



# 71.88 Gestión de Datos

Entrega 4 - Implementación de Procedures y Triggers

Grupo 02

Yasmin Ohana - 65098

Isidro Perasso - 65595

Andrés Glauberman - 65635

Julia Sexe - 65669

Fecha de entrega: Miércoles 21 de mayo

---

## Procedures

- **Procedure 1): Unidades sin multas dado un edificio y un período de tiempo.**

La utilidad del procedimiento sp\_cero\_mult es devolver el identificador de todas las unidades (id\_unidad) de un edificio que no registran ninguna multa dentro de un intervalo de fechas. Esto permite detectar departamentos sin multas para tareas de control de cumplimiento (por ejemplo, ofrecer beneficios para los departamentos sin multas), gestión administrativa (por ejemplo, generar informes excluyendo a quienes no recibieron ninguna multa) y, por último, para auditoría interna (por ejemplo, para corroborar que las multas fueron correctamente asignadas a las unidades).

El procedimiento recibe como parámetros de entrada las variables @id\_edificio (smallint), @fecha\_inicio (date) y @fecha\_fin (date). La primera corresponde al identificador del edificio que el usuario quiere analizar y las últimas dos marcan los límites del período. Durante la operatoria, se utilizan dos tablas: unidad, que contiene todas las unidades de todos los edificios, y multa, que almacena las infracciones de cada unidad con su fecha.

La operatoria del procedimiento es la siguiente: en primer lugar, se hace un left join entre unidad y multa, emparejando las filas por el id\_unidad y adjuntando las multas cuya fecha cae entre @fecha\_inicio y @fecha\_fin (si una unidad no cumple esa condición, las columnas de multa quedan en null). Segundo, se filtran las unidades, seleccionando aquellas cuyo identificador del edificio sea igual a @id\_edificio. En tercer lugar, se agrupa por id\_unidad para tener una sola fila por unidad. Luego, se cuentan cuántas multas hay en cada unidad y se indica que esa cantidad debe ser igual a cero. Por último, se seleccionan los id\_unidad de aquellas unidades que cumplieron esa condición.

Código:

```
CREATE PROCEDURE sp_cero_mult (@id_edificio smallint, @fecha_inicio date,  
@fecha_fin date)  
AS BEGIN
```

```

SELECT u.id_unidad
FROM UNIDAD u LEFT JOIN MULTA m ON u.ID_UNIDAD = m.ID_UNIDAD
AND m.fecha BETWEEN @fecha_inicio AND @fecha_fin
WHERE u.id_edificio = @id_edificio
GROUP BY u.id_unidad
HAVING COUNT(m.id_unidad) = 0
END

```

The screenshot shows a SQL Server Management Studio interface. The top window is titled "SQLQuery1.sql - 1...GD\_Grupo02M (115)\*". It contains the following SQL code:

```

select ID_MULTA, FECHA, m.ID_UNIDAD
from multa m
join UNIDAD u
on m.ID_UNIDAD=u.ID_UNIDAD
where u.ID_EDIFICIO=1

```

Below this is a results grid titled "Results". The grid displays the following data:

	ID_MULTA	FECHA	ID_UNIDAD
1	1	2025-02-18	1
2	2	2025-03-02	2
3	31	2025-02-26	1

```
EXEC sp_cero_mult 1, '2025-01-01', '2025-05-20'
```

100 %

Results Messages

	id_unidad
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	18
17	19
18	20

- **Procedure 2): Personal de mantenimiento con mejor calificación promedio dentro de los últimos dos años.**

La utilidad del procedimiento sp\_cal\_per es devolver el nombre y apellido del personal de mantenimiento con mejor calificación promedio en reparaciones realizadas dentro de los últimos dos años. Esto permite reconocer al empleado que mejor rendimiento tuvo para premios, aumentos, asignación de tareas críticas o campañas de comunicación interna (por ejemplo, destacar al empleado en comunicados).

El procedimiento no recibe ningún parámetro de entrada ya que el análisis del mejor personal de mantenimiento se realiza para todos los empleados del sistema en una ventana temporal de dos años. Durante la operatoria, se utilizan tres tablas: usuario, que contiene los atributos de los usuarios del sistema, personal\_mantenimiento, que contiene atributos específicos de los empleados y, por último, reparación, que registra cada trabajo realizado, incluyendo la fecha, descripción, calificación, entre otros.

La operatoria del procedimiento es la siguiente: en primer lugar, se hace un inner join entre usuario y personal\_mantenimiento por la relación entre id\_usuario e id\_personal y, posteriormente, con reparación, por medio del atributo id\_personal. Segundo, se filtran las reparaciones cuya calificación no esté vacía, la fecha esté entre la fecha de hoy y dos años atrás y el empleado debe estar activo en la aplicación. En tercer lugar, se agrupa por nombre y apellido del usuario para poder calcular el promedio de calificación de cada empleado. Luego, para colocar primero al mejor promedio se ordena de manera descendente según el promedio. Por último, se selecciona/n la/s primera/s fila/s según el orden anterior. Si varias filas empatan en el mismo valor de ordenamiento, WITH TIES incluye a todas. Esto lo utilizamos ya que sin WITH TIES, TOP 1 devuelve exactamente una fila y, si hay varios empleados con el mismo promedio máximo, el motor elige uno arbitrariamente. En cambio, con WITH TIES se garantiza que si hay un empate de mejor promedio entre dos o más empleados, todos sean seleccionados en el procedimiento.

Código:

```
CREATE PROCEDURE sp_cal_per
AS BEGIN
SELECT TOP 1 WITH TIES u.nombre_apellido
FROM USUARIO u INNER JOIN PERSONAL_MANTENIMIENTO pm
ON u.ID_USUARIO = pm.ID_PERSONAL INNER JOIN REPARACION r ON
pm.ID_PERSONAL = r.ID_PERSONAL
```

```

WHERE r.CALIFICACION IS NOT NULL AND r.FECHA >= GETDATE() - 365 * 2
AND r.FECHA <= GETDATE() AND u.FECHA_BAJA_APP IS NULL

GROUP BY u.NOMBRE_APELLIDO

ORDER BY AVG(r.CALIFICACION) DESC

END;

```

```

SQLQuery5.sql - 1...GD_Grupo02M (272)*  p  X  SQLQuery4.sql - 1...GD_Grupo02M (446))*
SQLQuery3.sql - 1...GD_Grupo02M (295))*  SQLQuery2.sql - 1...GD_Grupo02M (295))*
SELECT u.NOMBRE_APELLIDO, AVG(r.CALIFICACION)
FROM USUARIO u INNER JOIN PERSONAL_MANTENIMIENTO pm
ON u.ID_USUARIO = pm.ID_PERSONAL INNER JOIN REPARACION r ON pm.ID_PERSONAL = r.ID_PERSONAL
WHERE r.CALIFICACION IS NOT NULL AND r.FECHA >= GETDATE() - 365 * 2 AND r.FECHA <= GETDATE() AND u.FECHA_BAJA_APP IS NULL
GROUP BY u.NOMBRE_APELLIDO
ORDER BY AVG(r.CALIFICACION) DESC

```

Results

NOMBRE_APELLIDO	(No column name)
Irene Alcalde	10
Marina Bas	10
Simon Ayuso Camacho	8
Vicenta Salcedo Parejo	7
Gema del Anguita	7
Altana Ortiz Landa	4

```

EXEC sp_cal_per

```

Results

nombre_apellido
Irene Alcalde
Marina Bas

- **Procedure 3): Amenity más utilizado en el último año dado un edificio.**

La utilidad del procedimiento sp\_max\_ame es devolver el nombre de el o los amenities de un edificio que más se utilizaron en el último año, junto con la cantidad de reservas de ese o esos amenities. Si el procedimiento devuelve más de un amenity, significa que todos los amenities seleccionados tienen exactamente la misma cantidad de reservas máximas. Esto permite informar al consorcio cuál o cuáles instalaciones reciben mayor demanda, siendo esto útil para decisiones de mantenimiento, ampliación de horarios, inversiones o comunicación a los propietarios. A su vez, al tener como parámetro de salida el valor de la cantidad máxima de reservas, un consorcio puede, por ejemplo, decidir a partir de ese número si amplía el rango horario de reservas o realiza una inversión.

