

# Temas

- 1 Sentencia INSERT
  - 1.1 Insertar una sola fila de datos
  - 1.2 Insertar varias filas de datos
  - 1.3 Insertar datos en una tabla con una columna identidad
  - 1.4 Usar TOP para limitar los datos insertados de la tabla origen
  - 1.5 Ejercicio 1
  - 1.6 Ejercicio 2
- 2 Sentencia UPDATE
  - 2.1 Usar una instrucción UPDATE simple
  - 2.2 Actualizar varias columnas
  - 2.3 Usar la cláusula WHERE
  - 2.4 Usar la cláusula TOP
  - 2.5 Usar la cláusula WITH common\_table\_expression
  - 2.6 Especificar una subconsulta en la cláusula SET
  - 2.7 Ejercicio 3
- 3 Sentencia DELETE
  - 3.1 DELETE sin la cláusula WHERE
  - 3.2 Usar la cláusula WHERE para eliminar un conjunto de filas
  - 3.3 Usar la cláusula WHERE con una condición compleja
  - 3.4 Utilizar la cláusula TOP para limitar el número de filas eliminadas
  - 3.5 Ejercicio 4
- 4 Sentencia MERGE
  - 4.1 Usar MERGE para realizar operaciones INSERT y UPDATE en una tabla en una sola instrucción
  - 4.2 Usar MERGE para realizar operaciones UPDATE y DELETE en una tabla en una sola instrucción
- 5 Transacciones
  - 5.1 Definición

## 5.2 Propiedades de una Transacción

### 5.2.1 Atomicidad

### 5.2.2 Coherencia

### 5.2.3 Aislamiento

### 5.2.4 Durabilidad

## 5.3 Tipos de Transacciones

### 5.3.1 Transacciones de confirmación automática

### 5.3.2 Transacciones explícitas

### 5.3.3 Transacciones implícitas

## 1 Sentencia INSERT

```
[ WITH <common_table_expression> [ , ...n ] ]  
INSERT  
[ TOP ( expression ) [ PERCENT ] ]  
[ <OUTPUT Clause> ]  
INTO nombre_tabla [ ( columnas ) ]  
VALUES ( valores ) | Instrucción_SELECT
```

### 1.1 Insertar una sola fila de datos

```
USE RH;  
GO  
  
INSERT INTO DB0.ubicacion  
VALUES('U05','CHICLAYO','Av. Balta 1543 - Cercado');  
GO  
  
SELECT * FROM DB0.ubicacion  
GO
```

No es necesario especificar los nombres de columna en la lista de columnas porque se está suministrando valores para todas las columnas y en el mismo orden que se encuentran en la tabla.

### 1.2 Insertar varias filas de datos

```
USE EDUCA;  
GO  
  
INSERT INTO ALUMNO(alu_nombre, alu_direccion, alu_telefono, alu_email) VALUES  
( 'LLERENA BOLIVAR, PAMELA', 'LA MOLINA', '982354768', 'ollerena@gmail.com'),  
( 'SALAZAR MENDO, AMALIA IRENE', 'MIRAFLORES', NULL, 'asalazar@peru.com'),  
( 'VELASQUEZ RAMOS, MARIA EULALIA', 'SAN BORJA', NULL, 'mvelasquez@gmail.com');  
GO  
  
SELECT * FROM DB0.ALUMNO;  
GO
```

### 1.3 Insertar datos en una tabla con una columna identidad

```
USE EDUCA;  
GO  
  
SET IDENTITY_INSERT DB0.ALUMNO ON;  
GO
```

```
INSERT INTO ALUMNO(alu_id, alu_nombre, alu_direccion, alu_telefono, alu_email)
VALUES(100, 'AYALA FERNANDEZ,
VALERIA', 'SURCO', '875698456', 'vayala@hotmail.com');
GO

SET IDENTITY_INSERT DBO.ALUMNO OFF;
GO

SELECT * FROM DBO.ALUMNO
ORDER BY 1 DESC;
GO
```

