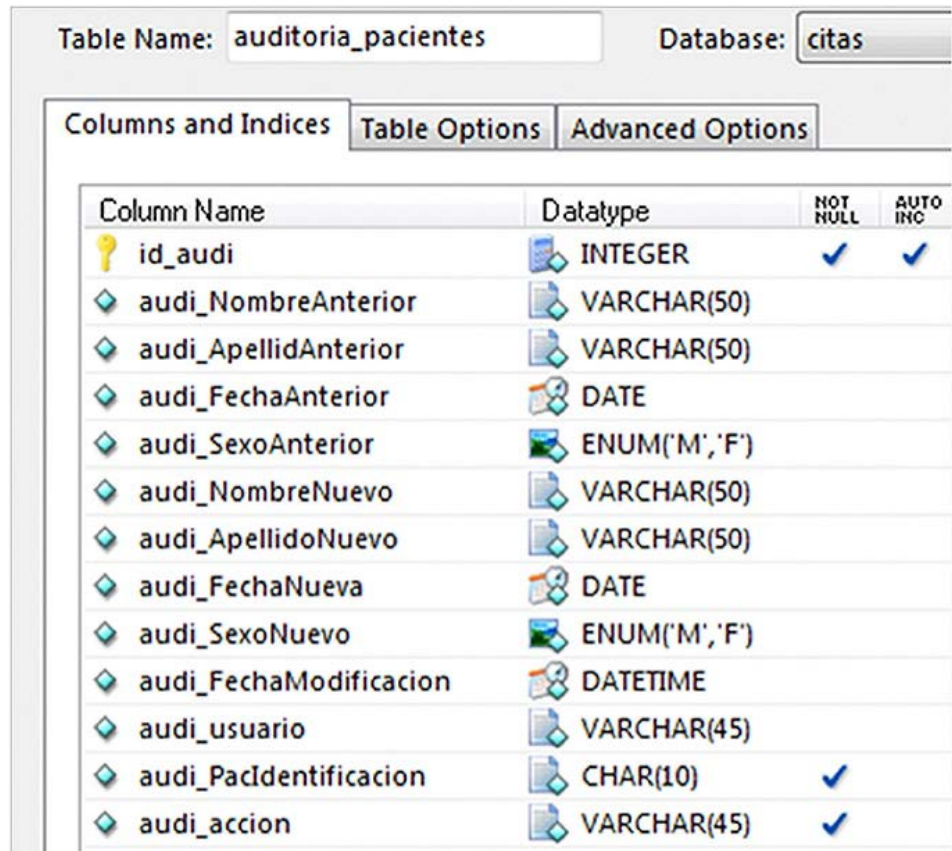
Figura 35. Estructura de la tabla pacientes en MySQL.

El control se va a llevar específicamente en la actualización de los datos (comando update) y en la eliminación de los registros (comando delete); los pasos a seguir son los siguientes:

Paso 1. Crear la tabla auditoria_pacientes: En esta tabla se deben incluir los campos que se tenían inicialmente, los campos que fueron modificados, la fecha de modificación, el usuario que realizó la acción (update o delete), y la acción que se realizó, es así que la tabla quedaría con los siguientes campos.

Figura 36. Estructura de la tabla Auditoria en MySQL.

Es muy importante manejar el orden de los campos y los tipos de los datos, porque de lo contrario se generarán muchos errores de sintaxis, preste atención a los siguientes detalles:



Column Name	Datatype	NOT NULL	AUTO INC
id_audi	INTEGER	✓	✓
audi_NombreAnterior	VARCHAR(50)		
audi_ApellidAnterior	VARCHAR(50)		
audi_FechaAnterior	DATE		
audi_SexoAnterior	ENUM('M','F')		
audi_NombreNuevo	VARCHAR(50)		
audi_ApellidoNuevo	VARCHAR(50)		
audi_FechaNueva	DATE		
audi_SexoNuevo	ENUM('M','F')		
audi_FechaModificacion	DATETIME		
audi_usuario	VARCHAR(45)		
audi_PacIdentificacion	CHAR(10)	✓	
audi_accion	VARCHAR(45)	✓	

Table Name: **auditoria_pacientes** Database: **citas**

Columns and Indices Table Options Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC
id_audi	INTEGER	✓	✓
audi_NombreAnterior	VARCHAR(50)		
audi_ApellidoAnterior	VARCHAR(50)		
audi_FechaAnterior	DATE		
audi_SexoAnterior	ENUM('M','F')		
audi_NombreNuevo	VARCHAR(50)		
audi_ApellidoNuevo	VARCHAR(50)		
audi_FechaNueva	DATE		
audi_SexoNuevo	ENUM('M','F')		
audi_FechaModificacion	DATETIME		now()
audi_usuario	VARCHAR(45)		current_user()
audi_PacIdentificacion	CHAR(10)	✓	
audi_accion	VARCHAR(45)	✓	

Figura 37. Campos tabla Auditoria y Pacientes en MySQL.

- Al momento de realizar una actualización de datos, mediante la instrucción `update`, estos campos guardarán la información que se tenía inicialmente. Note que están en estricto orden respecto a la tabla asociada a la auditoria y se manejan los mismos tipos de datos.
- Al momento de realizar una actualización de datos, mediante la instrucción `update`, estos campos guardarán la información que se modificó al utilizar el `update`. Note que están en estricto orden respecto a la tabla asociada a la auditoria y se manejan los mismos tipos de datos. Si lo que se realizan es el borrado de datos mediante la instrucción `delete` estos campos estarán vacíos y aparecerán con la palabra `null`.
- La información de estos campos se toma directamente del sistema mediante funciones ya predefinidas como se puede observar en el gráfico.
- Aunque es la llave principal (primary key) en la tabla “pacientes”, se coloca al final de todos los campos que se van a auditar, porque NUNCA se modifica un primary key, y es el último argumento al momento de trabajar el código SQL para el `update`.

Paso 2. Crear los triggers para almacenar los cambios generados a la tabla pacientes, específicamente registros actualizados y eliminados.