El procedimiento recibe como parámetro de entrada la variable @id\_edificio (smallint), que corresponde al identificador del edificio sobre el que se consulta, y de salida la variable @max\_res (int), que es la cantidad máxima de reservas encontradas en los amenities de ese edificio. Durante la operatoria, se utilizan tres tablas: amenity, que contiene los tipos de amenity, amenity\_x\_edificio, que contiene los amenities que existen en cada edificio y, por último, reserva\_amenity, que registra las reservas con fecha y hora, identificador del amenity y edificio, entre otros.

La operatoria del procedimiento se divide en dos consultas select. En la primera de ellas, se comienza uniendo las tres tablas previamente mencionadas y se filtra indicando que el identificador del edificio debe ser igual a @id\_edificio y la fecha y hora de reserva debe estar comprendida dentro del último año. Luego, se agrupa por tipo de amenity y se ordena el grupo contando la cantidad de reservas de cada amenity en orden descendente. Al realizar la selección del TOP 1, se asigna el mayor conteo a la variable @max\_res. Luego, este resultado será utilizado para utilizar en el segundo select.

En el segundo select, se repite la misma unión de tablas y filtros que en el anterior, se agrupa de la misma manera y se aplica una condición en el having indicando que el conteo de cada grupo debe ser igual al máximo, es decir, a la variable @max\_res. Luego, se selecciona el nombre de el o los amenities junto con la cantidad máxima

de reservas. Al utilizar la variable @max\_res, manejamos empates de reservas y evitamos que si hay dos amenities con la misma cantidad de reservas, únicamente se seleccione uno.

Código:

```
CREATE PROCEDURE sp_max_am ( @id_edificio smallint, @max_res int
OUTPUT)
```

```
AS BEGIN
```

```
SELECT TOP 1 @max_res = COUNT(ra.id_reserva)
```

```
FROM amenity a INNER JOIN amenity_x_edificio ae ON a.id_amenity =
ae.id_amenity INNER JOIN
```

```
reserva_amenity ra ON ae.id_amenity = ra.id_amenity AND ae.ID_EDIFICIO
= ra.ID_EDIFICIO
```

```
WHERE ae.id_edificio = @id_edificio AND ra.fecha_hora >=
DATEADD(YEAR,-1,GETDATE()) AND ra.fecha_hora <= GETDATE()
```

```
GROUP BY a.tipo_amenity
```

```
ORDER BY COUNT(ra.id_reserva) DESC;
```

```
SELECT a.tipo_amenity, @max_res
```

```
FROM amenity a INNER JOIN amenity_x_edificio ae ON a.id_amenity =
ae.id_amenity INNER JOIN
```

```
reserva_amenity ra ON ae.id_amenity = ra.id_amenity AND ae.ID_EDIFICIO
= ra.ID_EDIFICIO
```

```
WHERE ae.id_edificio = @id_edificio AND ra.fecha_hora >=
DATEADD(YEAR,-1,GETDATE()) AND ra.fecha_hora <= GETDATE()
```

GROUP BY a.tipo\_amenity

HAVING COUNT(ra.id\_reserva) = @max\_res

END;

```
SELECT * FROM RESERVA_AMENITY ra INNER JOIN AMENITY a ON
ra.ID_AMENITY = a.ID_AMENITY
WHERE ID_EDIFICIO = 30
```

100 %

Results Messages

	ID_RESERVA	FECHA_HORA	ID_UNIDAD	ID_EDIFICIO	ID_AMENITY	ID_EXPENSA	ID_AMENITY	TIPO_AMENITY
1	3	2025-03-26 00:00:00	300	30	3	900	3	Parrilla
2	4	2025-02-14 00:00:00	299	30	4	599	4	Salon
3	6	2025-02-16 00:00:00	300	30	5	600	5	Cancha de tenis
4	24	2025-02-27 00:00:00	300	30	3	600	3	Parrilla
5	25	2025-03-07 00:00:00	299	30	4	899	4	Salon
6	26	2025-03-13 00:00:00	297	30	5	897	5	Cancha de tenis
7	31	2025-05-14 20:46:00	297	30	1	NULL	1	Pileta
8	32	2025-05-14 20:53:00	297	30	1	NULL	1	Pileta

```
DECLARE @max_res int
EXEC sp_max_ame 30, @max_res OUTPUT
SELECT @max_res
```

100 %

Results Messages

	tipo_amenity	(No column name)
1	Cancha de tenis	2
2	Parrilla	2
3	Pileta	2
4	Salon	2

  

	(No column name)
1	2

- **Procedure 4): Inicialización del gasto mensual de un edificio en monto cero para el mes actual.**

La utilidad del procedimiento INS\_GASTO es insertar el gasto mensual para el mes de la fecha con el valor del monto en cero, que posteriormente va a ser actualizado. Esto permite que haya control y consistencia de los registros de gastos mensuales por edificio. A su vez, impide que el gasto se inserte en un mes y año que ya se insertó previamente. En la operación cotidiana del consorcio, tener esta herramienta es fundamental.

La utilidad del procedimiento es generar automáticamente un registro inicial para cada edificio al comienzo del mes, aún antes de conocer los importes definitivos. Esto permite preparar la estructura de datos para así luego poder cargar y actualizar los montos reales. Cargar los datos con el monto vacío a principio de mes permite generar las expensas con sus respectivos montos en cero para que, al insertar reparaciones, multas o reservas con costos, estas se reflejen en la expensa correspondiente de forma automática con un trigger (como algunos de los que creamos luego). También, este procedimiento evita la duplicación de registros dentro del mismo mes y año. Esto es fundamental para evitar insertar más de un registro para un mismo edificio y periodo, protegiendo así la integridad de la tabla y previniendo conflictos con pks y cálculos posteriores.

La operatoria del procedimiento es la siguiente: en primer lugar, verifica que el mes a insertar no exista ya que, de ser así, estaría mal agregar gastos correspondientes a meses y años ya existentes. Luego, hace el insert. Para los valores a insertar, el id del gasto corresponde al id\_edificio + el máximo id existente. Esto garantiza que si en un futuro se agregan más edificios el código estará apto para generar los gastos mensuales sin necesidad de realizar cambios manuales. Inicia el monto en cero y para definir la fecha se usa una estructura case que contempla los tres posibles días de generación de gastos de cada mes. Por último, para el id\_edificio se hace una verificación de que este no cuente con una fecha de baja.

