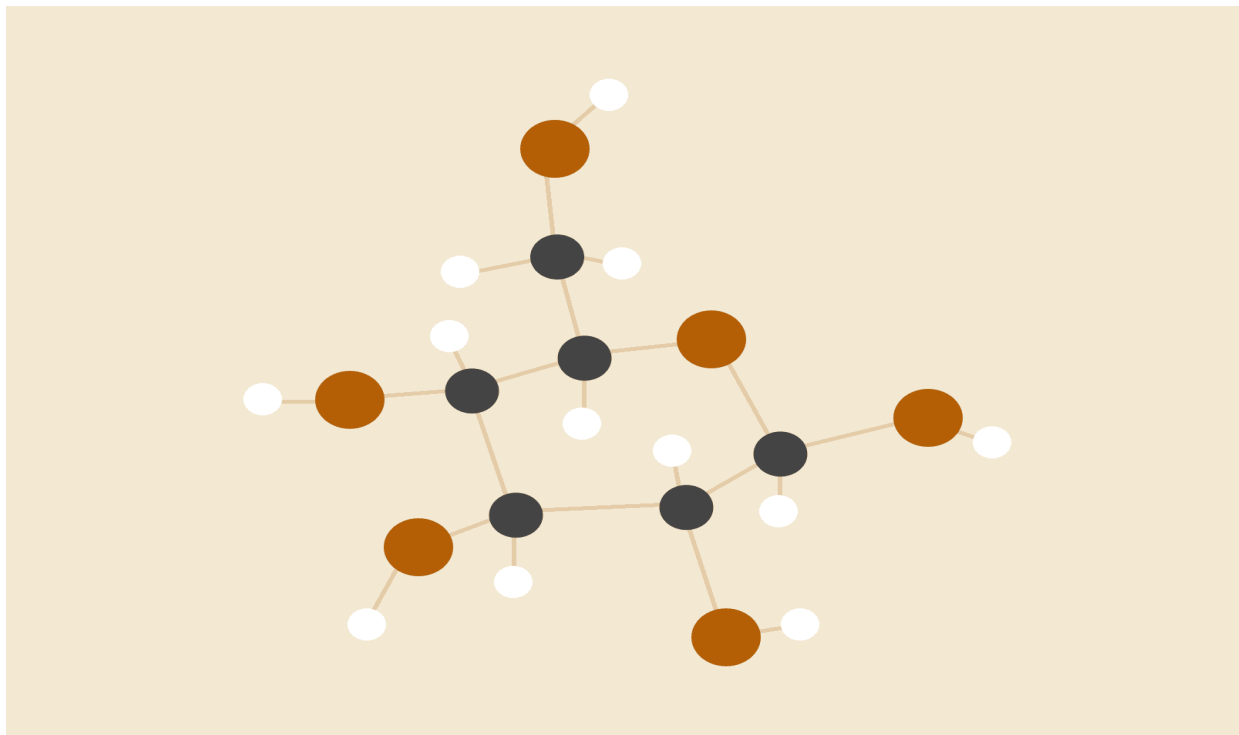


# PROCEDIMIENTOS Y FUNCIONES ALMACENADAS



**CADAU, Luis**  
**CRISTALDO, Fidel**

31/10/2023

**Asignatura**

Bases de Datos I

**Docentes**

Lic. Dario O. VILLEGAS

Lic. Walter O. VALLEJOS

Exp. Juan Jose Cuzziol

Lic. Ivan Sambrana

<b>INTRODUCCIÓN.....</b>	<b>3</b>
<b>MARCO CONCEPTUAL.....</b>	<b>3</b>
<b>METODOLOGÍA.....</b>	<b>4</b>
CREACIÓN DE LA BASE DE DATOS DE EJEMPLO.....	4
DESARROLLO DE NUEVOS SCRIPTS.....	4
PRUEBAS Y EVALUACION DE SCRIPTS.....	4
HERRAMIENTAS.....	4
<b>DESARROLLO.....</b>	<b>5</b>
MODELO LÓGICO.....	5
DICCIONARIO DE DATOS.....	6
Localidad.....	6
Provincia.....	6
Zona.....	7
Conserje.....	7
Consortio.....	7
Administrador.....	8
Inmueble.....	9
Gasto.....	10
TipoGasto.....	10
INSERTANDO DATOS.....	11
MODIFICANDO DATOS.....	13
ELIMINANDO DATOS.....	14
CONSULTADO DATOS.....	16
<b>CONCLUSIÓN.....</b>	<b>20</b>
<b>REFERENCIAS.....</b>	<b>21</b>

## INTRODUCCIÓN

En el dinámico mundo de la gestión de bases de datos, la eficacia y la eficiencia son elementos esenciales que impulsan el rendimiento de cualquier proyecto. El uso de las herramientas adecuadas puede marcar una diferencia significativa en la optimización de las operaciones de manipulación de datos.