Paso 2.1. Trigger para auditar registros actualizados: El objetivo de este trigger es llevar el control de los cambios realizados a los datos de los pacientes ya incluidos en la tabla “pacientes”. La instrucción es la siguiente:

Create trigger auditoria_Modificacionpacientes before update	Nombre del trigger y el evento disparador.
for each row	Cada vez que se ejecute el update.
insert into auditoria_pacientes	Tabla creada para la auditoría.
(audi_NombreAnterior, audi_ApellidAnterior, audi_FechaAnterior, audi_SexoAnterior, audi_NombreNuevo, audi_ApellidoNuevo, audi_FechaNueva, audi_SexoNuevo, audi_FechaModificacion, audi_usuario, audi_PacIdentificacion, audi_accion)	Antes de ejecutar el comando Update de SQL. También se puede utilizar el after (después) del update. Lo ideal es lanzar el disparador antes de ejecutar la instrucción para asegurar que solo realicen las actualizaciones de datos permitidas.
Values (old.PacNombres, old.PacApellidos, old.PacFechaNacimiento, oldPacSexo, new.PacNombre, new.PacApellidos, new.PacFechaNacimiento, newPacSexo, now(), current_user(), new:pacIdentificacion, 'Actualización');	Todos los campos de la tabla auditada, "pacientes" llevan la palabra old, para conservar los datos iniciales. Todos los campos de la tabla auditada, "pacientes" llevan la palabra new, para guardar los nuevos datos.

Ahora a verificar la función del trigger, como se configuró con la opción de actualización, se va a modificar un registro con la instrucción update, la sintaxis podría ser:

```
update pacientes set PacNombres='Josexx', PacApellidos='Rozo Torres', PacFechaNacimiento='1980-04-04', PacSexo='M' where PacIdentificacion='77191950';
```

Paso 2.2. Trigger para auditar registros eliminados: Una vez registrado un paciente, es muy importante llevar el histórico de estos, es por ello, que, si se requiere eliminar un registro, se deben dejar almacenados en la tabla de históricos, los datos que se eliminaron con el respectivo usuario y la fecha en la que se realizó el proceso; esa es la función de la tabla de Auditorias.

Create trigger auditoria_Modificacionpacientes after delete on paciente	Nombre del trigger y evento disparador, donde paciente es la tabla a la que se le está haciendo auditoría.
for each row	Cada vez que se ejecute el update.
Insert into auditoria_pacientes	Tabla creada para la auditoria.

(audi_NombreAnterior, audi_ApellidAnterior, audi_FechaAnterior, audi_SexoAnterior, audi_NombreNuevo, audi_ApellidoNuevo, audi_FechaNueva, audi_SexoNuevo, audi_FechaModificacion, audi_usuario, audi_PacIdentificacion, audi_accion)	Campos en el estricto orden de la tabla de auditorias.
Values (old.PacNombres, old. PacApellidos, old.PacFechaNacimiento, oldPacSexo, now(), current_user(), old. PacIdentificacion, 'Registro Eliminado');	Todos los campos de la tabla auditada, "pacientes" llevan la palabra old, para conservar los datos que fueron eliminados.

Values en los Triggers:

Note que los values en El disparador, “Modifica_auditoria_pacientes”, maneja las palabras reservadas “old” y “new”; porque en este proceso es importante dejar la evidencia de los valores iniciales y los valores anteriores.

Los campos que contengan old.nombre_campo, son los encargados de conservar los datos a modificar, y los que contengan new.nombre_campo, son los que tomarán los nuevos datos.

Por ejemplo, Un paciente informa que su nombre está mal escrito y solicita que sea cambiado su nombre “C@rlos Eurelio”, por el correcto, “Carlos Aurelio”. El campo que se manejaría con el old, por ser el dato inicial, sería “C@rlos Eurelio”, y el que se manejaría con el new sería “Carlos Aurelio”.

También puede identificar que en el trigger “auditoria_pacientesEliminados”, solo se maneja la palabra reservada old, porque en este disparador no se modifica ningún dato, solo se eliminan registros.

Puede suceder que por alguna situación se necesita borrar un trigger, la sintaxis para eliminarlos es:

```
drop trigger <Nombre_del_trigger >
```


4.2. Trigger en Oracle

Para crear un disparador en Oracle, se debe crear, al igual que en MYSQL, una tabla para manejar el histórico de las transacciones realizadas, la tabla que se va a crear es la siguiente y los campos deben manejar los tipos de datos acorde con la tabla a auditar:

AUDITORIA_PACIENTES

Tabla	Datos	Índices	Modelo	Restricciones	Permisos	Estadísticas	Valores por Defecto
<div><div>Agregar Columna</div><div>Modificar Columna</div><div>Cambiar Nombre de Columna</div><div>Borrar Columna</div><div>Cam</div></div>							
Nombre De Columna	Tipo De Dato	Nulo	Valor Por Defecto	Clave Primaria			
ID_AUDI	NUMBER	No	-	1			
AUDI_NOMBREANTERIOR	VARCHAR2(50)	Yes	-	-			
AUDI_APELLIDOANTERIOR	VARCHAR2(50)	Yes	-	-			
AUDI_FECHAANTERIOR	DATE	Yes	-	-			
AUDI_SEXOANTERIOR	VARCHAR2(1)	Yes	-	-			
AUDI_NOMBRENUEVO	VARCHAR2(50)	Yes	-	-			
AUDI_APELLIDONUEVO	VARCHAR2(50)	Yes	-	-			
AUDI_FECHANUEVA	DATE	Yes	-	-			
AUDI_SEXONUEVO	VARCHAR2(1)	Yes	-	-			
AUDI_FECHAMODI	DATE	Yes	-	-			
AUDI_USUARIO	VARCHAR2(45)	Yes	-	-			
AUDI_ACCION	VARCHAR2(45)	Yes	-	-			
AUDI_PACIDENTIFICACION	CHAR(10)	Yes	-	-			
1 - 13							

Figura 38. Estructura tabla Auditoria en Oracle.

PACIENTES

Tabla	Datos	Índices	Modelo	Restricciones	Permisos	Estadísticas	Valores por Defecto
<div>Agregar Columna</div> <div>Modificar Columna</div> <div>Cambiar Nombre de Columna</div> <div>Borrar Columna</div> <div>Cam</div>							
Nombre De Columna	Tipo De Dato	Nulo	Valor Por Defecto	Clave Primaria			
PACIDENTIFICACION	CHAR(10)	No	-	1			
PACNOMBRES	VARCHAR2(50)	Yes	-	-			
PACAPELLIDOS	VARCHAR2(50)	Yes	-	-			
PACFECHANACIMIENTO	DATE	Yes	-	-			
PACSEXO	CHAR(1)	Yes	-	-			
					1 - 5		

Figura 39. Estructura tabla Pacientes en Oracle.