Código:

```
CREATE PROCEDURE INS_GASTO
```

```
AS BEGIN

IF NOT( DATEPART(MONTH, GETDATE()) IN (SELECT MONTH(FECHA_GASTO)
FROM GASTO_MENSUAL) AND
DATEPART(YEAR, GETDATE()) IN (SELECT YEAR(FECHA_GASTO) FROM
GASTO_MENSUAL))

BEGIN

INSERT INTO GASTO_MENSUAL

SELECT ID_EDIFICIO+(SELECT MAX(ID_GASTO) FROM GASTO_MENSUAL), 0,
CASE

WHEN MONTH(GETDATE()) IN (1, 3, 5, 7, 8, 10, 12)

THEN CAST(FORMAT(GETDATE(), 'yyyy-MM-') + '31' AS DATE)

WHEN MONTH(GETDATE()) = 2

THEN CAST(FORMAT(GETDATE(), 'yyyy-MM-') + '28' AS DATE)

ELSE CAST(FORMAT(GETDATE(), 'yyyy-MM-') + '30' AS DATE)

END,

ID_EDIFICIO FROM EDIFICIO WHERE FECHA_BAJA IS NULL

END

ELSE

PRINT('ESTE MES YA SE HAN CARGADO LOS REGISTROS DE LOS GASTOS')

END;
```

```
select *  
from GASTO_MENSUAL
```

100 %

ID_GASTO	MONTO	FECHA_GASTO	ID_EDIFICIO	
109	109	18511064	2025-04-30	19
110	110	3303300	2025-04-30	20
111	111	15163720	2025-04-30	21
112	112	9938214	2025-04-30	22
113	113	11652784	2025-04-30	23
114	114	29798912	2025-04-30	24
115	115	18133544	2025-04-30	25
116	116	10419552	2025-04-30	26
117	117	33045584	2025-04-30	27
118	118	22887150	2025-04-30	28
119	119	13779480	2025-04-30	29
120	120	38538500	2025-04-30	30

Query executed successfully.

```
exec INS_GASTO
```

100 %

Messages

(29 rows affected)

Completion time: 2025-05-21T15:28:28.4605525-03:00

100 %

Query executed successfully.

Ln 6

```

select *
from GASTO_MENSUAL

```

The screenshot shows the results of the query execution. The table has four columns: ID\_GASTO, MONTO, FECHA\_GASTO, and ID\_EDIFICIO. The data consists of 36 rows, each representing a monthly expense record.

	ID_GASTO	MONTO	FECHA_GASTO	ID_EDIFICIO
117	117	33045584	2025-04-30	27
118	118	22887150	2025-04-30	28
119	119	13779480	2025-04-30	29
120	120	38538500	2025-04-30	30
121	121	0	2025-05-31	1
122	122	0	2025-05-31	2
123	123	0	2025-05-31	3
124	124	0	2025-05-31	4
125	125	0	2025-05-31	5
126	126	0	2025-05-31	6
127	127	0	2025-05-31	7
128	128	0	2025-05-31	8
129	129	0	2025-05-31	9
130	130	0	2025-05-31	10
131	131	0	2025-05-31	11
132	132	0	2025-05-31	12
133	133	0	2025-05-31	13
134	134	0	2025-05-31	14
135	135	0	2025-05-31	15
136	136	0	2025-05-31	16
137	137	0	2025-05-31	17
138	138	0	2025-05-31	18

Query executed successfully.

```

END;
EXEC INS_GASTO

```

The screenshot shows the execution of the stored procedure. The message in the output window indicates that all expenses for the month have been loaded.

Messages

ESTE MES YA SE HAN CARGADO LOS REGISTROS DE LOS GASTOS

Completion time: 2025-05-21T20:03:02.4103293-03:00

#### - Procedure 5): Listado de empleados actuales vinculados a un edificio.

La utilidad del procedimiento EMP\_POR\_EDI es devolver un listado actualizado de todos los empleados vinculados a un edificio, tanto administradores como personal de mantenimiento. Esto se debe a que comúnmente los consorcios necesitan este tipo de informes para tareas de coordinación, notificaciones, generación de credenciales de acceso o elaboración de reportes.

En primer lugar, centraliza en un único procedimiento la consulta que une dos tablas distintas evitando duplicar el mismo join y condiciones cada vez que se requiera esta información. Después, al recibir como parámetro el identificador de edificio, permite recuperar la información para cualquier edificio de forma sencilla. Es un procedimiento genérico que engloba a cualquiera sea el edificio que se desee consultar, siempre que este sea indicado como un input.

También, incluye sólo a las asignaciones actuales (la condición de FECHA\_FIN IS NULL), garantizando que no se muestren empleados cuya vinculación ya ha concluido. De esta manera, los usuarios de la aplicación siempre obtienen datos actualizados y relevantes.

Código:

```
CREATE PROCEDURE EMP_POR_EDI (@ID_EDIFICIO INT)
AS BEGIN
SELECT U.NOMBRE_APELLIDO, U.TELEFONO, U.EMAIL, U.TIPO_USUARIO AS
TIPO_EMPLEADO
FROM ADMINISTRADOR_X_EDIFICIO AE
JOIN ADMINISTRADOR A
ON AE.ID_ADMINISTRADOR=A.ID_ADMINISTRADOR
JOIN USUARIO U
ON A.ID_ADMINISTRADOR=U.ID_USUARIO
WHERE AE.FECHA_FIN IS NULL AND AE.ID_EDIFICIO=@ID_EDIFICIO
UNION
SELECT U.NOMBRE_APELLIDO, U.TELEFONO, U.EMAIL, U.TIPO_USUARIO AS
TIPO_EMPLEADO
FROM PERSONAL_X_EDIFICIO PE
```

```

JOIN PERSONAL_MANTENIMIENTO PM
ON PE.ID_PERSONAL=PM.ID_PERSONAL

JOIN USUARIO U
ON PM.ID_PERSONAL=U.ID_USUARIO

WHERE PE.FECHA_FIN IS NULL AND PE.ID_EDIFICIO=@ID_EDIFICIO

END

```

The screenshot shows a SQL Server Management Studio window titled 'SQLQuery1.sql - 1...(GD\_Grupo02M (55))\*.sql'. Below the title bar, there is a message bar with the text '| EXEC EMP\_POR\_EDI 1'. The main area is a grid labeled 'Results' which displays the output of the stored procedure. The columns are labeled 'NOMBRE\_APELLIDO', 'TELEFONO', 'EMAIL', and 'TIPO\_EMPLEADO'. The data consists of four rows:

	NOMBRE_APELLIDO	TELEFONO	EMAIL	TIPO_EMPLEADO
1	Estela Ortuno Sandoval	1116594060	estela.sandoval@gmail.com	PERSONAL_MANTENIMIENTO
2	Eli Rosario Criado Gullen	112233576	eli.gullen@gmail.com	ADMINISTRADOR
3	Matilde Frias Vicente	1146453278	matilde.vicente@gmail.com	PERSONAL_MANTENIMIENTO
4	Jonatan Planas Fortuny	1128270033	jonatan.fortuny@gmail.com	PERSONAL_MANTENIMIENTO

## Triggers

- **Trigger 1): Inserción en reparación, actualización en expensa.**

La utilidad del trigger INS\_REPARACION\_UPD\_EXPENSA es la actualización del atributo expensa al insertar una nueva reparación en una unidad. En nuestro modelo, la tabla EXPENSA contiene el importe total de cada expensa en varias partidas (multa, amenity, gasto mensual, reparación). Cada vez que se registra una nueva reparación en REPARACION, el monto de reparación asociado debe sumarse automáticamente al campo MONTO\_REPARACION de la expensa correspondiente, y simultáneamente ajustar el MONTO\_TOTAL. Si dejáramos este ajuste en manos del usuario (updates manuales por ejemplo) corremos el riesgo de generar inconsistencias. Los olvidos o errores en la suma de montos provocarían

que MONTO\_TOTAL no refleje la realidad, dificultando por ejemplo los reportes financieros.

Por otro lado, al centralizar la lógica en el trigger garantizamos que todo insert en reparaciones quede automáticamente reflejado en las expensas.

También, al incluir el rollback transaction, si ocurre un error en la actualización de la expensa, la inserción de la reparación se revierte y se vuelve al punto inicial. Esto es indispensable para preservar la integridad de los datos.