## 1.4 Usar TOP para limitar los datos insertados de la tabla origen

```
USE EDUCA;
GO

IF OBJECT_ID (' DBO.ALUMNOS2', ' U') IS NOT NULL
    DROP TABLE DBO.ALUMNOS2;
GO

CREATE TABLE dbo.alumnos2
( codigo    int NOT NULL,
  nombre    varchar(100) NOT NULL,
  email     varchar(50) NOT NULL
);
GO

INSERT TOP (4) INTO DBO.alumnos2(codigo, nombre, email)
SELECT alu_id, alu_nombre, alu_email FROM DBO.ALUMNO;
GO

SELECT * FROM DBO.alumnos2;
GO
```

Para ver las filas que se están insertando debemos utilizar la cláusula OUTPUT:

```
INSERT TOP (4) INTO DBO.alumnos2(codigo, nombre, email)
OUTPUT inserted.codigo, inserted.nombre, inserted.email
SELECT alu_id, alu_nombre, alu_email FROM DBO.ALUMNO;
GO
```

## 1.5 Ejercicio 1

En la base de datos **RH** crear una tabla de nombre **PLANILLA** que permita guardar el importe de la planilla por puesto de trabajo en cada departamento, la información a registrar por fila es la siguiente:

- Código de departamento
- Nombre de departamento
- Código de puesto de trabajo
- Nombre del puesto de trabajo
- Cantidad de trabajadores
- Importe de la planilla sin comisión
- Importe de la planilla con comisión

Luego, construya una sentencia **INSERT** para llenar la tabla **PLANILLA**.

## 1.6 Ejercicio 2

En la base de datos **EDUCA** crear la tabla **RESUMEN** que permita registrar la siguiente información por curso:

- Código del curso
- Nombre del curso
- Cantidad de matriculados
- Importe comprometido según las matriculas
- Importe recaudado según los pagos
- Cantidad de alumnos aprobados (Nota  $\geq 13$ )
- Cantidad de desaprobados (Nota  $< 13$ )
- Cantidad de ausentes (Nota = NULL)

Luego construya una sentencia **INSERT** para llenar la tabla **RESUMEN**.

## 2 Sentencia UPDATE

```
[ WITH <common_table_expression> [...n] ]  
UPDATE  
[ TOP ( expression ) [ PERCENT ] ]  
nombre_tabla  
SET column_name = expression [, . . . ]  
[ <OUTPUT Clause> ]  
[ FROM( <table_source> } [ ,...n ] ]  
[ WHERE <search_condition> ]
```

### 2.1 Usar una instrucción UPDATE simple

En el ejemplo siguiente se actualiza un solo valor de columna para todas las filas de la tabla **dbo.CURSO**.

```
USE EDUCA;  
GO  
  
UPDATE dbo. CURSO  
SET cur_precio = ROUND(cur_precio * 1.10, 0)
```

### 2.2 Actualizar varias columnas

En el siguiente ejemplo se actualizan los valores de las columnas **cur\_vacantes**, y **cur\_precio** para todas las filas de la tabla **CURSO**.

```
USE EDUCA;  
GO  
  
UPDATE dbo. CURSO  
SET cur_vacantes = cur_vacantes + 2,  
    cur_precio = ROUND(cur_precio * 1.10, 0)
```

### 2.3 Usar la cláusula WHERE

En el ejemplo siguiente se utiliza la cláusula **WHERE** para especificar la fila que se va a actualizar. La instrucción actualiza el valor de la columna **cur\_precio** de la tabla **CURSO** para todas las fila que corresponde al curso **SQL Server Administración**, la condición es **cur\_id=2**.

```
USE EDUCA;  
go  
  
SELECT cur_id, cur_precio  
FROM dbo. CURSO where cur_id = 2;  
go
```

```

cur_id      cur_precio
-----
2           1210,00

UPDATE dbo.CURSO
SET cur_precio = 1500.00
WHERE cur_id = 2;

SELECT cur_id, cur_precio
FROM dbo.CURSO where cur_id = 2;
go

cur_id      cur_precio
-----
2           1500,00

```

## 2.4 Usar la cláusula TOP

En los siguientes ejemplos use la cláusula TOP para limitar el número de filas que se modifican en una instrucción UPDATE.

