

Ejercicios Prácticos

Ejercicio Práctico 3.1

Escriba las sentencias para la creación de las tablas utilizadas en el ejercicio conceptual 2.12

Habrá que decidir interpretando los contenidos si los campos podrán almacenar o no valores nulos. En cualquier caso resultará instructivo tomar una decisión sobre cuáles serán las claves primarias y hacerla explícita en la resolución de este ejercicio.

SE RESUELVE PARA CASO 1

Se desea registrar la información relevante para la gestión de una empresa de transportes dedicada a repartir paquetes por todo el país. Los mismos conductores del vehículo son los encargados de llevar los paquetes.

Necesitamos guardar los datos del conductor de cada camión:

- dni
- nombre
- teléfono
- dirección
- salario
- ciudad de residencia

De los paquetes transportados interesa conocer:

- identificador único del paquete
- descripción
- destinatario
- dirección del destinatario
- remitente
- dirección del remitente

Un camionero distribuye muchos paquetes por día mientras que un paquete sólo

puede ser distribuido por un único camionero.

De los camiones que conducen los camioneros, interesa conocer:

- la patente o matrícula
- modelo
- marca
- potencia del motor

Un camionero puede manejar diferentes camiones en fechas diferentes, y un camión puede ser conducido por varios camioneros a lo largo de su vida. Interesa poder saber que camionero estaba conduciendo que camión un día concreto.

USANDO MYSQL

```
create table conductores (  
dni int primary key,  
nombre varchar(100) not null,  
telefono varchar(50) not null,  
direccion varchar(50),  
salario decimal(10,2) not null,  
ciudad_residencia varchar(50)  
);
```

```
create table paquetes (  
id int auto_increment primary key,  
descripcion varchar(100) not null,  
destinatario varchar(100) not null,  
direccion_destinatario varchar(100) not null,  
remitente varchar(50) not null,  
direccion_remitente varchar(100) not null  
);
```

```
create table camiones (  
patente varchar(50) primary key,  
modelo varchar(100),  
marca varchar(100),  
potencia_motor int  
);
```

Ejercicio Práctico 3.2

Identifique una clave primaria para cada tabla creada en el ejercicio 3.1 Genere un índice "clusterizado" para cada tabla incluyendo todos los campos de la clave primaria.

En MySQL, un índice clustered (o índice agrupado) no es algo que se pueda crear explícitamente, porque MySQL utiliza índices agrupados automáticamente para las tablas InnoDB. **En InnoDB, el índice primario (PRIMARY KEY) siempre es un índice clustered.** Si no se define una clave primaria, InnoDB buscará la primera clave única no nula y la usará como índice clustered. Si no se encuentra ninguna clave única, InnoDB generará una clave interna y la usará como índice clustered.

Las PRIMARY KEY y a su vez los ÍNDICES CLÚSTER de cada tabla son:

- conductores: dni
- paquetes: id
- camiones: patente

Ejercicio Práctico 3.3

Para cada una de las tablas del ejercicio 3.2 elija un atributo que no forme parte de la clave primaria y genere un índice NO "clusterizado"

-- conductores

```
CREATE INDEX idx_nombre ON conductores(nombre);
```

-- paquetes

```
CREATE INDEX idx_descripcion ON paquetes(descripcion);
```

-- camiones

```
CREATE INDEX idx_modelo ON camiones(modelo);
```

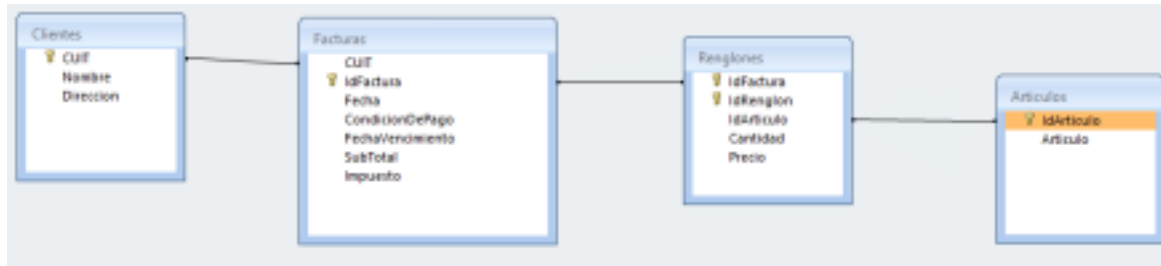
Ejercicio Práctico 3.4:

Crear una vista muestra la tabla clientes que están asignados al vendedor PEPE

```
create view vendedorPepe as select * from clientes where vendedor="Pepe";
```

Ejercicio Práctico 3.5:

Dado el esquema que se muestra:



Crear una vista donde se vea el CUIT, el IdFactura y el total de la factura

```
CREATE VIEW vista AS SELECT CUIT, IdFactura, sum(SubTotal+Impuesto) AS 'Total'
FROM Facturas;
```

Ejercicio Práctico 3.6:

Consultar la vista generada en el ejercicio 3.4

```
SELECT * FROM vendedorpepe;
```

Ejercicio Práctico 3.7:

Obtener la lista de los cuits de las facturas a partir de la vista generada en el ejercicio práctico 3.5

```
SELECT CUIT FROM vista;
```

Ejercicio Práctico 3.8:

Escribir las sentencias que eliminan las vistas generadas en los ejercicios prácticos 3.4 y 3.5

```
DROP VIEW vendedorPepe, vista
```

Ejercicio Práctico 3.9:

Inspirándose en la vista que generamos como ejemplo para obtener los clientes cuya fecha de alta pertenece al año 2018 armar un procedimiento almacenado que devuelva lo mismo que la vista.

```
CREATE PROCEDURE [dbo].[fechaAlta]
AS
BEGIN
SELECT * FROM clientes WHERE YEAR(FechaAlta)=2018
END
```

Ejercicio Práctico 3.10:

Incorporar un parámetro al ejercicio 3.9 para que el procedimiento almacenado sirva para cualquier año para el que se lo quiera usar.

```
CREATE PROCEDURE [dbo].[fechaAltaParametro]
@anio int
AS
BEGIN
SELECT * FROM clientes WHERE YEAR(FechaAlta)=@anio
END
```

Ejercicio Práctico 3.11:

Escribir la sentencia para ejecutar los procedimientos almacenados creados en los ejercicios 3.9 y 3.10.

```
EXEC dbo.fechaAlta
EXEC dbo.fechaAltaParametro 2018
```

Ejercicio Práctico 3.12:

Escribir las sentencias necesarias para eliminar los procedimientos creados en los ejercicios prácticos 3.9 y 3.10

```
DROP PROCEDURE dbo.fechaAlta
DROP PROCEDURE dbo.fechaAltaParametro
```

Ejercicio Práctico 3.13:

A. Usar una función para calcular la raíz cuadrada de 4

B. Usar una función para calcular el seno de 90 grados (recordar que 90 grados es $\text{Pi}/2$)

C. Usar una función para calcular el logaritmo natural de 10

D. Usar una función para redondear $1/3$ a tres decimales

E. Usar una función para calcular 5 al cubo

F. Usar una función para calcular la tangente de 45 grados

-- A

```
SELECT SQRT(4);
```

-- B

```
SELECT SIN(PI()/2);
```

-- C

```
SELECT LOG(10);
```

-- D

```
SELECT ROUND(0.33333,3);
```

-- E

```
SELECT POWER(5,3);
```

-- F

```
SELECT TAN(PI()/4);
```

G. Usar una función para calcular el coseno de 180 grados

H. Usar una función para calcular la raíz quinta de 32 (recordar que para calcular la raíz quinta se puede elevar a la $1/5$)

I. Calcular el logaritmo de 5, el logaritmo de 3 y sumarlos. Luego calcular el logaritmo de 15. ¿Les recuerda alguna propiedad de los logaritmos?