Es por estas razones que el trigger es esencial. Automatiza la propagación de montos, evitando así tareas manuales, y asegura la consistencia entre las tablas y el registro de las expensas. De esta manera siempre se reflejará con exactitud el impacto de cada reparación en las cuentas de expensas, aumentando la confianza del usuario hacia nosotros.

Código:

```
CREATE TRIGGER INS_REPARACION_UPD_EXPENSA
ON REPARACION
FOR INSERT
AS
BEGIN
    UPDATE E
    SET
        MONTO_REPARACION = E.MONTO_REPARACION + A.MONTO_TOTAL,
        MONTO_TOTAL = E.MONTO_TOTAL + A.MONTO_TOTAL
    FROM EXPENSA E
    JOIN (
```

```

SELECT ID_EXPENSA, SUM(MONTO) AS MONTO_TOTAL
FROM inserted
GROUP BY ID_EXPENSA
) A ON E.ID_EXPENSA = A.ID_EXPENSA;
IF @@ERROR != 0
BEGIN
    RAISERROR ('Error al actualizar monto reparacion en expensa', 16, 1)
    ROLLBACK TRANSACTION
END
END;

```

SELECT \* FROM EXPENSA WHERE EXPENSA.ID\_EXPENSA = 1200

ID_EXPENSA	FECHA	MONTO_MULTA	MONTO_RESERVA_AMENITY	MONTO_GASTO	MONTO_REPARACION	MONTO_TOTAL	ID_GASTO	ID_UNIDAD
1	2025-04-30	0	0	1541540.00	0	1541540.00	120	300

```

INSERT INTO REPARACION VALUES (31, '2025-04-01', 5, 85000, NULL, NULL, 941, 1200, 300)

100 % ▾
Messages
(1 row affected)
(1 row affected)
Completion time: 2025-05-07T21:35:09.1034206-03:00

```

```

SELECT * FROM EXPENSA WHERE EXPENSA.ID_EXPENSA = 1200

100 % ▾
Results Messages
ID_EXPENSA FECHA MONTO_MULTA MONTO_RESERVA_AMENITY MONTO_GASTO MONTO_REPARACION MONTO_TOTAL ID_GASTO ID_UNIDAD
1 1200 2025-04-30 0 0 1541540.00 85000 1626540.00 120 300

```

- **Trigger 2): Inserción de un registro en la tabla administrador\_x\_edificio, actualización del atributo cantidad de edificios en la tabla administrador.**

La utilidad del trigger INS\_ADMIN\_EDI\_UPD\_ADMINISTRADOR es mantener automáticamente el atributo CANTIDAD\_EDIFICIOS en la tabla ADMINISTRADOR, de modo que siempre refleje cuántos edificios vigentes administra cada persona. El trigger se dispara tras la inserción de nuevas asignaciones en ADMINISTRADOR\_X\_EDIFICO. Su principal beneficio es que elimina la necesidad de que la aplicación realice un update manual cada vez que se asigna un edificio a un administrador (es decir, FECHA\_FIN está null), reduciendo así el riesgo de errores u omisiones.

La operatoria del trigger es la siguiente: cuando se inserta uno o varios registros en ADMINISTRADOR\_X\_EDIFICO, la tabla inserted contiene exactamente las filas recién agregadas. Se agrupan los identificadores de los administradores y se calculan cuántos edificios nuevos recibió cada administrador en esa inserción, contabilizando únicamente aquellos edificios cuya fecha de fin es nula. Luego, se actualiza el atributo CANTIDAD\_EDIFICIOS, siendo este la cantidad anterior más la nueva cantidad de edificios que se registraron para ese administrador. De esta manera, el trigger soporta inserciones múltiples en una sola sentencia insert contabilizando correctamente la nueva cantidad de edificios del administrador.

Código:

```
CREATE TRIGGER INS_ADMIN_EDI_UPD_ADMINISTRADOR
ON ADMINISTRADOR_X_EDIFICIO
FOR INSERT
AS
BEGIN
    UPDATE A
    SET CANTIDAD_EDIFICIOS = CANTIDAD_EDIFICIOS + X.CANT
    FROM ADMINISTRADOR A
    JOIN (
        SELECT ID_ADMINISTRADOR, COUNT(*) AS CANT
        FROM inserted
        WHERE FECHA_FIN IS NULL
        GROUP BY ID_ADMINISTRADOR
    ) X ON A.ID_ADMINISTRADOR = X.ID_ADMINISTRADOR
    IF @@ERROR != 0
        BEGIN
            RAISERROR ('ERROR AL ACTUALIZAR LA CANTIDAD DE EDIFICIOS EN
ADMINISTRADOR', 16, 1)
            ROLLBACK TRANSACTION
        END
END
```

GO

```
SELECT * FROM ADMINISTRADOR WHERE ADMINISTRADOR.ID_ADMINISTRADOR = 931
```

	CANTIDAD_EDIFICIOS	EMPRESA	CONTACTO_EMPRESA	ID_ADMINISTRADOR
1	1			931

```
INSERT INTO ADMINISTRADOR_X_EDIFICIO VALUES (GETDATE(),NULL, 1000000,931, 2)
```

(1 row affected)
(1 row affected)
Completion time: 2025-05-07T21:56:19.8796277-03:00

```
SELECT * FROM ADMINISTRADOR WHERE ADMINISTRADOR.ID_ADMINISTRADOR = 931
```

	CANTIDAD_EDIFICIOS	EMPRESA	CONTACTO_EMPRESA	ID_ADMINISTRADOR
1	2			931

- **Trigger 3): Actualización del atributo monto gasto (inicializado en cero) ante una actualización de la tabla gasto mensual.**

El trigger UPD\_GASTO\_UPD\_EXPENSA se utiliza cuando se conozca el gasto mensual de un edificio, se cargará en la tabla GASTO\_MENSUAL que previamente el monto debería estar en 0 (luego de que se insertaron a partir del trigger INS\_GASTO) pero no necesariamente. Al actualizar esta tabla con el monto correspondiente se dispara el trigger para que se actualicen de forma automática todos los montos correspondientes a gastos de ese edificio en la tabla EXPENSA para las unidades de ese edificio de acuerdo al porcentaje del edificio que tenga cada unidad, es decir, cada unidad pagará el % del gasto mensual que es el % que ocupa su unidad en ese edificio. Su principal beneficio es que elimina la necesidad de que se deba realizar muchos updates manualmente cada vez que se actualiza el gasto de un edificio, reduciendo así el riesgo de errores u omisiones y optimizando el uso del tiempo.

La operatoria del trigger es que al actualizar el monto del gasto mensual de uno o muchos edificios se va a hacer un join entre las tablas inserted y expensa y unidad, luego se va a filtrar las unidades que tengan el mismo ID\_GASTO que el o los que se insertaron y se actualizará el MONTO\_GASTO y el MONTO\_TOTAL sumándole el % que aparece en la tabla unidad multiplicado por el gasto total.

Código:

```
CREATE TRIGGER UPD_GASTO_UPD_EXPENSA
ON GASTO_MENSUAL FOR UPDATE AS
BEGIN
    UPDATE EXPENSA
    SET MONTO_GASTO=I.MONTO * U.PORCENTAJE_EDIFICIO,
        MONTO_TOTAL=e.MONTO_MULTA+e.MONTO_RESERVA_AMENITY+I.MONTO *
        U.PORCENTAJE_EDIFICIO+e.MONTO_REPARACION
    FROM INSERTED I
    join EXPENSA E
```

```
on e.ID_GASTO=i.ID_GASTO  
JOIN UNIDAD U  
ON U.ID_UNIDAD=E.ID_UNIDAD  
WHERE U.ID_EDIFICIO=I.ID_EDIFICIO  
  
IF @@error != 0  
  
BEGIN  
  
    RAISERROR ('Error al actualizar cantidad de edificios en  
administrador', 16, 1)  
  
    ROLLBACK TRANSACTION  
  
END  
  
END  
  
;
```

SQLQuery1.sql - 1...GD\_Grupo02M (236)\*

```
select *
from GASTO_MENSUAL
```

Results Messages

ID_GASTO	MONTO	FECHA_GASTO	ID_EDIFICIO
1	1000000	2025-01-31	1
2	4660000	2025-01-31	2
3	4644000	2025-01-31	3
4	4224000	2025-01-31	4
5	3918000	2025-01-31	5
6	18512000	2025-01-31	6
7	7144000	2025-01-31	7
8	6042000	2025-01-31	8
9	10320000	2025-01-31	9
10	6120000	2025-01-31	10
11	8600000	2025-01-31	11
12	10520000	2025-01-31	12

Query executed successfully.