Este estudio se adentra en el universo de las bases de datos y se enfoca en dos herramientas fundamentales: los procedimientos almacenados y las funciones definidas por el usuario en SQL Server. Estos elementos desempeñan un papel crucial en la simplificación y eficiencia de las operaciones relacionadas con bases de datos.

Este estudio proporcionará a profesionales de bases de datos, desarrolladores y administradores, el conocimiento básico sobre cómo utilizar procedimientos almacenados y funciones definidas por el usuario en sus proyectos. Con ello, se espera mejorar la eficiencia y la productividad en la gestión de datos.

## MARCO CONCEPTUAL

Un **procedimiento almacenado** de SQL Server es un grupo de una o varias instrucciones Transact-SQL o una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework.

Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos. Entre otras, pueden contener llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores, y el motivo de estos.

Una **función** definida por el usuario es una rutina de Transact-SQL o Common Language Runtime (CLR) que acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como valor.

El valor devuelto puede ser un valor escalar (único) o una tabla. Utilice esta instrucción para crear una rutina reutilizable que se pueda utilizar de estas formas:

- En instrucciones Transact-SQL como SELECT
- En las aplicaciones que llaman a la función
- En la definición de otra función definida por el usuario
- Para parametrizar una vista o mejorar la funcionalidad de una vista indizada
- Para definir una columna en una tabla
- Para definir una restricción CHECK en una columna
- Para reemplazar un procedimiento almacenado
- Usar una función insertada como predicado de filtro de la directiva de seguridad

## **METODOLOGÍA**

### **CREACIÓN DE LA BASE DE DATOS DE EJEMPLO**

En una etapa inicial, se generó una base de datos de ejemplo utilizando los scripts proporcionados por el equipo docente. Esta base de datos sirvió como punto de partida para nuestros estudios y experimentos.

### **DESARROLLO DE NUEVOS SCRIPTS**

Estos scripts representaron la aplicación de los conocimientos obtenidos durante la investigación y constituyeron una parte esencial de nuestro trabajo.

### **PRUEBAS Y EVALUACION DE SCRIPTS**

Los nuevos scripts desarrollados fueron sometidos a pruebas para evaluar su rendimiento y eficacia. Durante este proceso, se identificaron las ventajas de utilizar los objetos de la investigación en el contexto de la base de datos