Luego de construir la tabla a auditar, y la del histórico de la auditoria, se procede a configurar el disparador, para este proceso, se ubica sobre el nombre de la tabla a auditar, para este caso (pacientes), y se busca sobre las opciones de la tabla, el icono “disparadores”.



Figura 40. Lista de triggers de la tabla Pacientes.

Luego se hace clic en el botón crear, a continuación, cinco aspectos a tener en cuenta para codificar el disparador.

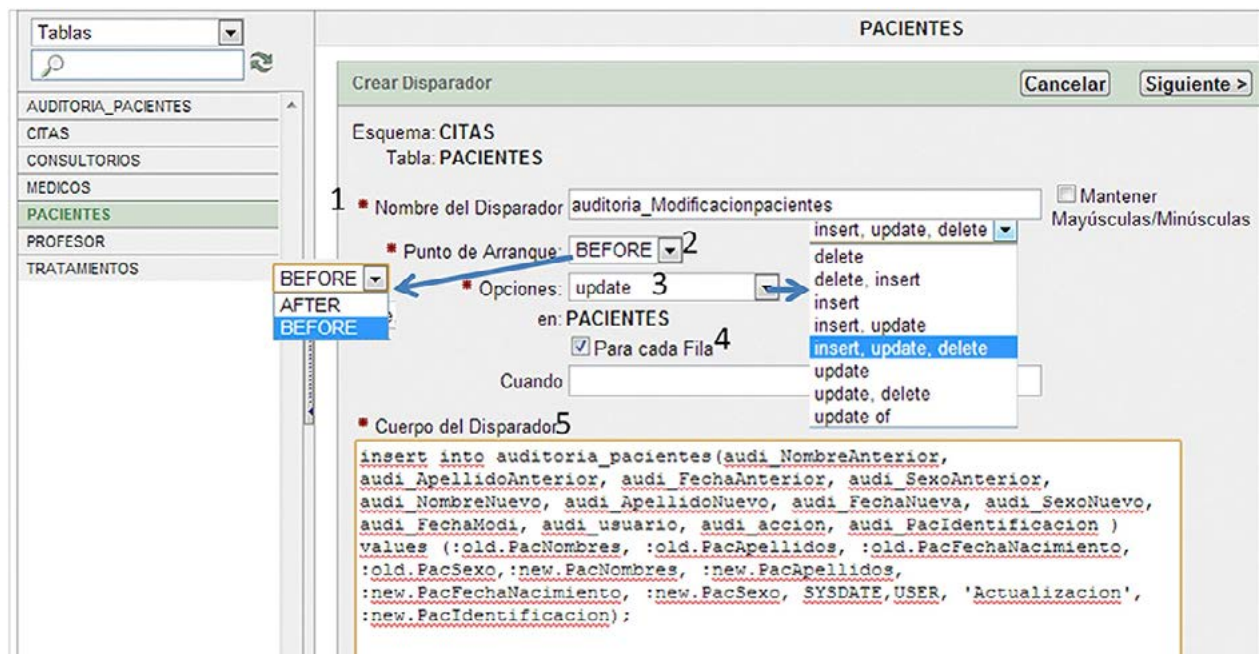


Figura 41. Creación de trigger en la tabla Pacientes.

1. El nombre normalmente se convierte a mayúsculas, excepto si se activa la opción de Mantener Mayúsculas/Minúsculas.
2. El punto de arranque de un disparador, es Antes o después de ejecutar una instrucción SQL.

3. Las opciones corresponden a la instrucción SQL a ejecutar, como se puede observar, Oracle permite todas las cláusulas SQL para manipulación de las tablas.
4. Es muy importante que esté activa la opción “para cada fila”, porque esto nos permite, que cada vez que se ejecute la instrucción SQL, se active el disparador.
5. El código utilizado en el cuerpo del disparador debe contener la instrucción para insertar el respectivo registro en la tabla creada para la auditoria, para nuestro ejemplo “auditoria_pacientes”, con los campos en el respectivo orden para guardar el histórico.

Recuerde que existen las palabras reservadas old (información actual, note que antes de la palabra old van dos puntos) y new (para la nueva captura, igual antes de la palabra new, van los dos puntos).

El sistema nos presenta el código construido, para analizar el código construido, se puede hacer clic en terminar o en anterior, en ocasiones el código SQL puede estar bien, pero pueden suceder errores como el siguiente:

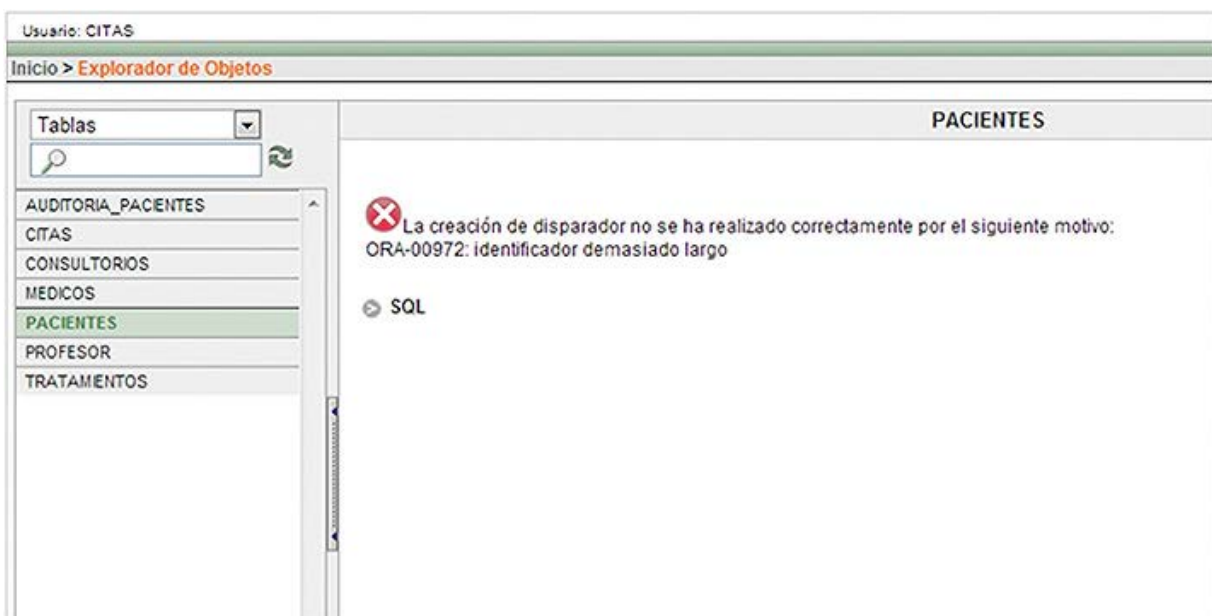


Figura 42. Ventana de errores en Oracle.