SQLQuery1.sql - 1...GD\_Grupo02M (236)\*

```
select *
from EXPENSA
```

Results Messages

ID_EXPENSA	FECHA	MONTO_MULTA	MONTO_RESERVA_AMENITY	MONTO_GASTO	MONTO_REPARACION	MONTO_TOTAL	ID_GASTO	ID_UNIDAD
1	20250131	0	0	66600.00	0	66600.00	1	1
2	20250131	0	0	33300.00	0	33300.00	1	2
3	20250131	0	0	66600.00	0	66600.00	1	3
4	2025-01-31	0	0	33300.00	0	33300.00	1	4
5	20250131	0	0	66600.00	0	66600.00	1	5
6	20250131	0	0	33300.00	0	33300.00	1	6
7	20250131	0	0	66600.00	0	66600.00	1	7
8	20250131	0	0	33300.00	0	33300.00	1	8
9	20250131	0	0	66600.00	0	66600.00	1	9
10	20250131	0	0	33300.00	0	33300.00	1	10
11	20250131	0	0	58300.00	0	58300.00	1	11
12	20250131	0	0	41600.00	0	41600.00	1	12

Query executed successfully.

The screenshot shows a SQL Server Management Studio interface. The top window is titled "SQLQuery1.sql - 1...GD\_Grupo02M (236)\*" and contains the following SQL code:

```
UPDATE GASTO_MENSUAL
SET MONTO=200000
WHERE ID_EDIFICIO=1 AND FECHA_GASTO='2025-01-31'
```

The bottom window is titled "Messages" and displays the following output:

```
(20 rows affected)
(1 row affected)
Completion time: 2025-05-14T20:04:16.0461724-03:00
```

SQLQuery1.sql - 1...GD\_Grupo02M (236)\*

```
select *
from GASTO_MENSUAL
```

100 %

Results Messages

	ID_GASTO	MONTO	FECHA_GASTO	ID_EDIFICIO
1	1	2000000	2025-01-31	1
2	2	4660000	2025-01-31	2
3	3	4644000	2025-01-31	3
4	4	4224000	2025-01-31	4
5	5	3918000	2025-01-31	5
6	6	18512000	2025-01-31	6
7	7	7144000	2025-01-31	7
8	8	6042000	2025-01-31	8
9	9	10320000	2025-01-31	9
10	10	6120000	2025-01-31	10
11	11	8600000	2025-01-31	11
12	12	10520000	2025-01-31	12

Query executed successfully.

SQLQuery1.sql - 1...GD\_Grupo02M (236)\*

```
select *
from EXPENSA
```

100 %

Results Messages

	ID_EXPENSA	FECHA	MONTO_MULTA	MONTO_RESERVA_AMENITY	MONTO_GASTO	MONTO_REPARACION	MONTO_TOTAL	ID_GASTO	ID_UNIDAD
1	1	2025-01-31	0	0	133200.00	0	133200.00	1	1
2	2	2025-01-31	0	0	66600.00	0	66600.00	1	2
3	3	2025-01-31	0	0	133200.00	0	133200.00	1	3
4	4	2025-01-31	0	0	66600.00	0	66600.00	1	4
5	5	2025-01-31	0	0	133200.00	0	133200.00	1	5
6	6	2025-01-31	0	0	66600.00	0	66600.00	1	6
7	7	2025-01-31	0	0	133200.00	0	133200.00	1	7
8	8	2025-01-31	0	0	66600.00	0	66600.00	1	8
9	9	2025-01-31	0	0	133200.00	0	133200.00	1	9
10	10	2025-01-31	0	0	66600.00	0	66600.00	1	10
11	11	2025-01-31	0	0	116600.00	0	116600.00	1	11
12	12	2025-01-31	0	0	83200.00	0	83200.00	1	12

Query executed successfully.

172.16.0.14 (16.0 RTM) | GD\_Grupo02M (236)

- **Trigger 4): Borrado de un registro en la tabla administrador\_x\_edificio, actualización del atributo cantidad de edificios en la tabla administrador.**

La utilidad del trigger BORRAR\_ADMIN\_X\_EDIF es mantener automáticamente el atributo CANTIDAD\_EDIFICIOS en la tabla ADMINISTRADOR, de modo que siempre refleje cuántos edificios vigentes administra cada persona. El trigger se dispara tras la eliminación de un registro en ADMINISTRADOR\_X\_EDIFICIO. Su principal beneficio es que elimina la necesidad de que la aplicación realice un update manual cada vez que se elimina un edificio a un administrador (es decir, FECHA\_FIN no está null), reduciendo así el riesgo de errores u omisiones.

La operatoria del trigger es la siguiente: cuando se eliminan uno o varios registros en ADMINISTRADOR\_X\_EDIFICIO, la tabla deleted contiene exactamente las filas recién eliminadas. Se agrupan los identificadores de los administradores y se calculan cuántos edificios perdió cada administrador en esa eliminación, contabilizando únicamente aquellos edificios cuya fecha de fin no es nula. Luego, se actualiza el atributo CANTIDAD\_EDIFICIOS, siendo este la cantidad anterior menos la nueva cantidad de edificios que se eliminaron para ese administrador. De esta manera, el trigger soporta borrados múltiples en una sola sentencia delete contabilizando correctamente la nueva cantidad de edificios del administrador.

Código:

```
CREATE TRIGGER BORRAR_ADMIN_X_EDIF
ON ADMINISTRADOR_X_EDIFICIO
FOR DELETE
AS
BEGIN
    UPDATE A
        SET CANTIDAD_EDIFICIOS = CANTIDAD_EDIFICIOS -
X.CANTIDAD_A_RESTAR
    FROM ADMINISTRADOR A
    JOIN (SELECT ID_ADMINISTRADOR, COUNT(*) AS CANTIDAD_A_RESTAR
```

```
FROM deleted

WHERE FECHA_FIN IS NULL

GROUP BY ID_ADMINISTRADOR) X

ON A.ID_ADMINISTRADOR = X.ID_ADMINISTRADOR

IF @@ERROR != 0

BEGIN

    RAISERROR ('ERROR AL ACTUALIZAR LA CANT DE EDFICIOS EN ADMIN',
16, 1)

    ROLLBACK TRANSACTION

END

END

GO
```

SQLQuery2.sql - 1...GD\_Grupo02M (118)\* → SQLQuery1.sql - 1...GD\_Grupo02M (115)\*

```
select * from ADMINISTRADOR
```

100 %

Results Messages

CANTIDAD_EDIFICIOS	EMPRESA	CONTACTO_EMPRESA	ID_ADMINISTRADOR
1	Conocerios del Rio S.A.	1143957382	901
1	Administracion Urbana Integral SRL	1137462183	902
1			903
1	Grupo Agora Administracion	1172310045	904
1			905
1			906
1	Servicios Patrimoniales Norte S.A.	1187643245	907
1			908
1			909
1	Edificio Seguro Gestion Inmobiliaria	1167483092	910
1	Administracion Urbana Integral SRL	1137462183	911
1			912
1			913