### **HERRAMIENTAS**

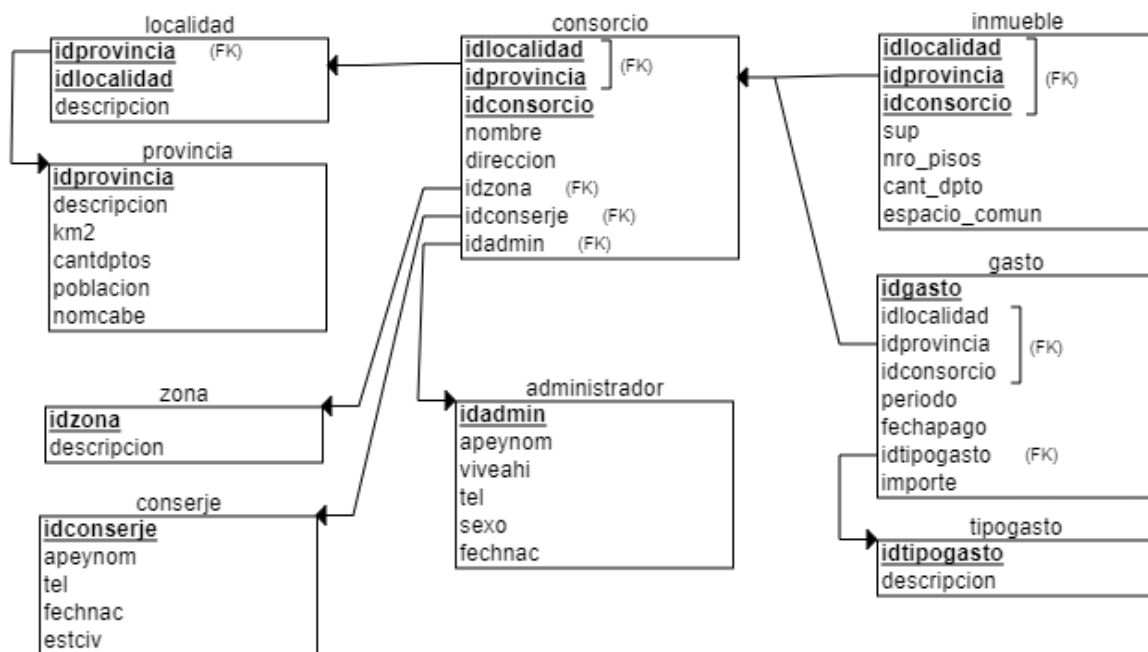
En lo que respecta a las herramientas utilizadas en este estudio, se optó por emplear el motor de base de datos SQL Server 16 y el SQL Server Management Studio 18. Estas herramientas proporcionaron el entorno adecuado para llevar a cabo las pruebas y experimentos de manera eficiente.

Además, cabe destacar que los conocimientos necesarios para llevar a cabo esta investigación se obtuvieron a partir de la documentación oficial de SQL Server proporcionada por Microsoft. Esta documentación sirvió como fuente de referencia fundamental, garantizando la correcta aplicación de los conceptos y procedimientos relacionados con SQL Server.

## DESARROLLO

En este experimento iniciaremos con una base de datos que mantiene la información relacionadas a consorcios, administradores y conserjes de edificios.

### MODELO LÓGICO



## DICCIONARIO DE DATOS

### Localidad

CAMPO	TIPO	DESCRIPCION
idprovincia	int	Referencia a la provincia a la que pertenece la localidad
idlocalidad	int	Identificador único dentro de la provincia
descripcion	varchar(50)	Nombre de la localidad. Puede ser nulo.

### Provincia

CAMPO	TIPO	DESCRIPCION
idprovincia	int	Identificador único de la provincia
descripcion	varchar(50)	Nombre de la provincia. Puede ser nulo.
km2	int	Superficie de la provincia en kilómetros cuadrados. Puede ser nulo.
cantdptos	int	Cantidad de localidades que posee la provincia. Puede ser nulo.
poblacion	int	Cantidad de habitantes que posee la provincia. Puede ser nulo.
nomcabe	varchar(50)	Nombre de la capital de la provincia. Puede ser nulo.

### Zona

CAMPO	TIPO	DESCRIPCION
idzona	int	Identificador único de la zona
descripcion	varchar(50)	Nombre de la zona. Puede ser nulo.

### Conserje

CAMPO	TIPO	DESCRIPCION
idconserje	int	Identificador único del conserje.
apeynom	varchar(50)	Apellido y nombre del conserje. Puede ser nulo.
tel	varchar(20)	Teléfono del conserje. Puede ser nulo.
fechnac	datetime	Fecha de nacimiento del conserje. Puede ser nulo.
estciv	varchar(1)	Estado civil del conserje. Puede ser nulo. Valores posibles: <ul style="list-style-type: none"><li>- S - Soltero</li><li>- C - Casado</li><li>- D - Divorciado</li><li>- O - Otro</li></ul>

### Consortio

CAMPO	TIPO	DESCRIPCION
idprovincia	int	Referencia a la provincia a la que pertenece el consorcio

idlocalidad	int	Referencia, junto con el idprovincia, a la localidad a la que pertenece el consorcio
idconsorcio	int	Identificador único del consorcio dentro de la localidad
nombre	varchar(50)	Nombre del consorcio. Puede ser nulo.
direccion	varchar(250)	Dirección en la que se encuentra ubicado el consorcio. Puede ser nulo.
idzona	int	Referencia a la zona en donde está ubicado el consorcio. Puede ser nulo.
idconserje	int	Referencia al conserje asignado al consorcio. Puede ser nulo.
idadmin	int	Referencia al administrador asignado al consorcio. Puede ser nulo.