Cuando se usa una cláusula TOP (n) con UPDATE, la operación de actualización se realiza en una selección aleatoria de un número de filas 'n'.

En el siguiente ejemplo se incrementa en un 25 por ciento el sueldo de 5 empleados seleccionados en forma aleatoria. En el resultado, la columna **old** muestra el sueldo antes de la actualización y la columna **new** el sueldo después de la actualización.

```

USE RH;
go

UPDATE TOP (5) dbo.empleado
SET sueldo = sueldo * 1.10
OUTPUT deleted.i dempleado, deleted.suel do old, inserted.suel do new;

i dempleado old          new
-----
E0001      25000,00      27500,00
E0002       8000,00       8800,00
E0003      15000,00      16500,00
E0004       1800,00       1980,00
E0005       7000,00       7700,00

(5 filas afectadas)

```

Si necesita usar TOP para aplicar actualizaciones por orden cronológico, o por algún otro criterio de ordenamiento, debe utilizarla junto con ORDER BY en una subconsulta.

En el siguiente ejemplo se incrementa en un 25 por ciento el sueldo de 5 empleados que tienen el menor salario en la empresa. En el resultado, la columna **old** muestra el sueldo antes de la actualización y la columna **new** el sueldo después de la actualización.

```
USE RH;
go

UPDATE dbo. empleado
SET sueldo = sueldo * 1.10
OUTPUT deleted.i dempleado, deleted.suel do old, inserted.suel do new
FROM (SELECT TOP 5 i dempleado FROM dbo. empleado
      order by sueldo asc) AS t
WHERE dbo. empl eado.i dempl eado = t.i dempl eado;
```

i dempleado	old	new
E0004	1980, 00	2178, 00
E0011	2000, 00	2200, 00
E0018	2000, 00	2200, 00
E0015	2500, 00	2750, 00
E0014	3000, 00	3300, 00

(5 filas afectadas)

## 2.5 Usar la cláusula WITH common\_table\_expression

En el siguiente ejemplo se está agregando la columna **cur\_recaudado** a la tabla **CURSO** para almacenar el importe recaudado por los pagos efectuados de los alumnos, utilizando la cláusula **WITH** se está construyendo una sentencia **SELECT** para obtener el importe recaudado por curso que luego es utilizada en la sentencia **UPDATE**. La cláusula **OUTPUT** muestra los cambios realizados.

```
USE EDUCA;
GO

ALTER TABLE dbo. CURSO
ADD cur_recaudado money NOT NULL DEFAULT 0.0;
GO

WITH recaudado(cur_id, importe) as
(
  SELECT cur_id, sum(pag_importe)
  FROM dbo. PAGO
  GROUP BY cur_id
)
UPDATE dbo. CURSO
SET cur_recaudado = recaudado.importe
OUTPUT deleted.cur_id, deleted.cur_recaudado old, inserted.cur_recaudado new
FROM recaudado
```



```
WHERE dbo.CURSO.cur_id = recaudado.cur_id;
GO
```

cur_id	old	new
1	0, 00	1800, 00
2	0, 00	3310, 00

(2 filas afectadas)

## 2.6 Especificar una subconsulta en la cláusula SET

En el siguiente ejemplo se usa una subconsulta en la cláusula **SET** para determinar el valor que se utilizará para actualizar la columna. La subconsulta debe devolver solo un valor escalar. Es decir, un solo valor.

En el siguiente ejemplo se está utilizando la base de datos EDUCA para crear la tabla INGRESOS, el propósito de esta tabla es almacenar el importe de la suma de los ingresos por curso. Se está utilizando una subconsulta para actualizar la columna **importe**.

```
use EDUCA;
go
```

```
IF OBJECT_ID('INGRESOS', 'U') IS NOT NULL
    DROP TABLE INGRESOS;
GO
```

```
SELECT cur_id, cur_nombre, cast(0.0 as money) as importe
INTO dbo.INGRESOS
FROM dbo.CURSO;
go
```

```
SELECT * FROM dbo.INGRESOS;
go
```

cur_id	cur_nombre	importe
3	Inteligencia de Negocios	0. 00
6	Java Cliente-Servidor	0. 00
5	Java Fundamentos	0. 00
4	Programación Transact-SQL	0. 00
2	SQL Server Administración	0. 00
1	SQL Server Implementación	0. 00