Es así como se debe volver a realizar el proceso, teniendo la precaución de manejar nombres menos largos.

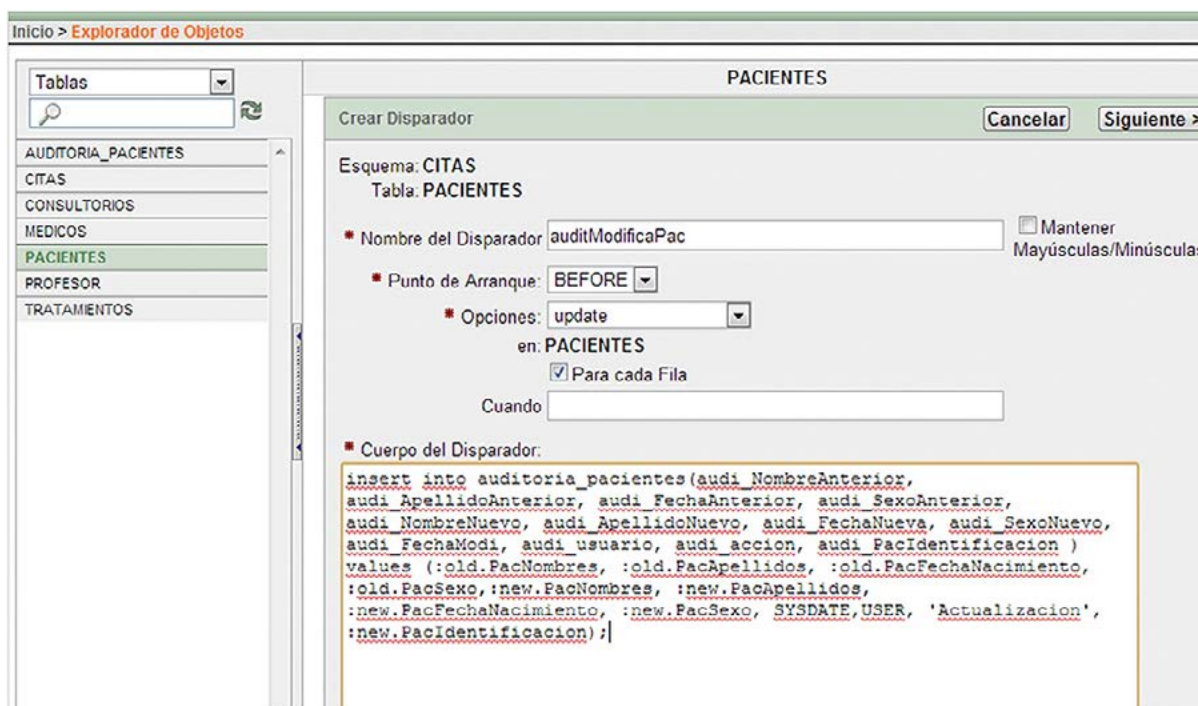


Figura 43. Creación de trigger en la tabla Pacientes.

Clic en siguiente y luego terminar, automáticamente se visualiza la siguiente presentación para activar el respectivo disparador.



Figura 44. Resultado de Creación de trigger en la tabla Pacientes.

Se debe hacer clic en el botón activar, para que el asistente nos permita activar el respectivo disparador, probablemente aparezca más de un disparador para activar.

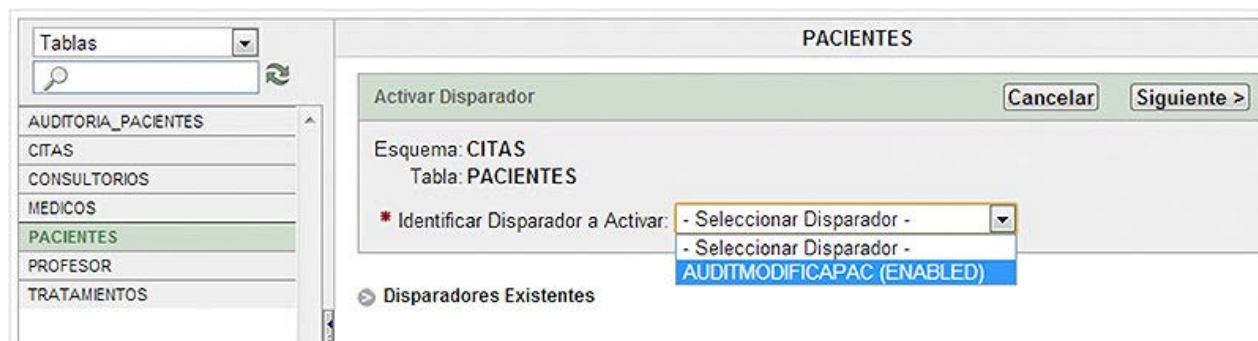


Figura 45. Activación de trigger en la tabla Pacientes.

Una vez seleccionado el disparador, se hace clic en el botón siguiente.