J. Calcular el logaritmo de 25 y el logaritmo de 5. ¿Algún recuerdo de lo que pasa con el logaritmo de una potencia?

-- G

```
SELECT COS(PI());
```

-- H

```
SELECT POWER(32,0.2);
```

-- I

```
SELECT LOG(5)+LOG(3);  
SELECT LOG(15);
```

-- J

```
SELECT LOG(25);  
SELECT LOG(5);
```

Ejercicio Práctico 3.14:

A. Obtener los 5 primeros caracteres de la cadena

"ABCDEFGHIJ"

B. Obtener los 5 últimos caracteres de la cadena "ABCDEFGHIJ"

C. Convertir a minúsculas la cadena "ABCDEFGHIJ"

D. Convertir a mayúsculas el resultado de C.

E. Obtener los caracteres 3ro, 4to y 5to de la cadena

"ABCDEFGHIJ"

F. Obtener una cadena que repita veinte veces la letra "A"

G. Tomar la cadena "MAMA" y reemplazar cada "M" por una "C".
(Prohibido protestar si la respuesta les huele mal)

H. Eliminar los espacios en blanco a derecha de la cadena " ABC "

I. Eliminar los espacios en blanco a izquierda de la cadena " ABC "

J. Eliminar los espacios en blanco a derecha y a izquierda en la cadena " ABC "

-- A

```
SELECT LEFT('ABCDEFGHIJ',5);
```

-- B

```
SELECT RIGHT('ABCDEFGHIJ',5);
```

-- C

```
SELECT LOWER('ABCDEFGHIJ');
```

-- D

```
SELECT UPPER(LOWER('ABCDEFGHIJ'));
```

-- E

```
SELECT SUBSTRING('ABCDEFGHIJ',3,3);
```


-- F

```
SELECT REPLICATE('A',20);
```

-- G

```
SELECT REPLACE('MAMA','M','C');
```

-- H

```
SELECT RTRIM(' ABC ');
```

-- I

```
SELECT LTRIM(' ABC ');
```

-- J

```
SELECT TRIM(' ABC ');
```

Ejercicio Práctico 3.15:

A. Obtenga la fecha actual

B. Obtenga el año de la fecha actual

C. Obtenga el mes de la fecha actual

D. Obtenga el día de la fecha actual

E. Obtenga la fecha correspondiente a 7 días en el futuro de la fecha actual

F. Obtenga el mes de 30 días en el futuro de la fecha actual

G. Obtenga la fecha correspondiente a un año en el futuro desde la fecha actual

H. Obtenga la fecha correspondiente a 30 años en el futuro desde la fecha de su cumpleaños.

I. Obtenga el año correspondiente a 30 años después de la fecha de su cumpleaños.

J. Obtenga el mes correspondiente a 9 meses antes de su fecha de cumpleaños.

-- A

```
SELECT GETDATE();
```

-- B

```
SELECT YEAR(GETDATE());
```

-- C

```
SELECT MONTH(GETDATE());
```

-- D

```
SELECT DAY(GETDATE());
```

-- E

```
SELECT DATEADD(day,7,getdate());
```

```
-- F  
SELECT MONTH(DATEADD(day,30,getdate()));
```

```
-- G  
SELECT DATEADD(year,1,getdate());
```

```
-- H  
SELECT DATEADD(year,30,'1993-03-19');
```

```
-- I  
SELECT YEAR(DATEADD(year,30,'1993-03-19'));
```

```
-- J  
SELECT MONTH(DATEADD(MONTH,-9,'1993-03-19'));
```

Ejercicio Práctico 3.16:

Averigüe si el año 2000 fue bisiesto.

Aplique el código de la función y vea cuantos días obtiene para febrero de 2000. Investigue (por ejemplo en la wikipedia sobre los años bisiestos y verá que hace falta corregir la regla que hemos desarrollado para los años que son múltiplos de 1000)

A. Escriba la regla en castellano

B. Genere el diagrama de flujo correspondiente y pruebe de marcar el año 2000 en amarillo y repetir las marcas para los años del ejercicio Conceptual 3.3

C. Escriba el código correspondiente al diagrama de flujo desarrollado

El año 2000 fue bisiesto porque es múltiplo de 4, de 100 y 400.