(6 filas afectadas)

```
update dbo.INGRESOS
set importe = (select SUM(pag_importe)
from dbo.PAGO
where dbo.INGRESOS.cur_id = dbo.PAGO.cur_id);
```

```
go
```

```
SELECT * FROM dbo. INGRESOS;
```

```
go
```

cur_id	cur_nombre	importe
3	Inteligencia de Negocios	NULL
6	Java Cliente- Servidor	NULL
5	Java Fundamentos	NULL
4	Programación Transact-SQL	NULL
2	SQL Server Administración	3310.00
1	SQL Server Implementación	1800.00

```
(6 filas afectadas)
```

### Reto

Cuál sería la modificación a la sentencia UPDATE para que no grabe valores nulos en la columna **importe**.

## 2.7 Ejercicio 3

En la base de datos RH, a la tabla cargo agregarle una columna de nombre **EMPS**, luego utilizando una sentencia UPDATE con subconsulta en esta columna **EMPS** debe guardar la cantidad de empleados por cargo.

## 3 Sentencia DELETE

```
[ WITH <common_table_expression> [ , ... n ] ]
DELETE
[ TOP ( expression ) [ PERCENT ] ]
[ FROM ] nombre_tabla
[ <OUTPUT Clause> ]
[ FROM { <table_source> } [ , ... n ] ]
[ WHERE <search_condition> ]
```

### 3.1 DELETE sin la cláusula WHERE

En el ejemplo siguiente se eliminan todas las filas de la tabla **alumno2** porque no se utiliza una cláusula WHERE para limitar el número de filas eliminadas.

```
USE EDUCA;
```

```
GO
```

```
DELETE FROM dbo. alumnos2;
```

```
GO
```

### 3.2 Usar la cláusula WHERE para eliminar un conjunto de filas

En el siguiente ejemplo se desarrolla en la base de datos EDUCA, y lo primero que se está realizando es crear una tabla de cursos llamada CURSOS2 con todas las filas de la tabla CURSO para hacer la demostración.

De la tabla CURSOS2 se está eliminando todos los cursos que no tienen ningún alumno matriculado. Finalmente se está eliminando la tabla CURSOS2.

```
USE EDUCA;
```

```
go
```

```
IF OBJECT_ID(' CURSOS2' , ' U' ) IS NOT NULL
```

```
    DROP TABLE CURSOS2;
```

```
GO
```

```
SELECT * INTO dbo. CURSOS2 FROM dbo. CURSO;
```

```
go
```

```
SELECT cur_id, cur_nombre, cur_matriculados FROM dbo. CURSOS2;
```

```
go
```

cur_id	cur_nombre	cur_matriculados
1	SQL Server Implementación	3
2	SQL Server Administración	5
3	Inteligencia de Negocios	0
4	Programación Transact-SQL	0
5	Java Fundamentos	0
6	Java Cliente-Servidor	0

(6 filas afectadas)

```
DELETE FROM dbo. CURSOS2 WHERE cur_matriculados = 0;
```

```
go
```

(4 filas afectadas)

```
SELECT cur_id, cur_nombre, cur_matriculados FROM dbo. CURSOS2;
```

```
go
```

cur_id	cur_nombre	cur_matriculados
1	SQL Server Implementación	3
2	SQL Server Administración	5

```
DROP TABLE dbo. CURSOS2;
```

```
go
```

### 3.3 Usar la cláusula WHERE con una condición compleja

El siguiente ejemplo es muy similar al anterior, la diferencia está en que se eliminan los cursos que no tienen ningún alumno matriculado y que además no tienen profesor asignado, además se está mostrando las filas eliminadas.

```
USE EDUCA;
go

IF OBJECT_ID('CURS02','U') IS NOT NULL
    DROP TABLE CURS02;
GO

select * into dbo.CURS02 from dbo.CURS0;
go

select cur_id, cur_nombre, cur_profesor, cur_matriculados from dbo.curso2;
go
```

cur_id	cur_nombre	cur_profesor	cur_matriculados
1	SQL Server Implementación	Gustavo coronel	3
2	SQL Server Administración	Gustavo coronel	5
3	Inteligencia de Negocios	Sergio Matsukawa	0
4	Programación Transact-SQL	NULL	0
5	Java Fundamentos	Gustavo Coronel	0
6	Java Cliente-Servidor	Gustavo Coronel	0