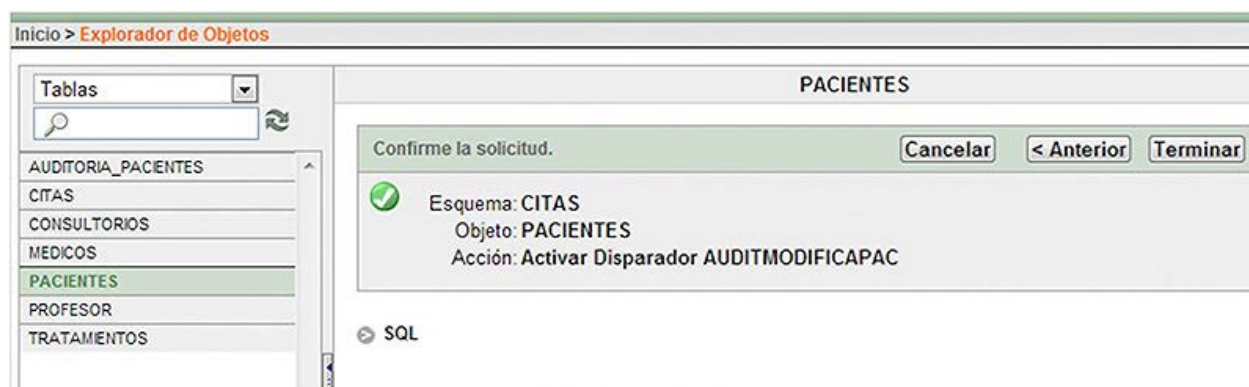
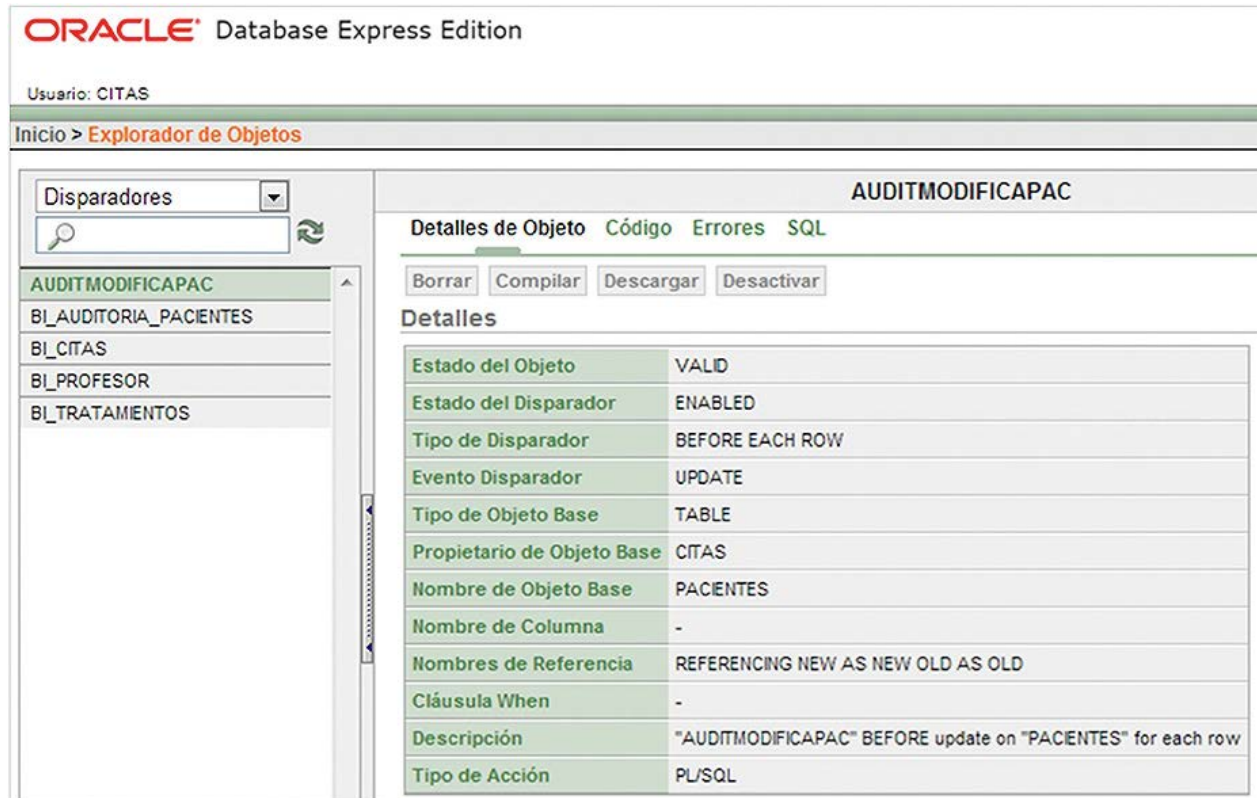


Figura 46. Confirmar la Activación de trigger en la tabla Pacientes.

Y para terminar el proceso de activación del disparador se hace clic en el botón terminar, si se logró activar correctamente el disparador a la tabla aparecerá la siguiente presentación.



Oracle Database Express Edition

Usuario: CITAS

Inicio > Explorador de Objetos

Disparadores

AUDITMODIFICAPAC

BI_AUDITORIA_PACIENTES

BI_CITAS

BI_PROFESOR

BI_TRATAMIENTOS

Detalles de Objeto Código Errores SQL

Borrar Compilar Descargar Desactivar

Detalles

Estado del Objeto	VALID
Estado del Disparador	ENABLED
Tipo de Disparador	BEFORE EACH ROW
Evento Disparador	UPDATE
Tipo de Objeto Base	TABLE
Propietario de Objeto Base	CITAS
Nombre de Objeto Base	PACIENTES
Nombre de Columna	-
Nombres de Referencia	REFERENCING NEW AS NEW OLD AS OLD
Cláusula When	-
Descripción	"AUDITMODIFICAPAC" BEFORE update on "PACIENTES" for each row
Tipo de Acción	PL/SQL

Figura 47. Resultado de la Activación de trigger en la tabla Pacientes.

Columnas

Columna	Tabla	Propietario De Tabla	Uso
PACIDENTIFICACION	<u>PACIENTES</u>	CITAS	New
PACSEXO	<u>PACIENTES</u>	CITAS	-
PACFECHANACIMIENTO	<u>PACIENTES</u>	CITAS	-
PACNOMBRES	<u>PACIENTES</u>	CITAS	-
PACPELLIDOS	<u>PACIENTES</u>	CITAS	-
1 - 5			

Figura 48. Estructura de la tabla Pacientes.

Por seguridad es importante verificar el botón de errores, para estar completamente seguros que el proceso quedó correctamente.

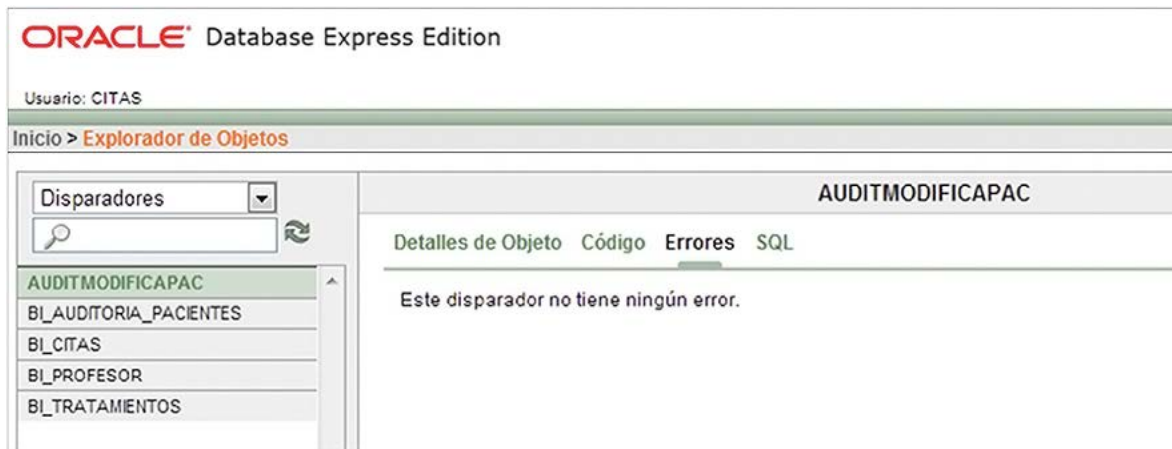


Figura 49. Validación de errores del trigger de la tabla Pacientes.