Query executed successfully. 172.16.0.14

SQLQuery2.sql - 1...GD\_Grupo02M (118)\* → SQLQuery1.sql - 1...GD\_Grupo02M (115)\*

```
select * from ADMINISTRADOR_X_EDIFICIO
```

100 %

Results Messages

FECHA_INICIO	FECHA_FIN	SUELDO_BRUTO	ID_ADMINISTRADOR	ID_EDIFICIO
2024-09-01	NULL	1097732	901	1
2019-04-12	2021-03-01	1025430	902	2
2025-05-21	NULL	1000000	902	21
2020-08-19	NULL	914690	903	3
2019-12-07	NULL	990285	904	4
2024-08-01	NULL	1116332	905	5
2020-07-18	NULL	1103995	906	6
2024-11-02	NULL	85322	907	7
2020-08-11	NULL	1021783	908	8
2019-08-01	NULL	893758	909	9
2024-06-19	NULL	886426	910	10
2023-03-30	NULL	988567	911	11
2023-10-16	NULL	1094972	912	12

Query executed successfully. 172.16.0.14

SQLQuery3.sql - 1...GD\_Grupo02M (122)\* → SQLQuery2.sql - 1...GD\_Grupo02M (118)\* → SQLQuery1.sql - 1...GD\_Grupo02M (115)\*

```
delete from ADMINISTRADOR_X_EDIFICIO
where ID_ADMINISTRADOR=902 and FECHA_FIN is null
```

100 %

Messages

```
(1 row affected)
(1 row affected)
```

Completion time: 2024-05-21T18:20:30.9351528-03:00

100 %

Query executed successfully. 172.16.0.14 (16.0 RTM) | GD\_Grupo02M (122) | GD\_Grupo02M (118)

```

SQLQuery2.sql - 1...GD_Grupo02M (122)* SQLQuery2.sql - 1...GD_Grupo02M (118)* SQLQuery1.sql - 1...GD_Grupo02M (115)*
select * from ADMINISTRADOR

```

CANTIDAD_EDIFICIOS	EMPRESA	CONTACTO_EMPRESA	ID_ADMINISTRADOR
1	Consortios del Rio S.A.	1143957382	901
2	Administración Urbana Integral SRL	1137462183	902
3			903
4	Grupo Agora Administracion	1172310045	904
5			905
6			906
7	Servicios Patrimoniales Norte S.A.	1187643245	907
8			908
9			909
10	Edificio Seguro Gestión Inmobiliaria	1167483092	910
11	Administración Urbana Integral SRL	1137462183	911
12			912
13			913

Query executed successfully. 172.16.0.14

  

```

SQLQuery3.sql - 1...GD_Grupo02M (122)* SQLQuery2.sql - 1...GD_Grupo02M (118)* SQLQuery1.sql - 1...GD_Grupo02M (115)*
select * from ADMINISTRADOR_X_EDIFICIO

```

FECHA_INICIO	FECHA_FIN	SUELDO_BRUTO	ID_ADMINISTRADOR	ID_EDIFICIO
2024-09-01	NULL	1097732	901	1
2019-04-12	2021-03-01	1025430	902	2
2020-08-19	NULL	914690	903	3
2019-12-07	NULL	990285	904	4
2024-08-01	NULL	1116832	905	5
2020-07-18	NULL	1103895	906	6
2024-11-02	NULL	853222	907	7
2020-08-11	NULL	1021783	908	8
2019-08-01	NULL	893758	909	9
2024-06-19	NULL	886426	910	10
2023-03-30	NULL	988567	911	11
2023-10-16	NULL	1094972	912	12
2023-07-02	NULL	915105	913	13

Query executed successfully. 172.16.0.14 (16.0 RTM) GD\_Grupo02M (115) GD\_Grup

- Trigger 5): Finalización del contrato con el administrador, personal y cierre de unidades al cerrar un edificio.**

La utilidad del trigger CIERRE\_EDIFICIO es que cuando se cierra un edificio, automáticamente se le pone fecha de baja a todos los registros relacionados a ese edificio que no tenían fecha de baja anteriormente, tanto para residentes, administradores y personal de mantenimiento. El trigger se dispara tras la actualización de registros en EDIFICIO cuando se cambia la fecha de baja de null a la de cierre. Su principal beneficio, al igual que el trigger 3, es que elimina la necesidad de que se deba realizar muchos updates manualmente cada vez que se da de baja un edificio, reduciendo así el riesgo de errores u omisiones y optimizando el uso del tiempo.

La operatoria del trigger es que al actualizar un edificio poniéndole fecha de baja, cuando anteriormente esta era nula, se van a disparar 3 sentencias update.

La primera le va a poner fecha de fin a todos los registros de ADMINISTRADOR\_X\_EDIFICIO en los casos donde FECHA\_FIN es nula, la tabla inserted (que hace referencia al registro actualizado de EDIFICIO) tenga FECHA\_BAJA no nula y la tabla deleted (que hace referencia al registro viejo de EDIFICIO) tenga FECHA\_BAJA nula

La segunda hace lo mismo que la primera pero sobre la tabla PERSONAL\_X\_EDIFICIO en vez de ADMINISTRADOR\_X\_EDIFICIO

La tercera hace lo mismo que las anteriores con la diferencia que para poder determinar a qué edificio pertenecen las unidades además se le hace un join a UNIDAD\_X\_RESIDENTE con UNIDAD y en UNIDAD se determina que pertenece al edificio

Cabe destacar que en todas las sentencias update se hace inner join con las tablas deleted e inserted por lo que no hace falta aclarar que los identificadores del edificio coincidan sino que los hace el join directamente.

Código:

```
CREATE TRIGGER CIERRE_EDIFICIO
```

```
ON EDIFICIO
```

```
FOR UPDATE
```

```
AS
```

```
BEGIN
```

```
    UPDATE AE
```

```
        SET FECHA_FIN = GETDATE()
```

```
FROM ADMINISTRADOR_X_EDIFICIO AE  
JOIN inserted i ON AE.ID_EDIFICIO = i.ID_EDIFICIO  
JOIN deleted d ON i.ID_EDIFICIO = d.ID_EDIFICIO  
WHERE AE.FECHA_FIN IS NULL  
AND d.FECHA_BAJA IS NULL  
AND i.FECHA_BAJA IS NOT NULL;
```

```
UPDATE PE  
SET FECHA_FIN = GETDATE()  
FROM PERSONAL_X_EDIFICIO PE  
JOIN inserted i ON PE.ID_EDIFICIO = i.ID_EDIFICIO  
JOIN deleted d ON i.ID_EDIFICIO = d.ID_EDIFICIO  
WHERE PE.FECHA_FIN IS NULL  
AND d.FECHA_BAJA IS NULL  
AND i.FECHA_BAJA IS NOT NULL;
```

```
UPDATE UR  
SET FECHA_HORA_BAJA = GETDATE()  
FROM UNIDAD_X_RESIDENTE UR  
JOIN UNIDAD U ON UR.ID_UNIDAD = U.ID_UNIDAD  
JOIN inserted i ON U.ID_EDIFICIO = i.ID_EDIFICIO
```

```
JOIN deleted d ON i.ID_EDIFICIO = d.ID_EDIFICIO  
WHERE UR.FECHA_HORA_BAJA IS NULL  
AND d.FECHA_BAJA IS NULL  
AND i.FECHA_BAJA IS NOT NULL;  
  
IF @@ERROR != 0  
  
BEGIN  
  
RAISERROR ('ERROR AL ACTUALIZAR UN EDIFICIO', 16, 1)  
  
ROLLBACK TRANSACTION  
  
END  
  
END;
```

```
select *
from EDIFICIO
where ID_EDIFICIO=29
```