#### Administrador

CAMPO	TIPO	DESCRIPCION
idadmin	int	Identificador único del administrador
apeynom	varchar(50)	Apellido y nombre del administrador. Puede ser nulo.
viveahi	varchar(1)	Indica si el administrador reside en el consorcio al que esta asignado. Puede ser nulo. Valores posibles: - N - No - S - Si



tel	varchar(20)	Teléfono del administrador. Puede ser nulo.
sexo	varchar(1)	Sexo del administrador. Valores posibles: - M - Masculino - F - Femenino
fechnac	datetime	Fecha de Nacimiento del administrador. Puede ser nulo.

### Inmueble

CAMPO	TIPO	DESCRIPCION
idprovincia	int	Referencia a la provincia a la que pertenece el inmueble
idlocalidad	int	Referencia, junto a idprovincia, a la localidad a la que pertenece el inmueble
idconsorcio	int	Referencia, junto a idprovincia e idlocalidad, al consorcio al que pertenece el inmueble
sup	decimal(6,2)	Superficie del inmueble en metros cuadrados. Puede ser nulo.
nro_pisos	int	Cantidad de pisos que posee el inmueble. Puede ser nulo.
cant_dpto	int	Cantidad de departamentos que posee el inmueble. Puede ser nulo.
espacio_comun	int	Cantidad de espacios comunes

### Gasto

CAMPO	TIPO	DESCRIPCION
idgasto	int	Identificador único del gasto
idprovincia	int	Referencia a la provincia donde esta situado el consorcio al que pertenece el gasto
idlocalidad	int	Referencia a la localidad, junto con idprovincia, donde está situado el consorcio al que pertenece el gasto
idconsorcio	int	Referencia al consorcio, junto con idprovincia e idlocalidad, al que pertenece el gasto
periodo	int	Número de mes del año al que pertenece el gasto. Puede ser nulo.
fechapago	datetime	Fecha en la que se realizó el pago del gasto. Puede ser nulo.
idtipogasto	int	Referencia al tipo de gasto. Puede ser nulo.
importe	decimal(8,2)	Importe del gasto. Puede ser nulo.

### TipoGasto

CAMPO	TIPO	DESCRIPCION
idtipogasto	int	Identificador único del tipo de gasto
descripcion	varchar(50)	Descripción del gasto.

		Puede ser nulo.
--	--	-----------------

## INSERTANDO DATOS

En la siguiente imagen se observa la forma mas simple de insertar datos en la tabla administrador utilizando la sentencia INSERT:

```
INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
VALUES ('MARIELA CORREA', 'N', '3794123456', 'F', '1990-10-03')
```

La ejecución de la misma inserta los datos de forma exitosa. Pero qué sucede cuando se viola alguna restricción de la tabla?

```
INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
VALUES ('MARIELA CORREA', 'G', '3794123456', 'F', '1990-10-03')
```

Se obtiene el siguiente error:

```
Msg 547, Level 16, State 0, Line 3
The INSERT statement conflicted with the CHECK constraint "CK_habitante_viveahi". The conflict occurred in database "base_consortio", table "dbo.administrador", column 'viveahi'.
The statement has been terminated.

Completion time: 2023-10-28T19:12:46.3705245-03:00
```

En el script anterior, se está violando la restricción del campo viveahi, eso hace que la inserción falle, y eso ocasiona el campo idadmin se incremente de todas formas, ya que posee la propiedad de ser autoincremental. Esto puede prevenirse con el manejo de errores. Observe el siguiente script:

```
BEGIN TRY
    BEGIN TRAN
    INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
    VALUES ('MARIELA CORREA', 'G', '3794123456', 'F', '1990-10-03')
    COMMIT TRAN
END TRY
BEGIN CATCH
    DECLARE @mensaje varchar(200)
    SET @mensaje = CASE
        WHEN ERROR_MESSAGE() LIKE '%CK_habitante_viveahi%' THEN 'Valor incorrecto para el campo viveahi. Los valores posibles son S y N.'
        ELSE ERROR_MESSAGE()
    END
    ROLLBACK TRAN;
    THROW 51000, @mensaje, 1;
END CATCH
```