Ahora para verificar que el disparador está funcionando y que se están enviando los datos a la tabla auditoria_pacientes; en el editor de comandos de SQL, al cual se llega haciendo clic en el icono SQL.



Figura 50. Explorador de objetos de Oracle.

Se va a digitar la siguiente instrucción SQL:

```
update pacientes set PacNombres='Josexx', PacApellidos='Rozo
Torres', PacFechaNacimiento='', PacSexo='M' where
PacIdentificacion='37821203';
```

Esta identificación debe ser de un registro que esté incluido en la tabla pacientes. Y para verificar que la información que se acabó de modificar quedo en la tabla auditoria_pacientes, se utiliza la siguiente instrucción:

```
select * from auditoria_pacientes;
```


Para eliminar un disparador, el procedimiento es bastante sencillo, solo es cuestión de ubicarse en el disparador, luego hacer clic en la opción “detalles de Objeto” y clic en borrar.

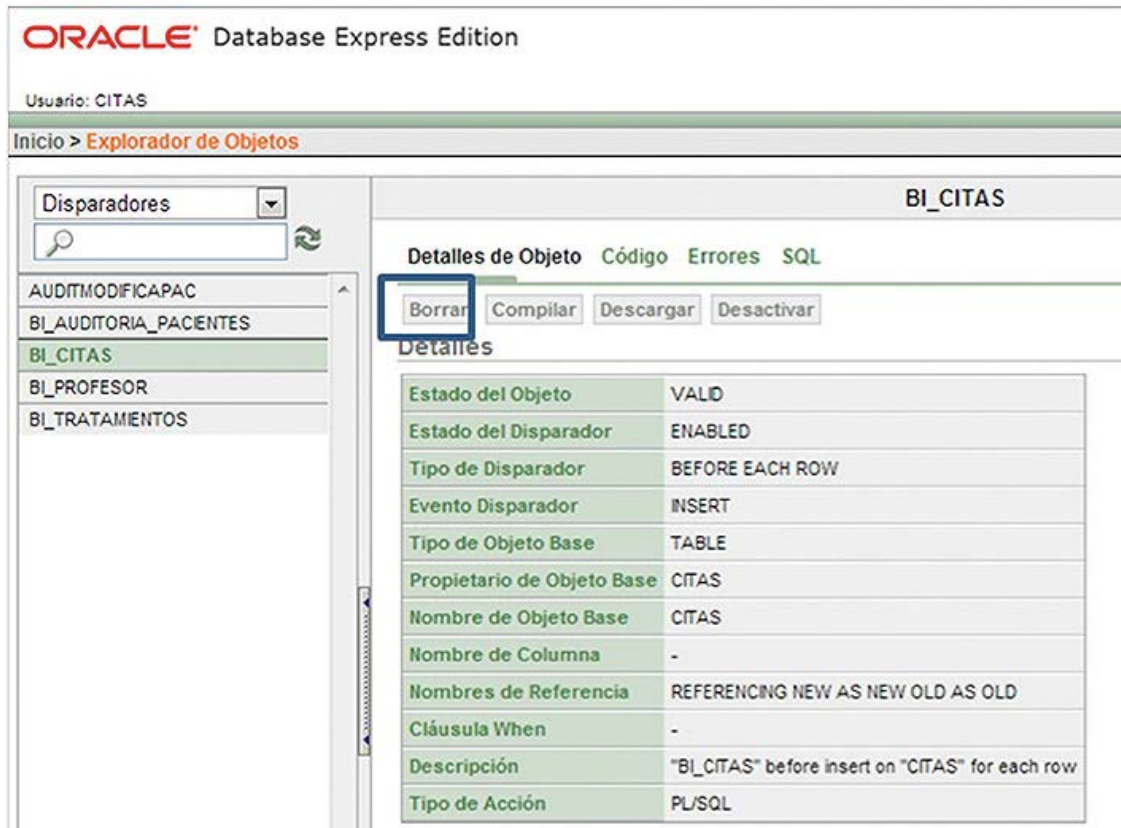


Figura 51. Proceso de eliminación de trigger en Oracle.

Automáticamente el sistema presenta el pantallazo para la confirmación de la eliminación del disparador.



Figura 52. Confirmación de eliminación de trigger en Oracle.

4.3. Trigger en SQL Server

Los triggers o desencadenadores son una versión de procedimientos almacenados que se ejecutan según eventos de una tabla, por lo tanto, ellos no pueden ser directamente ejecutados, si no que responden a eventos de los comandos INSERT (insertar), UPDATE (actualizar), DELETE (eliminar). Los triggers son ejecutados una vez por transacción no por cada registro afectado.

Los siguientes comandos permiten administrar los Triggers os en SQL Server:

CREATE TRIGGER	Permite crear un nuevo desencadenador.
ALTER TRIGGER	Permite editar o modificar un desencadenador ya existente.
DROP TRIGGER	Permite eliminar un desencadenador existente.

En cualquiera de los casos, el usuario de la conexión actual al motor de base de datos debe de tener los permisos suficientes para su ejecución.

Aspectos a tener en cuenta al trabajar con triggers:

- No reciben ni retornan parámetros.
- No se pueden invocar por si mismos, se disparan automáticamente.
- Se pueden crear más de un trigger para un mismo evento en una tabla, con lo cual se pueden controlar múltiples alternativas sobre la misma tabla.
- Permiten evaluar el estado de una tabla antes y después de una modificación y tomar acciones acordes a la evaluación.