SQLQuery4.sql - 1..(GD\_Grupo02M (54)) \* SQLQuery3.sql - 1..(GD\_Grupo02M (63)) \* SQLQuery1.sql - 1..(GD\_Grupo02M (53)) \*

```
select *
from UNIDAD_X_RESIDENTE ur
join UNIDAD u
on u.ID_UNIDAD=ur.ID_UNIDAD
where u.ID_EDIFICIO=29
```

100 %

Results Messages

	FECHA_HORA_ALTA	FECHA_HORA_BAJA	ID_RESIDENTE	ID_UNIDAD	ID_UNIDAD	UNIDAD_FUNCIONAL	PISO	DEPARTAMENTO	TORRE	METROS2	PORCENTAJE_EDIFICIO	ID_EDIFICIO
1	2023-05-16 20:30:00	NULL	9	270	270	5	5	NULL	NULL	82	0.0666	29
2	2021-10-30 21:44:00	NULL	40	268	268	3	3	NULL	NULL	82	0.0666	29
3	2019-04-17 15:56:00	NULL	41	279	279	14	14	NULL	NULL	82	0.0666	29
4	2022-06-29 10:46:00	NULL	83	266	266	1	1	NULL	NULL	82	0.0666	29
5	2018-01-30 14:52:00	NULL	87	269	269	4	4	NULL	NULL	82	0.0666	29
6	2023-02-19 14:27:00	NULL	100	269	269	4	4	NULL	NULL	82	0.0666	29
7	2024-02-11 17:53:00	NULL	113	277	277	12	12	NULL	NULL	82	0.0666	29
8	2016-12-28 15:51:00	NULL	127	275	275	10	10	NULL	NULL	82	0.0666	29
9	2017-05-30 18:28:00	NULL	133	276	276	11	11	NULL	NULL	82	0.0666	29
10	2018-10-28 18:33:00	NULL	136	274	274	9	9	NULL	NULL	82	0.0666	29
11	2022-01-05 11:43:00	NULL	141	270	270	5	5	NULL	NULL	82	0.0666	29
12	2016-07-31 11:06:00	NULL	190	266	266	1	1	NULL	NULL	82	0.0666	29
13	2023-01-06 16:37:00	NULL	215	280	280	15	15	NULL	NULL	82	0.0666	29

Query executed successfully.

172.16.0.14 (16.0 RTM) GD\_Grupo02M (54) GD\_Grupo02M 00:00:00 42 rows

```
SQLQuery5.sql - 1...(GD_Grupo02M (80)) * X SQLQuery4.sql - 1...(GD_Grupo02M (54)) * SQLQuery3.sql - 1...(GD_Grupo02M (63))  
- GD  
select *  
from PERSONAL_X_EDIFICIO  
where ID_EDIFICIO=29
```

Results			
	FECHA CONTRATACION	FECHA FIN	ID EDIFICIO
1	2021-10-19	NULL	29
2	2021-10-09	NULL	960

```
SQLQuery6.sql - 1...(GD_Grupo02M (51)) * X SQLQuery5.sql - 1...(GD_Grupo02M (80)) * SQLQuery4.sql - 1...(GD_Grupo02M (54)) *  
- GD  
select *  
from ADMINISTRADOR_X_EDIFICIO  
where ID_EDIFICIO=29
```

Results				
	FECHA_INICIO	FECHA_FIN	SUELDO_BRUTO	ID_ADMINISTRADOR
1	2024-02-03	NULL	957418	929

SQLQuery5.sql - 1...(GD\_Grupo02M (80))\*

SQLQuery4.sql - 1...(GD\_Grupo02M (54))\*

SQL

```
UPDATE EDIFICIO
SET FECHA_BAJA=GETDATE()
WHERE ID_EDIFICIO=29
```

100 % < Messages

(1 row affected)

(2 rows affected)

(42 rows affected)

(1 row affected)

Completion time: 2025-05-21T11:39:04.9244185-03:00

SQLQuery5.sql - 1...(GD\_Grupo02M (80))\*

SQLQuery4.sql - 1...(GD\_Grupo02M (54))\*

SQLQuery3.sql - 1...(GD\_Grupo02M (63))\*

SQLQuery6.sql - 1...(GD\_Grupo02M (51))\*

```
select *
from EDIFICIO
where ID_EDIFICIO=29
```

100 % < Results Messages

ID_EDIFICIO	FECHA_ALTA	NOMBRE	DIRECCION	CUENTA_CORRIENTE_EDIFICIO	CANTIDAD_UNIDADES	CANTIDAD_TORRES	METROS2_CUBIERTOS	FECHA_BAJA
29	2011-10-11	Torre Rio Plaza	Av. Figuerica Alcorta 6900, Palermo	CC00029	15	NULL	1230	2025-05-21

SQLQuery5.sql - 1...(GD\_Grupo02M (80)) SQLQuery4.sql - 1...(GD\_Grupo02M (54)) X SQLQuery3.sql - 1...(GD\_Grupo02M (63)) SQLQuery6.sql - 1...(GD\_Grupo02M (31))

```

select
  from UNIDAD_X_RESIDENTE ur
  join UNIDAD u
  on u.ID_UNIDAD=ur.ID_UNIDAD
  where u.ID_EDIFICIO=29

```

100 %

Results Messages

	FECHA_HORA_ALTA	FECHA_HORA_BAJA	ID_RESIDENTE	ID_UNIDAD	ID_UNIDAD	UNIDAD_FUNCIONAL	PISO	DEPARTAMENTO	TORRE	METROS2	PORCENTAJE_EDIFICIO	ID_EDIFICIO
1	2023-09-16 20:32:00	2025-05-21 11:38:00	9	270	270	5	5	NULL	NULL	82	0.0666	29
2	2021-10-30 21:44:00	2025-05-21 11:38:00	40	268	268	3	3	NULL	NULL	82	0.0666	29
3	2019-04-06 17:56:00	2025-05-21 11:38:00	41	279	279	14	14	NULL	NULL	82	0.0666	29
4	2022-06-29 10:46:00	2025-05-21 11:38:00	83	266	266	1	1	NULL	NULL	82	0.0666	29
5	2019-01-30 14:52:00	2025-05-21 11:38:00	87	269	269	4	4	NULL	NULL	82	0.0666	29
6	2023-02-19 14:27:00	2025-05-21 11:38:00	100	269	269	4	4	NULL	NULL	82	0.0666	29
7	2024-08-11 17:53:00	2025-05-21 11:38:00	113	277	277	12	12	NULL	NULL	82	0.0666	29
8	2016-12-28 15:51:00	2025-05-21 11:38:00	127	275	275	10	10	NULL	NULL	82	0.0666	29
9	2017-05-30 18:28:00	2025-05-21 11:38:00	133	276	276	11	11	NULL	NULL	82	0.0666	29
10	2018-10-28 18:33:00	2025-05-21 11:38:00	136	274	274	9	9	NULL	NULL	82	0.0666	29
11	2022-01-05 11:43:00	2025-05-21 11:38:00	141	270	270	5	5	NULL	NULL	82	0.0666	29
12	2016-07-31 11:06:00	2025-05-21 11:38:00	190	266	266	1	1	NULL	NULL	82	0.0666	29
13	2023-01-06 16:37:00	2025-05-21 11:38:00	215	280	280	15	15	NULL	NULL	82	0.0666	29

Query executed successfully.