(6 filas afectadas)

```
delete from dbo.CURS02
output deleted.cur_id, deleted.cur_profesor, deleted.cur_matriculados
where cur_matriculados = 0 AND cur_profesor is null;
go
```

cur_id	cur_profesor	cur_matriculados
4	NULL	0

(1 filas afectadas)

```
select cur_id, cur_nombre, cur_profesor, cur_matriculados from dbo.curso2;
go
```

cur_id	cur_nombre	cur_profesor	cur_matriculados
1	SQL Server Implementación	Gustavo coronel	3
2	SQL Server Administración	Gustavo coronel	5
3	Inteligencia de Negocios	Sergio Matsukawa	0
5	Java Fundamentos	Gustavo Coronel	0

```

6          Java Cliente-Servidor          Gustavo Coronel          0

(5 filas afectadas)

DROP TABLE dbo. CURS02;
go

```

### 3.4 Utilizar la cláusula TOP para limitar el número de filas eliminadas

#### Caso 1

En el siguiente ejemplo se está eliminado 5 empleados de manera aleatoria de la tabla EMP2 que se está creando para propósitos de la demostración en la base de datos RH.

```

use rh;
go

IF OBJECT_ID('EMP2', 'U') IS NOT NULL
    DROP TABLE EMP2;
GO

select * into dbo.emp2 from dbo.empleado;
go

select COUNT(*) emps from dbo.emp2;
go

emps
-----
22

(1 filas afectadas)

delete top (5) from dbo.emp2
output deleted.i empleado, deleted.nombre;
go

i empleado nombre
-----
E0001      Gustavo
E0002      Claudia
E0003      Sergio
E0004      Mariela
E0005      Roberto

(5 filas afectadas)

select COUNT(*) emps from dbo.emp2;
go

emps

```

```
-----
17
```

```
(1 filas afectadas)
```

```
drop table dbo.emp2;
go
```

## Caso 2

Este caso es similar al Caso 1, la diferencia está en que se eliminan los 5 empleados que tienen los mejores sueldos.

```
use rh;
go
```

```
IF OBJECT_ID('EMP2', 'U') IS NOT NULL
    DROP TABLE EMP2;
GO
```

```
select * into dbo.emp2 from dbo.empleado;
go
```

```
select COUNT(*) emps from dbo.emp2;
go
```

```
emps
```

```
-----
22
```

```
(1 filas afectadas)
```

```
delete from dbo.emp2
output deleted.idempleado, deleted.nombre, deleted.suel do
where idempleado in ( select top 5 idempleado
                      from dbo.emp2 order by suel do desc );
go
```

id empleado	nombre	suel do
E0001	Gustavo	25000.00
E0012	Hugo	15000.00
E0003	Sergio	15000.00
E0009	Ricardo	15000.00
E0016	Nora	15000.00

```
(5 filas afectadas)
```

```
select COUNT(*) emps from dbo.emp2;
go
```

```
emp2
-----
17

(1 filas afectadas)

drop table dbo.emp2;
go
```

### 3.5 Ejercicio 4

En la base de datos RH crear una tabla de empleados auxiliar **EMP2** con todo el contenido de la tabla **EMPLEADO** para desarrollar este ejercicio.

Luego proceda a eliminar de la tabla **EMP2** todos los empleados cuyo sueldo se encuentra fuera del rango según el cargo que desempeña.