Los triggers son útiles para:

- Validación compleja de datos
- Eventos de auditoría
- Forzar reglas de negocio
- Mantener la integridad de los datos

Los triggers en SQL Server manejan dos tablas especiales que contienen toda la información que se necesitan: inserted y deleted. Ambas tablas son subconjuntos de la tabla que contiene el trigger, precisamente con los registros afectados por la sentencia desencadenante.

La tabla especial **deleted**, contiene los registros con los viejos valores para triggers que se desencadenan con sentencias DELETE (registros borrados) y UPDATE (valor anterior para los registros actualizados).

Mientras que la tabla especial inserted, contiene los registros con los nuevos valores para triggers que se desencadenan con sentencias INSERT (nuevos registros) y UPDATE (nuevo valor para registros actualizados).

Tipo de Trigger	Utiliza tabla especial
AFTER INSERT	Inserted (consultar nuevos valores).
AFTER DELETE	Deleted (consultar valores anteriores).
AFTER UPDATE	Deleted e Inserted (consultar así los valores antes y después de actualizarse los registros correspondientes).

Sintaxis del commando Trigger:

```
CREATE TRIGGER <Nombre del Trigger>
ON <Nombre de la Tabla>
AFTER <INSERT,DELETE,UPDATE>
AS
BEGIN
SET NOCOUNT ON;
-- Inserta aquí las instrucciones
END
```

Con base en el ejercicio de la Auditoría de la tabla Pacientes, se procede con las siguientes actividades:

- Crear tabla Auditoria_Pacientes: Se puede utilizar el ambiente gráfico de SSMS (SQL Server Management Studio) o utilizar la sentencia Create table. Los campos de la tabla Auditoría son:

Nombre de columna	Tipo de datos	Permitir valores NULL
ID_AUDI	int	<input type="checkbox"/>
AUDI_PACIDENTIFICACION	char(10)	<input checked="" type="checkbox"/>
AUDI_NOMBRE_ANTERIOR	varchar(50)	<input checked="" type="checkbox"/>
AUDI_APELLIDO_ANTERIOR	varchar(50)	<input checked="" type="checkbox"/>
AUDI_FECHA_ANTERIOR	date	<input checked="" type="checkbox"/>
AUDI_NOMBRE_NUEVO	varchar(50)	<input checked="" type="checkbox"/>
AUDI_APELLIDO_NUEVO	varchar(50)	<input checked="" type="checkbox"/>
AUDI_FECHA_NUEVO	date	<input checked="" type="checkbox"/>
AUDI_USUARIO	varchar(50)	<input checked="" type="checkbox"/>
AUDI_ACCION	varchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Figura 53. Estructura de la tabla de Auditoría en SQL Server.

- Creación del Trigger de Auditoría:

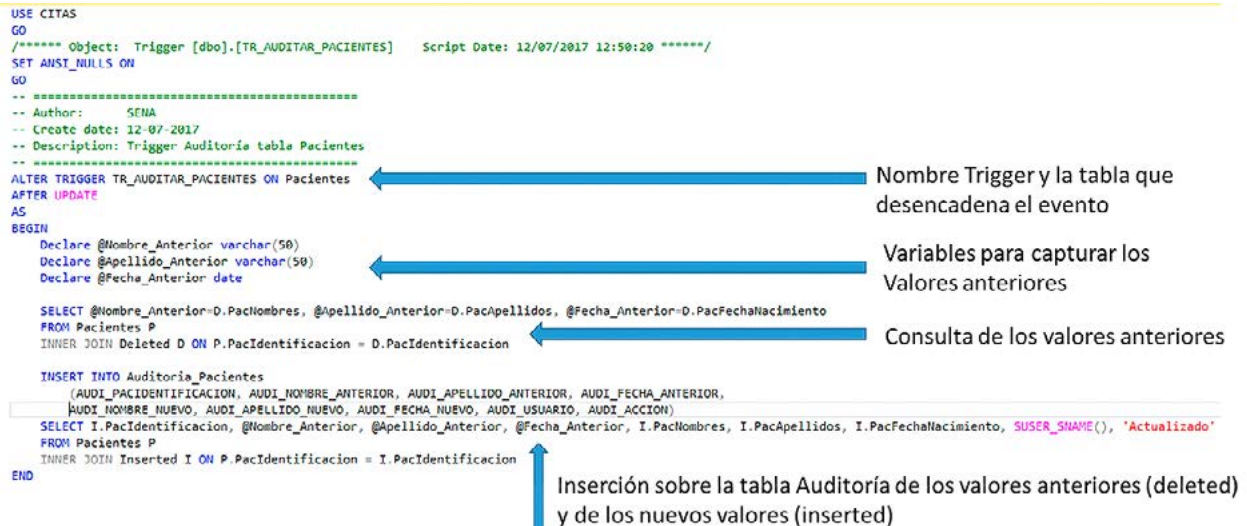


Figura 54. Estructura de un trigger en SQL Server.

- Consulta sobre los datos actuales de la tabla Pacientes:

```

SELECT PacIdentificacion, PacNombres, PacApellidos, PacFechaNacimiento, PacSexo
FROM Pacientes
    
```

	PacIdentificacion	PacNombres	PacApellidos	PacFechaNaci...	PacSexo
▶	1098765678	Carolina	Rojas Zabala	1984-06-28	F
	37821200	Evelia	Arias Mendoza	1970-03-25	F
	37821203	Maria Fernanda	Rodríguez Perez	1970-07-28	F
	77191950	Carlos José	Gómez Plata	1980-04-12	M
*	NULL	NULL	NULL	NULL	NULL

Figura 55. Consulta de la tabla Pacientes en SQL Server.

- Se inserta un nuevo registro en la tabla Pacientes, sobre este nuevo registro se van a realizar las modificaciones que van a ser Auditadas y se realiza consulta de resultados sobre las tablas Pacientes y Auditoría para validar sus contenidos:

```

INSERT INTO Pacientes (PacIdentificacion,PacNombres,PacApellidos,PacFechaNacimiento,PacSexo) VALUES ('91267508','EDGAR','VEGA','1969-08-12','M')
SELECT * FROM PACIENTES
SELECT * FROM AUDITORIA_PACIENTES

```

	PacIdentificacion	PacNombres	PacApellidos	PacFechaNacimiento	PacSexo
1	1098765678	Carolina	Rojas Zabala	1984-06-28	F
2	37821200	Evelia	Arias Mendoza	1970-03-25	F
3	37821203	Maria Fernanda	Rodríguez Perez	1970-07-28	F
4	77191950	Carlos José	Gómez Plata	1980-04-12	M
5	91267508	EDGAR	VEGA	1969-08-12	M

ID_AUDI	AUDI_PACIDENTIFICACION	AUDI_NOMBRE_ANTERIOR	AUDI_APELLIDO_ANTERIOR	AUDI_FECHA_ANTERIOR	AUDI_NOMBRE_NUEVO	AUDI_APELLIDO_NUEVO	AUDI_FECHA_NUEVO	AUDI_USUARIO	AUDI_ACC
---------	------------------------	----------------------	------------------------	---------------------	-------------------	---------------------	------------------	--------------	----------

Figura 56. Inserción de registro en tabla Pacientes y consultas de tablas Pacientes y Auditoria.