172.16.0.14 (16.0 RTM) | GD\_Grupo02M (54) | GD\_Grupo02M | 00:00:00 | 42 rows

```
SQLQuery3.sql - 1...(GD_Grupo02M (80)) SQLQuery4.sql - 1...(GD_Grupo02M (54)) SQLQuery5.sql - 1...(GD_Grupo02M (63))
select *
from PERSONAL_X_EDIFICIO
where ID_EDIFICIO=29
```

100 %

Results Messages

	FECHA CONTRATACION	FECHA FIN	ID EDIFICIO	ID PERSONAL
1	2021-10-19	2025-05-21	29	940
2	2021-10-09	2025-05-21	29	960

```
SQLQuery5.sql - 1...(GD_Grupo02M (80)) SQLQuery4.sql - 1...(GD_Grupo02M (54)) SQLQuery3.sql - 1...(GD_Grupo02M (63)) SQLQuery
select *
from ADMINISTRADOR_X_EDIFICIO
where ID_EDIFICIO=29
```

100 %

Results Messages

	FECHA INICIO	FECHA FIN	SUELDO BRUTO	ID ADMINISTRADOR	ID EDIFICIO
1	2024-02-03	2025-05-21	957418	929	29

- **Trigger 6): Cuando se inserta a partir de la tercera multa en una unidad para un mismo mes, el monto de la multa se incrementa en distintos porcentajes dependiendo del número de multa.**

La utilidad del trigger INS\_MULTA\_EXTRA es que, al insertar una nueva multa que sea mayor o igual a la tercera, el monto de la multa será: un 20% más si es la tercera multa, un 50% más si es la cuarta multa y el doble a partir de la quinta multa. Su objetivo es sancionar a los infractores frecuentes, generando que la sanción económica sea mayor y, por lo tanto, desincentivando que el residente vuelva a generar muchas multas en el mismo mes. Cuanto más reincide el residente, mayor es la penalidad sin intervención manual. A su vez, el trigger evita que el administrador deba revisar periódicamente el historial de cada unidad para aplicar sanciones acumulativas, liberando tiempo y reduciendo errores humanos.

La operatoria del trigger es la siguiente: en primer lugar, el trigger se activa automáticamente después de cada inserción en la tabla MULTA. Al dispararse, vincula la fila recién agregada (en la tabla inserted) con la propia MULTA mediante el ID\_MULTA, y luego calcula cuántas infracciones acumula la misma unidad dentro del mes y año de la nueva multa: lo hace contando en MULTA todas las filas cuyo ID\_UNIDAD coincide con el de la inserción y cuya fecha comparte el mismo mes y año. Según ese total, actualiza el campo MONTO de la multa recién creada con un recargo progresivo: si la infracción resulta ser la tercera del mes, el importe sube un 20%; si es la cuarta, aumenta un 50%; y a partir de la quinta, se duplica. Las dos primeras multas del mes mantienen su valor original.

Código:

```
CREATE TRIGGER INS_MULTA_EXTRA
```

```
ON MULTA
```

```
FOR INSERT
```

```
AS
```

```
BEGIN
```

```
    UPDATE M
```

```

SET M.MONTO = CASE

    WHEN (SELECT COUNT(*) FROM MULTA MUL JOIN INSERTED INS ON
MUL.ID_UNIDAD          =           INS.ID_UNIDAD          AND
MONTH(INS.FECHA)=MONTH(MUL.FECHA)          AND
YEAR(INS.FECHA)=YEAR(MUL.FECHA)) = 3 THEN M.MONTO * 1.2

    WHEN (SELECT COUNT(*) FROM MULTA MUL JOIN INSERTED INS ON
MUL.ID_UNIDAD          =           INS.ID_UNIDAD          AND
MONTH(INS.FECHA)=MONTH(MUL.FECHA)          AND
YEAR(INS.FECHA)=YEAR(MUL.FECHA)) = 4 THEN M.MONTO * 1.5

    WHEN (SELECT COUNT(*) FROM MULTA MUL JOIN INSERTED INS ON
MUL.ID_UNIDAD          =           INS.ID_UNIDAD          AND
MONTH(INS.FECHA)=MONTH(MUL.FECHA)          AND
YEAR(INS.FECHA)=YEAR(MUL.FECHA)) >= 5 THEN M.MONTO * 2.0

    ELSE M.MONTO

END

FROM MULTA M

JOIN inserted I

ON M.ID_MULTA = I.ID_MULTA;

END

```

Caso tercera multa:

```
INSERT INTO MULTA VALUES (32, 'RUIDOS', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

	ID_MULTA	MOTIVO	MONTO	FECHA	ID_EXPENSA	ID_UNIDAD
1	1	Ruidos molestos	100000	2025-02-18	301	1
2	31	Ruidos molestos	110000	2025-02-26	301	1
3	32	RUIDOS	24000	2025-02-27	301	1

Caso cuarta multa:

```
SQLQuery1.sql - 1...D_Grupo02M (125)*  ↵ X
INSERT INTO MULTA VALUES (33, 'MAL ESTACIONAMIENTO', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

```
161 %  ↵
(1 row affected)
(1 row affected)
Completion time: 2025-05-21T21:04:27.3721877-03:00
```

```
SQLQuery1.sql - 1...D_Grupo02M (125)* + X

INSERT INTO MULTA VALUES (33, 'MAL ESTACIONAMIENTO', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

161 % < Results Messages

	ID_MULTA	MOTIVO	MONTO	FECHA	ID_EXPENSA	ID_UNIDAD
1	1	Ruidos molestos	100000	2025-02-18	301	1
2	31	Ruidos molestos	110000	2025-02-26	301	1
3	32	RUIDOS	24000	2025-02-27	301	1
4	33	MAL ESTACIONAMIENTO	30000	2025-02-27	301	1

Caso quinta multa:

```
SQLQuery1.sql - 1...D_Grupo02M (125)* + X

INSERT INTO MULTA VALUES (35, 'MAL ESTACIONAMIENTO', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

161 % < Results Messages

(1 row affected)

(1 row affected)

Completion time: 2025-05-21T21:05:36.1198103-03:00

```
SQLQuery1.sql - 1...D_Grupo02M (125)* # X

INSERT INTO MULTA VALUES (35, 'MAL ESTACIONAMIENTO', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

Results Messages

ID_MULTA	MOTIVO	MONTO	FECHA	ID_EXPENSA	ID_UNIDAD
1	Ruidos molestos	100000	2025-02-18	301	1
2	Ruidos molestos	110000	2025-02-26	301	1
3	RUIDOS	24000	2025-02-27	301	1
4	MAL ESTACIONAMIENTO	30000	2025-02-27	301	1
5	MAL ESTACIONAMIENTO	40000	2025-02-27	301	1

Caso sexta multa:

```
SQLQuery1.sql - 1...D_Grupo02M (125)* # X

INSERT INTO MULTA VALUES (36, 'RUIDOS MOLESTOS', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

Results Messages

(1 row affected)

(1 row affected)

Completion time: 2025-05-21T21:06:54.2904695-03:00

SQLQuery1.sql - 1...D\_Grupo02M (125)\* X

```
INSERT INTO MULTA VALUES (36, 'RUIDOS MOLESTOS', 20000, '2025-02-27', 301, 1)
GO
SELECT * FROM MULTA WHERE ID_UNIDAD = 1
GO
```

161 % < Results Messages

ID_MULTA	MOTIVO	MONTO	FECHA	ID_EXPENSA	ID_UNIDAD
1	Ruidos molestos	100000	2025-02-18	301	1
2	Ruidos molestos	110000	2025-02-26	301	1
3	RUIDOS	24000	2025-02-27	301	1
4	MAL ESTACIONAMIENTO	30000	2025-02-27	301	1
5	MAL ESTACIONAMIENTO	40000	2025-02-27	301	1
6	RUIDOS MOLESTOS	40000	2025-02-27	301	1