---

## 4 Sentencia MERGE

```
[ WITH <common_table_expression> [, ... n] ]
MERGE
    [ TOP ( expression ) [ PERCENT ] ]
    [ INTO ] <target_table> [ [ AS ] table_alias ]
    USING <table_source>
    ON <merge_search_condition>
    [ WHEN MATCHED [ AND <clause_search_condition> ]
      THEN <merge_matched> ] [ ... n ]
    [ WHEN NOT MATCHED [ BY TARGET ] [ AND <clause_search_condition> ]
      THEN <merge_not_matched> ]
    [ WHEN NOT MATCHED BY SOURCE [ AND <clause_search_condition> ]
      THEN <merge_matched> ] [ ... n ]
    [ <output_clause> ] ;
```

### 4.1 Usar MERGE para realizar operaciones INSERT y UPDATE en una tabla en una sola instrucción

Ejecute el archivo NUEVOS\_ALUMNOS.SQL que le será proporcionado por el profesor, este archivo crea una tabla NUEVOS\_ALUMNOS en la base de datos EDUCA.

La tabla NUEVOS\_ALUMNOS tiene datos de alumnos que deben ser insertados en la tabla ALUMNO, pero algunos de ellos ya están registrados, de los que ya están registrados se debe actualizar las columnas **alu\_direccion** y **alu\_telefono**, la columna que se debe verificar para saber si se debe hacer un **INSERT** o **UPDATE** es **alu\_email**.

```
USE EDUCA;
GO

SELECT * FROM dbo.ALUMNO;
```

go

	alu_id	alu_nombre	alu_direccion	alu_telefono	alu_email
1	1	YESENIA VIRHUEZ	LOS OLIVOS	986412345	yesenia@hotmail.com
2	2	OSCAR ALVARADO FERNANDEZ	MIRAFLORES	NULL	oscar@gmail.com
3	3	GLADYS REYES CORTIJO	SAN BORJA	875643562	gladys@hotmail.com
4	4	SARA RIEGA FRIAS	SAN ISIDRO	NULL	sara@yahoo.com
5	5	JHON VELASQUEZ DEL CASTILLO	LOS OLIVOS	78645345	jhon@mivistar.com
6	6	LLERENA BOLIVAR, PAMELA DEL ROSARIO	LA MOLINA	982354768	ollerena@gmail.com
7	7	SALAZAR MENDO, AMALIA IRENE	MIRAFLORES	NULL	asalazar@peru.com
8	8	VELASQUEZ TORVISCO, MARIA EULALIA	SAN BORJA	NULL	mvelasquez@gmail.com

```
SELECT * FROM dbo. NUEVOS_ALUMNOS;
GO
```

	alu_nombre	alu_direccion	alu_telefono	alu_email
1	YESENIA VIRHUEZ	LA MOLINA	897678567	yesenia@hotmail.com
2	GLADYS REYES CORTIJO	SAN MIGUEL	456879023	gladys@hotmail.com
3	GABRIEL FLORES ARROYO	SAN MIGUEL	435679456	gabriel@gmail.com
4	LUIS ROJAS CASTRO	LOS OLIVOS	546784768	lrojas@hotmail.com
5	WILLY SANCHEZ CACHAY	SAN ISIDRO	345879567	wsanchez@gmail.com
6	SANDRA SOLER GARCIA	SURCO	967435672	ssoler@gmail.com

```
MERGE INTO dbo.alumno AS target
USING (
    SELECT alu_nombre, alu_direccion, alu_telefono, alu_email
    FROM dbo.nuevos_alumnos
) AS source(nombre, direccion, telefono, email)
ON (target.alu_email = source.email)
WHEN MATCHED THEN
    UPDATE SET
        alu_direccion = source.direccion,
        alu_telefono = source.telefono
WHEN NOT MATCHED THEN
    INSERT (alu_nombre, alu_direccion, alu_telefono, alu_email)
    VALUES (source.nombre, source.direccion, source.telefono, source.email)
OUTPUT deleted.*, $action, inserted.*;
```