- Se procede a realizar la actualización sobre el nuevo registro, con el comando Update y se realiza la consulta sobre las tablas Pacientes y Auditoría:

Figura 57. Actualización de registro en tabla Pacientes y consultas de tablas Pacientes y

```

UPDATE Pacientes SET PacNombres='EDUARDO',PacApellidos='ARANGO',PacFechaNacimiento='2000-08-19',PacSexo='M'
WHERE PacIdentificacion='91267508'
SELECT * FROM PACIENTES
SELECT * FROM AUDITORIA_PACIENTES

```

	PacIdentificacion	PacNombres	PacApellidos	PacFechaNacimiento	PacSexo
1	1098765678	Carolina	Rojas Zabala	1984-06-28	F
2	37821200	Evelia	Arias Mendoza	1970-03-25	F
3	37821203	Maria Fernanda	Rodríguez Perez	1970-07-28	F
4	77191950	Carlos José	Gómez Plata	1980-04-12	M
5	91267508	EDUARDO	ARANGO	2000-08-19	M

ID_AUDI	AUDI_PACIDENTIFICACION	AUDI_NOMBRE_ANTERIOR	AUDI_APELLIDO_ANTERIOR	AUDI_FECHA_ANTERIOR	AUDI_NOMBRE_NUEVO	AUDI_APELLIDO_NUEVO	AUDI_FECHA_NUEVO	AUDI_USUARIO	AUDI_ACC
1	91267508	EDGAR	VEGA	1969-08-12	EDUARDO	ARANGO	2000-08-19	sa	Actualizado

Auditoria.

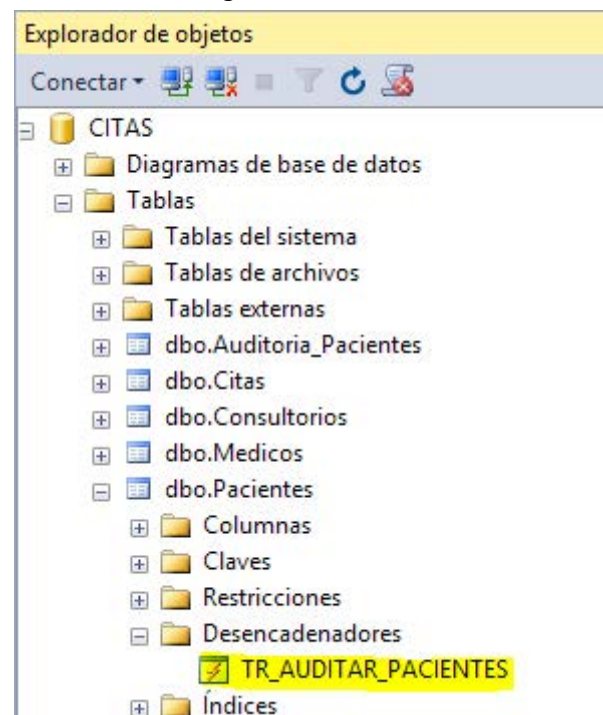
Al realizar el Update se dispara el Trigger, insertando el registro auditado en la tabla Auditoria_Pacientes como se ilustra en la imagen anterior.

Por último, se puede observar en la siguiente imagen el trigger creado en la tabla Pacientes:

Figura 58. Lista de triggers de la tabla Pacientes en SQL Server.

Para eliminar un trigger en SQL Server utilizar la siguiente instrucción:

```
drop trigger nombre_trigger
```



Glosario

BEGIN: limitador del procedimiento.

CALL: llamar un procedimiento previamente creado.

DELIMITER: restablece el punto y coma como delimitador.

END: fin del procedimiento.

IN: indica que el parámetro del trigger será de entrada.

INTOUT: indica que el parámetro del trigger será tanto de Entrada como de salida.

OUT: nos indica que el parámetro del trigger será de salida.

SGBD: Sistema de Gestión de Bases de Datos.

SSMS: SQL Server Management Studio.

Trigger: disparador.

Bibliografía

Documentación técnica de SQL Server. (2017). Recuperado de <https://docs.microsoft.com/es-es/sql/sql-server/sql-server-technical-documentation>

Manual de Referencia MySQL. (2017). Recuperado de <http://dev.mysql.com/doc/refman/5.7/en/>

Manual de Referencia Oracle. Recuperado de <https://www.oracle.com/technetwork/index.html>

Nielsen P. (2008). *Microsoft SQL Server 2008 Bible*. Wiley Publishing Inc.

Oracle Database 11g. (2009). *Develop PL/SQL Program Units*. Volumen I. Student Guide.

Tutorial SQL Server Management Studio. (2016). Recuperado de <https://docs.microsoft.com/es-es/sql/ssms/tutorials/tutorial-sql-server-management-studio>

Control de documento

CONSTRUCCIÓN OBJETO DE APRENDIZAJE



LENGUAJE TRANSACCIONAL EN SQL

Centro Industrial de Mantenimiento Integral - CIMI
Regional Santander

Líder línea de producción: Santiago Lozada Garcés

Asesores pedagógicos: Rosa Elvia Quintero Guasca
Claudia Milena Hernández Naranjo

Líder expertos temáticos: Rita Rubiela Rincón Badillo

Experto temático: Magda Milena García Gamboa (V1)
Edgar Eduardo Vega Arango (V2)

Diseño multimedia: Luis Gabriel Urueta Alvarez

Programador: Francisco José Lizcano Reyes

Producción de audio: Víctor Hugo Tabares Carreño

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de la licencia que el trabajo original.



**creative
commons**

BY NC SA

MySQL ® y Oracle Database ®,
Copyright © Oracle Corporation.
Microsoft SQL Server ®,
Copyright © Microsoft.
Todos los derechos reservados.



Registered trademark