A

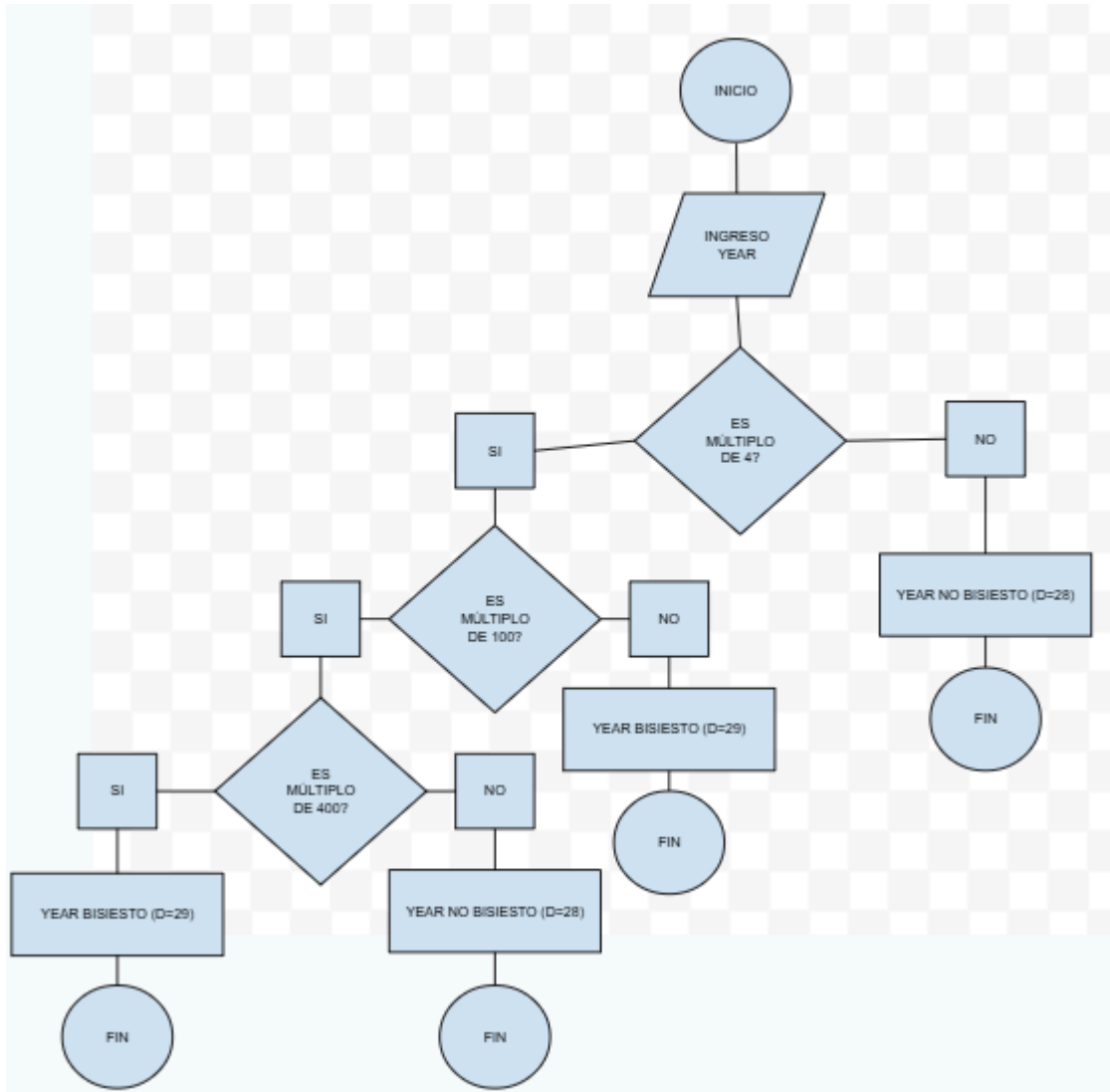
La regla para determinar si un año es bisiesto es la siguiente:

- El año debe ser divisible por 4.
- Si el año es divisible por 100, entonces debe ser divisible por 400 para ser bisiesto.