Aquí se ve como puede prevenirse el incremento del campo autoincremental con el uso de transacciones y de bloques TRY-CATCH. También se añadió la personalización del mensaje de error con el uso de la instrucción THROW. Se obtuvo el siguiente resultado:

```
(0 rows affected)
Msg 51000, Level 16, State 1, Line 16
Valor incorrecto para el campo viveahi. Los valores posibles son S y N.

Completion time: 2023-10-28T19:27:08.8894239-03:00
```

Toda esta lógica utilizada tendría que repetirse cada vez que el usuario necesite insertar un registro, bien podría estar dentro de un procedimiento almacenado. Esto se vería de la siguiente forma:

```
CREATE PROCEDURE [InsertarAdministrador] (
    @apeynom varchar(50) = null,
    @viveahi varchar(1) = null,
    @tel varchar(20) = null,
    @sexo varchar(1) = null,
    @fechnac datetime = null
)
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN
            INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
            VALUES (@apeynom, @viveahi, @tel, @sexo, @fechnac)
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            DECLARE @mensaje varchar(200)
            SET @mensaje = CASE
                WHEN ERROR_MESSAGE() LIKE '%CK_habitante_viveahi%' THEN 'Valor incorrecto para el campo viveahi. Los valores posibles son S y N.'
                ELSE ERROR_MESSAGE()
            END
            ROLLBACK TRAN;
            THROW 51000, @mensaje, 1;
        END CATCH
    END
```

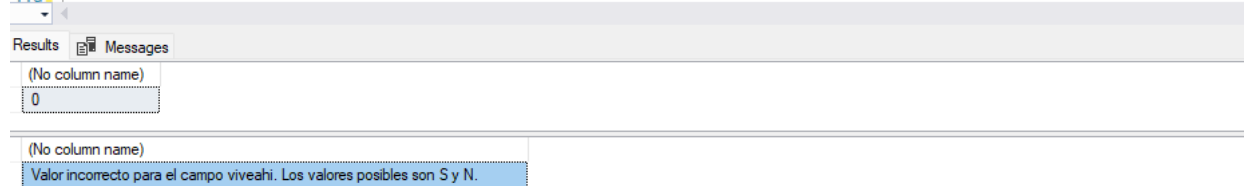
O bien podría utilizar el tipo de parámetros OUT para indicar si la inserción fue exitosa.

Observe:

```
CREATE PROCEDURE [InsertarAdministrador] (
    @apeynom varchar(50) = null,
    @viveahi varchar(1) = null,
    @tel varchar(20) = null,
    @sexo varchar(1) = null,
    @fechnac datetime = null,
    @exito bit OUT,
    @error varchar(200) OUT
)
AS
BEGIN
    SET @exito = 1
    BEGIN TRY
        BEGIN TRAN
            INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
            VALUES (@apeynom, @viveahi, @tel, @sexo, @fechnac)
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            SET @error = CASE
                WHEN ERROR_MESSAGE() LIKE '%CK_habitante_viveahi%' THEN 'Valor incorrecto para el campo viveahi. Los valores posibles son S y N.'
                ELSE ERROR_MESSAGE()
            END
            ROLLBACK TRAN;
            SET @exito = 0
        END CATCH
    END
```

La siguiente imagen demuestra el comportamiento ante una inserción fallida:

```
114 DECLARE @insercionExitosa bit
115 DECLARE @mensajeError varchar(200)
116 EXEC InsertarAdministrador 'MARIELA CORREA', 'G', '3794123456', 'F', '1990-10-03', @insercionExitosa OUT, @mensajeError OUT
117 SELECT @insercionExitosa
118 SELECT @mensajeError
119
```

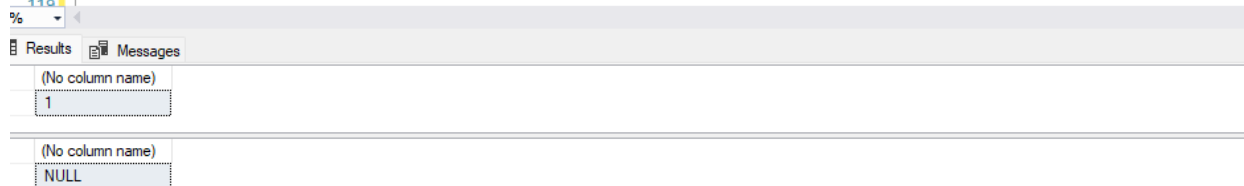


(No column name)
0

(No column name)
Valor incorrecto para el campo viveahi. Los valores posibles son S y N.