En el resultado que usted verá en podrá comprobar que algunas filas se actualizaron y otras filas se insertaron como nuevas.

```
SELECT * FROM dbo.ALUMNO;
go
```



	alu_id	alu_nombre	alu_direccion	alu_telefono	alu_email
1	1	YESENIA VIRHUEZ	LA MOLINA	897678567	yesenia@hotmail.com
2	2	OSCAR ALVARADO FERNANDEZ	MIRAFLORES	NULL	oscar@gmail.com
3	3	GLADYS REYES CORTIJO	SAN MIGUEL	456879023	gladys@hotmail.com
4	4	SARA RIEGA FRIAS	SAN ISIDRO	NULL	sara@yahoo.com
5	5	JHON VELASQUEZ DEL CASTILLO	LOS OLIVOS	78645345	jhon@mivistar.com
6	6	LLERENA BOLIVAR, PAMELA DEL ROSARIO	LA MOLINA	982354768	ollerena@gmail.com
7	7	SALAZAR MENDO, AMALIA IRENE	MIRAFLORES	NULL	asalazar@peru.com
8	8	VELASQUEZ TORVISCO, MARIA EULALIA	SAN BORJA	NULL	mvelasquez@gmail.com
9	101	GABRIEL FLORES ARROYO	SAN MIGUEL	435679456	gabriel@gmail.com
10	102	LUIS ROJAS CASTRO	LOS OLIVOS	546784768	lrojas@hotmail.com
11	103	SANDRA SOLER GARCIA	SURCO	967435672	ssoler@gmail.com
12	104	WILLY SANCHEZ CACHAY	SAN ISIDRO	345879567	wsanchez@gmail.com

En el listado anterior también se puede verificar las nuevas filas, así como las actualizaciones realizadas.

## 4.2 Usar MERGE para realizar operaciones UPDATE y DELETE en una tabla en una sola instrucción

En el siguiente ejemplo utilizaremos la base de datos **RH**, la demostración se realizará sobre una tabla auxiliar llamada **EMP2**, que inicialmente contiene los mismos datos que la tabla **EMPLEADO**.

Se trata de encontrar el sueldo promedio por departamento, luego debemos eliminar de la tabla EMP2 los empleados que tienen su sueldo menor que el sueldo promedio en su departamento, y los que quedan su sueldo debe incrementarse en 30%.

```
USE RH;
GO

IF OBJECT_ID('EMP2', 'U') IS NOT NULL
    DROP TABLE EMP2;
GO

SELECT * INTO DBO. emp2 FROM DBO. empleado;
GO

MERGE INTO dbo.EMP2 AS target
USING (
    SELECT iddepartamento , avg(suel do)
    FROM dbo. empleado
    GROUP BY iddepartamento
) AS source(dpto, sueldo_prom)
ON (target.iddepartamento = source. dpto)
WHEN MATCHED and target. sueldo < source. sueldo_prom THEN
    DELETE
WHEN  MATCHED THEN
    UPDATE SET
        target. sueldo = target. sueldo * 1. 30
OUTPUT deleted. idempleado, deleted. iddepartamento, deleted. sueldo,
        Saction, inserted. idempleado, inserted. iddepartamento, inserted. sueldo;
```

```
DROP TABLE DB0. EMP2;
```

A continuación parte del resultado de la sentencia MARGE, se puede apreciar que para algunas filas se ejecuta la sentencia DELETE y para otras la sentencia UPDATE.

	idempleado	iddepartamento	sueldo	\$action	idempleado	iddepartamento	sueldo
5	E0011	101	2000,00	DELETE	NULL	NULL	NULL
6	E0003	102	16500,00	UPDATE	E0003	102	21450,00
7	E0004	102	1980,00	DELETE	NULL	NULL	NULL
8	E0005	102	7700,00	UPDATE	E0005	102	10010,00
9	E0006	102	7500,00	UPDATE	E0006	102	9750,00
10	E0007	102	7000,00	DELETE	NULL	NULL	NULL
11	E0008	102	3500,00	DELETE	NULL	NULL	NULL
12	E0016	103	15000,00	UPDATE	E0016	103	19500,00
13	E0017	103	7500,00	UPDATE	E0017	103	9750,00
14	E0018	103	2000,00	DELETE	NULL	NULL	NULL
15	E0019	103	3500,00	DELETE	NULL	NULL	NULL
16	E0020	103	3000,00	DELETE	NULL	NULL	NULL

## 5 Transacciones

### 5.1 Definición

Una transacción es un grupo de acciones que hacen transformaciones consistentes en las tablas preservando la consistencia de la base de datos. Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y eliminaciones de información. Por supuesto, se quiere asegurar que la base de datos nunca entre en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.



Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una

transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

## 5.2 Propiedades de una Transacción

Una transacción debe tener las propiedades ACID, que son las iniciales en inglés de las siguientes características: Atomicity, Consistency, Isolation, Durability.

### 5.2.1 Atomicidad

Una transacción constituye una unidad atómica de ejecución y se ejecuta exactamente una vez; o se realiza todo el trabajo o nada de él en absoluto.

### 5.2.2 Coherencia

Una transacción mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado coherente de datos. Los datos enlazados por una transacción deben conservarse semánticamente.

### 5.2.3 Aislamiento

Una transacción es una unidad de aislamiento y cada una se produce aislada e independientemente de las transacciones concurrentes. Una transacción nunca debe ver las fases intermedias de otra transacción.

### 5.2.4 Durabilidad

Una transacción es una unidad de recuperación. Si una transacción tiene éxito, sus actualizaciones persisten, aun cuando falle el equipo o se apague. Si una transacción no tiene éxito, el sistema permanece en el estado anterior antes de la transacción.

## 5.3 Tipos de Transacciones

### 5.3.1 Transacciones de confirmación automática

El modo de confirmación automática es el modo de administración de transacciones predeterminado de SQL Server. Cada instrucción SQL se confirma o se deshace cuando finaliza. Una conexión de SQL Server funciona en modo de confirmación automática siempre que este modo predeterminado no haya sido sustituido por transacciones explícitas.

Una conexión de SQL Server funciona en modo de confirmación automática hasta que una instrucción **BEGIN TRANSACTION** inicia una transacción explícita. Cuando la transacción explícita se confirma o revierte, SQL Server vuelve al modo de confirmación automática.

### 5.3.2 Transacciones explícitas

Una transacción explícita es aquella en la que se definen explícitamente el inicio y el final de la transacción.

Las instrucciones SQL que se utilizan son las siguientes:

**BEGIN TRANSACTION**

Marca el punto de inicio de una transacción explícita para una conexión.

### COMMIT TRANSACTION

Finaliza correctamente una transacción si no se han encontrado errores. Todos los datos modificados por la transacción se convierten en parte permanente de la base de datos. Se liberan los recursos ocupados por la transacción.

### ROLLBACK TRANSACTION

Borra una transacción en la que se han encontrado errores. Todos los datos modificados por la transacción vuelven al estado en el que estaban al inicio de la transacción. Se liberan los recursos ocupados por la transacción.

## 5.3.3 Transacciones implícitas

Cuando una conexión funciona en modo de transacciones implícitas, SQL Server Database Engine (Motor de base de datos de SQL Server) inicia automáticamente una nueva transacción después de confirmar o revertir la transacción actual. No tiene que realizar ninguna acción para establecer el inicio de una transacción, sólo tiene que confirmar o revertir cada transacción. El modo de transacciones implícitas genera una cadena continua de transacciones.

Tras establecer el modo de transacciones implícitas en una conexión, la instancia de Motor de base de datos inicia automáticamente una transacción la primera vez que ejecuta una de las siguientes sentencias:

ALTER TABLE	INSERT
CREATE	OPEN
DELETE	REVOKE
DROP	SELECT
FETCH	TRUNCATE TABLE
GRANT	UPDATE

La transacción sigue activa hasta que se ejecute una instrucción **COMMIT** o **ROLLBACK**. Una vez que la primera transacción se ha confirmado o revertido, la instancia del Motor de base de datos inicia automáticamente una nueva transacción la siguiente vez que la conexión ejecuta una de estas instrucciones. La instancia continúa generando una cadena de transacciones implícitas hasta que se desactiva el modo de transacciones implícitas.

La sentencia SET IMPLICIT\_TRANSACTIONS activa o desactiva el modo de transacciones implícitas, su sintaxis es:

```
SET IMPLICIT_TRANSACTIONS { ON | OFF }
```