Y este script que sigue una inserción exitosa:

```
113
114 DECLARE @insercionExitosa bit
115 DECLARE @mensajeError varchar(200)
116 EXEC InsertarAdministrador 'MARIELA CORREA', 'S', '3794123456', 'F', '1990-10-03', @insercionExitosa OUT, @mensajeError OUT
117 SELECT @insercionExitosa
118 SELECT @mensajeError
119
```



(No column name)
1

(No column name)
NULL

## MODIFICANDO DATOS

Veamos qué sucede cuando lo que se desea es modificar datos de una tabla. Observe el siguiente script:

```
UPDATE administrador
SET viveahi='S'
WHERE idadmin=174
```

De forma análoga a lo realizado en la inserción de los datos, se puede aplicar lógica para el manejo de errores aplicado el uso de subrutinas.

```

CREATE PROCEDURE [CambiarResidenciaAdministrador] (
    @idadmin int = null,
    @viveahi varchar(1) = null,
    @exito bit OUT,
    @error varchar(200) OUT
)
AS
BEGIN
    SET @exito = 1
    BEGIN TRY
        BEGIN TRAN
        UPDATE administrador
        SET viveahi=@viveahi
        WHERE idadmin=@idadmin
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        SET @error = CASE
            WHEN ERROR_MESSAGE() LIKE '%CK_habitante_viveahi%' THEN 'Valor incorrecto para el campo viveahi. Los valores posibles son S y N.'
            ELSE ERROR_MESSAGE()
        END
        ROLLBACK TRAN;
        SET @exito = 0
    END CATCH
END

```

Observe el resultado al tratar de actualizar información que viola las restricciones de la tabla:

```

121 DECLARE @insercionExitosa bit
122 DECLARE @mensajeError varchar(200)
123 EXEC CambiarResidenciaAdministrador 174, 'F' , @insercionExitosa OUT, @mensajeError OUT
124 SELECT @insercionExitosa
125 SELECT @mensajeError

```

Results	Messages
(No column name)	
0	
(No column name)	
	Valor incorrecto para el campo viveahi. Los valores posibles son S y N.

## ELIMINANDO DATOS

Cuando lo que se quiere es eliminar un registro sucede lo mismo, observe la siguiente sentencia:

```

DELETE FROM administrador
WHERE idadmin=174

```

Cuando se quiere eliminar, para verificar que se elimina el registro deseado, se puede agregar una validacion de la existencia del mismo:

```

CREATE PROCEDURE [EliminarAdministrador] (
    @idadmin int = null,
    @exito bit OUT,
    @error varchar(200) OUT
)
AS
BEGIN
    SET @exito = 1
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM administrador WHERE idadmin=@idadmin)
        BEGIN
            SET @exito = 0
            SET @error = 'Administrador inexistente'
            RETURN
        END
        BEGIN TRAN
        DELETE FROM administrador
        WHERE idadmin=@idadmin
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN;
        SET @error = ERROR_MESSAGE()
        SET @exito = 0
    END CATCH
END

```

En el caso que no exista el registro, se puede informar mediante los parametros OUT:

```

126
127 DECLARE @insercionExitosa bit
128 DECLARE @mensajeError varchar(200)
129 EXEC [EliminarAdministrador] 174, @insercionExitosa OUT, @mensajeError OUT
130 SELECT @insercionExitosa
131 SELECT @mensajeError

```

Results	Messages
(No column name)	
0	
(No column name)	
	Administrador inexistente

## CONSULTADO DATOS

Cuando se consultan datos sucede lo siguiente:

```
3 SELECT
4     [Nombre y Apellido]    = apeynom,
5     [Telefono]             = tel,
6     [Sexo]                 = IIF(sexo='F', 'Femenino', 'Masculino'),
7     [Fecha de Nacimiento] = FORMAT(CONVERT(date, fechnac), 'dd-MM-yyyy'),
8     [Edad]                 = DATEDIFF(YEAR, fechnac, GETDATE())
9 FROM administrador
```

%

Results Messages

	Nombre y Apellido	Telefono	Sexo	Fecha de Nacimiento	Edad
	Perez Juan Manuel	3794112233	Masculino	18-02-1985	38
	BASUALDO DELMIRA	3624231689	Femenino	09-10-1980	43
	SEGOVIA ALEJANDRO H.	3624232689	Masculino	02-06-1974	49
	ROMERO ELEUTERIO	3624233689	Masculino	19-08-1972	51
	NAHMIAS DE K. NIDIA	3624234689	Femenino	28-11-1971	52
	CORREA DE M. MARIA G.	3624235689	Femenino	16-01-1990	33
	NAHMIAS JOSE	3624236689	Masculino	02-09-1974	49
	NAHMIAS DE R. REBECA J.	3624237689	Femenino	07-03-1989	34
	LOVATO CERENTINI ISABEL	3624238689	Femenino	15-10-1973	50
0	GOMEZ MATIAS GABRIEL	3624239689	Masculino	20-03-1974	49
1	CORREA HUGO E.	3624231689	Masculino	11-08-1993	30
2	MACHUCA CEFERINA	3624232689	Femenino	16-09-1991	32
3	CARDOZO MAXIMA	3624233689	Femenino	07-11-1988	35
4	RODRIGUEZ MARTIN J.	3624234689	Masculino	09-12-1985	38
5	SOTELO GERTRUDIS	3624235689	Femenino	12-11-1990	33
6	AYALA FLORENTINA	3624236689	Femenino	27-04-1974	49

Esto bien se puede usar dentro de un procedimiento almacenado para poder ver los datos de los administradores de una forma deseada.

```
CREATE PROCEDURE [VerAdministradores]
AS
BEGIN
SELECT
    [Nombre y Apellido]    = apeynom,
    [Telefono]             = tel,
    [Sexo]                 = IIF(sexo='F', 'Femenino', 'Masculino'),
    [Fecha de Nacimiento] = FORMAT(CONVERT(date, fechnac), 'dd-MM-yyyy'),
    [Edad]                 = DATEDIFF(YEAR, fechnac, GETDATE())
FROM administrador
END
```



Pero nos encontramos ante un problema si queremos saber la edad de algun administrador por fuera de este procedimiento. Por ello utilizaremos funciones definidas por el usuario. Esto nos da la habilidad de si hay algun tipo de error en el cálculo, solo haya que modificarse en un solo lugar, la definición de dicha función.

```
CREATE FUNCTION [CalcularEdad] (  
    @FechaNacimiento date  
)  
RETURNS int  
AS  
BEGIN  
    RETURN DATEDIFF(YEAR, @FechaNacimiento, GETDATE())  
END
```

Quedando así el procedimiento para ver los datos de los administradores:

```
CREATE PROCEDURE [VerAdministradores]  
AS  
BEGIN  
SELECT  
    [Nombre y Apellido]      = apeynom,  
    [Telefono]               = tel,  
    [Sexo]                   = IIF(sexo='F', 'Femenino', 'Masculino'),  
    [Fecha de Nacimiento]    = FORMAT(CONVERT(date, fechnac), 'dd-MM-yyyy'),  
    [Edad]                   = dbo.CalcularEdad(fechnac)  
FROM administrador  
END
```

La desventaja de utilizar procedimientos para mostrar datos, es que no se pueden aplicar filtros a los datos, como por ejemplo, solo ver los datos de los administradores mayores a 35 años. Pero esto se ve resuelto con otro tipo de funciones llamadas funciones de tabla.

```

CREATE FUNCTION [FuncionVerAdministradores] (
    @edadMinima int
)
RETURNS TABLE
AS
RETURN (
SELECT
    [Nombre y Apellido]      = apeynom,
    [Telefono]               = tel,
    [Sexo]                   = IIF(sexo='F', 'Femenino', 'Masculino'),
    [Fecha de Nacimiento]    = FORMAT(CONVERT(date, fechnac), 'dd-MM-yyyy'),
    [Edad]                   = dbo.CalcularEdad(fechnac)
FROM administrador
WHERE dbo.CalcularEdad(fechnac) >= @edadMinima
)

```

Observe:

140

141

142

SELECT \* FROM FuncionVerAdministradores(35)

%

Results

Messages

	Nombre y Apellido	Telefono	Sexo	Fecha de Nacimiento	Edad
1	Perez Juan Manuel	3794112233	Masculino	18-02-1985	38
2	BASUALDO DELMIRA	3624231689	Femenino	09-10-1980	43
3	SEGOVIA ALEJANDRO H.	3624232689	Masculino	02-06-1974	49
4	ROMERO ELEUTERIO	3624233689	Masculino	19-08-1972	51
5	NAHMIA DE K. NIDIA	3624234689	Femenino	28-11-1971	52
6	NAHMIA JOSE	3624236689	Masculino	02-09-1974	49
7	LOVATO CERENTINI ISABEL	3624238689	Femenino	15-10-1973	50
8	GOMEZ MATIAS GABRIEL	3624239689	Masculino	20-03-1974	49
9	CARDOZO MAXIMA	3624233689	Femenino	07-11-1988	35
10	RODRIGUEZ MARTIN J.	3624234689	Masculino	09-12-1985	38
11	AYALA FLORENTINA	3624236689	Femenino	27-04-1974	49
12	RATTI JUAN E.	3614238689	Masculino	07-08-1967	56
13	ROCH GRACIELA	3614238689	Femenino	24-05-1969	54
14	VILLALBA AMANDU, ALEJAND	3614231689	Masculino	27-04-1981	42
15	CACERES ROSA	3614232689	Femenino	08-10-1980	43
16	PALACIOS DE F. OLGA	3624233689	Femenino	17-10-1966	57
17	CONTRERAS JUAN	3624231689	Femenino	11-02-1985	38

Vea que la función puede ser utilizada como si fuese una tabla nueva, y así poder aplicar filtros desde la llamada:

```
140
141 SELECT * FROM FuncionVerAdministradores(35)
142 WHERE Sexo='Masculino'
143
```

9 %

Results Messages

	Nombre y Apellido	Telefono	Sexo	Fecha de Nacimiento	Edad
1	Perez Juan Manuel	3794112233	Masculino	18-02-1985	38
2	SEGOVIA ALEJANDRO H.	3624232689	Masculino	02-06-1974	49
3	ROMERO ELEUTERIO	3624233689	Masculino	19-08-1972	51
4	NAHMIAS JOSE	3624236689	Masculino	02-09-1974	49
5	GOMEZ MATIAS GABRIEL	3624239689	Masculino	20-03-1974	49
6	RODRIGUEZ MARTIN J.	3624234689	Masculino	09-12-1985	38
7	RATTI JUAN E.	3614238689	Masculino	07-08-1967	56
8	VILLALBA AMANDU, ALEJAND	3614231689	Masculino	27-04-1981	42
9	SAUCEDO LORENZO	3614235689	Masculino	19-04-1967	56

## CONCLUSION

Basándonos en la investigación y los experimentos realizados acerca de las funcionalidades proporcionadas por el motor de base de datos SQL Server, hemos observado que las capacidades de funciones y procedimientos almacenados nos permiten generalizar comportamientos, lo que a su vez facilita la reutilización y la obtención de un control más sistemático en nuestras consultas. Estos elementos pueden emplearse para realizar validaciones y para calcular nuevos valores basados en datos preexistentes, así como para operar en subconjuntos de datos que han sido previamente modificados.

Asimismo, estas funcionalidades contribuyen a la creación de scripts de mayor complejidad, que pueden ser guardados para su uso posterior, evitando la necesidad de reescribirlos en futuras ocasiones. Cabe destacar que el propio motor de SQL Server es capaz de seleccionar el plan de ejecución más apropiado para cada script, a la vez que registra estadísticas relacionadas con los mismos, lo que le permite optimizar continuamente dichos planes de ejecución.

En resumen, nuestros hallazgos revelan que SQL Server ofrece herramientas poderosas que impulsan la eficiencia y la versatilidad en el manejo de datos, al tiempo que promueven la reutilización de código y la optimización en la ejecución de consultas. Estas características son esenciales para mejorar la productividad y la eficacia en el entorno de bases de datos.

## REFERENCIAS

WilliamDAssafMSFT. (2023, 3 abril). *Procedimientos almacenados (motor de base de datos) - SQL server*. Microsoft Learn.

<https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16>

rwestMSFT. (2023, 23 mayo). *Funciones definidas por el usuario - SQL Server*.

Microsoft Learn.

<https://learn.microsoft.com/es-es/sql/relational-databases/user-defined-functions/user-defined-functions?view=sql-server-ver16>