En resumen:

- Un año es bisiesto si es divisible por 4.
- Sin embargo, si también es divisible por 100, debe serlo además por 400.

B



C

-- Crear la función

```
CREATE FUNCTION dbo.b(@year INT)
RETURNS VARCHAR(12)
```

```
AS
```

```
BEGIN
```

```
    IF @year % 4 = 0
```

```
    BEGIN
```

```
        IF @year % 100 = 0
```

```
        BEGIN
```

```
            IF @year % 400 = 0
```

```
            BEGIN
```

```
                RETURN 'AÑO BISIESTO';
```

```
            END
```

```
        ELSE
```

```
        BEGIN
```

```
            RETURN 'AÑO NO BISIESTO';
```

```
        END;
```

```
    END
```

```
    ELSE
```

```
    BEGIN
```

```
        RETURN 'AÑO BISIESTO';
```

```
    END;
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
    RETURN 'AÑO NO BISIESTO';
```

```
END;
```

```
    RETURN 'Error'; -- Para asegurar que siempre haya un RETURN al final
END;
```

-- Probar la función

```
DECLARE @year INT;
```

```
SET @year = 2000; -- Puedes cambiar el año aquí para probar con diferentes valores
```

```
SELECT @year AS Anio, dbo.b(@year) AS Resultado;
```

Ejercicio Práctico 3.17:

Escriba el código para la creación de la función desarrollada en el ejercicio 3.16

Idem punto 3.16 - C

Ejercicio Práctico 3.18:

Escriba el código para mostrar todos los números pares menores que 23 (Sugerencia: considere arrancar en un número par y sumar de a 2)

-- CREAR LA FUNCION

DROP FUNCTION dbo.Pares;

CREATE FUNCTION dbo.Pares(@numeroFinal int)

RETURNS @Pares TABLE (Numero INT)

AS

BEGIN

DECLARE @numero INT = 0;

WHILE @numero < @numeroFinal

BEGIN

INSERT INTO @Pares (Numero) VALUES (@numero);

SET @numero = @numero + 2;

END

RETURN;

END;

-- PROBAR LA FUNCION

DECLARE @numero int;

SET @numero=23;

SELECT * FROM dbo.Pares(@numero);

Ejercicio Práctico 3.19:

Escriba el código para mostrar todos los números menores que 100 indicando si son pares o impares.

(Sugerencia: considere incluir dentro del ciclo a un IF que pregunte si $2 * \text{FLOOR}(@N/2)$ es igual a $@N$)

-- CREAR LA FUNCION

CREATE FUNCTION dbo.ParesMenoresQue100()

RETURNS @tabla TABLE (numero int, ParoImpar varchar(5))

AS

BEGIN

DECLARE @NUM INT SET @NUM=0;

WHILE @NUM<100

BEGIN

IF @NUM%2=0

BEGIN

INSERT INTO @tabla (numero,ParoImpar) VALUES (@NUM,
'PAR');

END

ELSE

BEGIN

INSERT INTO @tabla (numero,ParoImpar) VALUES (@NUM,
'IMPAR');

END

SET @NUM=@NUM+1;

END

RETURN;

END;

-- PROBAR LA FUNCION

SELECT * FROM dbo.ParesMenoresQue100();

Ejercicio Práctico 3.20:

Asuma que tiene en una tabla SUELDOS con campos NOMBRE y SUELDO los datos correspondientes al personal de su empresa. Necesita escribir las consultas para determinar:

- a. El mayor sueldo
- b. El menor sueldo
- c. El sueldo promedio
- d. La cantidad de empleados

```
CREATE TABLE sueldos(id bigint identity(1,1) primary key, nombre varchar(50), sueldo decimal(10,2));
```

-- A

```
SELECT MAX(sueldo) FROM sueldos;  
SELECT nombre,sueldo FROM sueldos WHERE sueldo=(SELECT MAX(sueldo) FROM sueldos);
```

-- B

```
SELECT MIN(sueldo) FROM sueldos;  
SELECT nombre,sueldo FROM sueldos WHERE sueldo=(SELECT MIN(sueldo) FROM sueldos);
```

-- C

```
SELECT AVG(sueldo) FROM sueldos;  
SELECT nombre,sueldo FROM sueldos WHERE sueldo=(SELECT AVG(sueldo) FROM sueldos);
```

-- D

```
SELECT COUNT(*) FROM sueldos;
```

Ejercicio Práctico 3.21:

Asuma que tiene en una tabla SUELDOS con campos NOMBRE y SUELDO los datos correspondientes al personal de su empresa. También tiene una tabla de liquidaciones con NOMBRE, MES y LIQUIDACIÓN donde se guarda lo que se pagó a cada persona.

a. Escriba un trigger que borre las liquidaciones al borrar un empleado en la tabla SUELDOS.

b. Escriba un trigger que inserte al empleado en la tabla SUELDOS al cargar una liquidación

c. Escriba un trigger que modifique el nombre en la tabla liquidaciones si se modifica el nombre en la tabla SUELDOS. (Muy desafiante)

A

-- CREACION DEL TRIGGER

CREATE TRIGGER borrarEmpleado

ON sueldos

AFTER delete

AS

BEGIN

DELETE FROM liquidaciones WHERE id_empleado in (SELECT id FROM DELETED);

END;

-- CONSULTA DE REGISTROS

SELECT * FROM sueldos;

SELECT * FROM liquidaciones;

-- PRUEBA DEL TRIGGER

DELETE FROM sueldos WHERE id=1;

146 %

Results Messages

	id	nombre	suelo
1	1	juan	57082.00
2	2	maría	39564.00
3	3	carlos	51044.00
4	4	ana	31939.00
5	5	luis	29228.00
6	6	elena	58128.00
7	7	miguel	68055.00
8	8	sofia	60771.00

	id_empleado	nombre	mes	liquidacion
1	1	juan	enero	56100.00
2	2	maría	enero	35523.00
3	3	carlos	enero	66944.00
4	4	ana	enero	69186.00
5	5	luis	enero	57300.00

Se borra el empleado “Juan” de la tabla “sueldos”.

`DELETE FROM sueldos WHERE id=1;`

146 %

Messages

(1 row affected)

(1 row affected)

146 %

Results Messages

	id	nombre	sueldo
1	2	maría	39564.00
2	3	carlos	51044.00
3	4	ana	31939.00
4	5	luis	29228.00
5	6	elena	58128.00
6	7	miguel	68055.00
7	8	sofia	60771.00
8	9	pedro	69642.00

	id_empleado	nombre	mes	liquidacion
1	2	maría	enero	20798.00
2	3	carlos	enero	30054.00
3	4	ana	enero	25904.00
4	5	luis	enero	31499.00

Se habían declarado las tablas con las restricciones de clave foránea entre:

- id(tabla sueldos)
- id_empleado (tabla liquidaciones),

pero no fue posible utilizar el TRIGGER por el siguiente error:

- (The DELETE statement conflicted with the REFERENCE constraint "FK__liquidaci__id_em__4BAC3F29". The conflict occurred in database "m1_u3", table "dbo.liquidaciones", column 'id_empleado'.)

B

-- CREACION DEL TRIGGER

```
CREATE TRIGGER insertarEmpleado
```

```
ON liquidaciones
```

```
AFTER INSERT
```

```
AS
```

```
BEGIN
```

```
INSERT INTO sueldos (nombre,sueldo) SELECT nombre,liquidacion from inserted;
```

```
END;
```

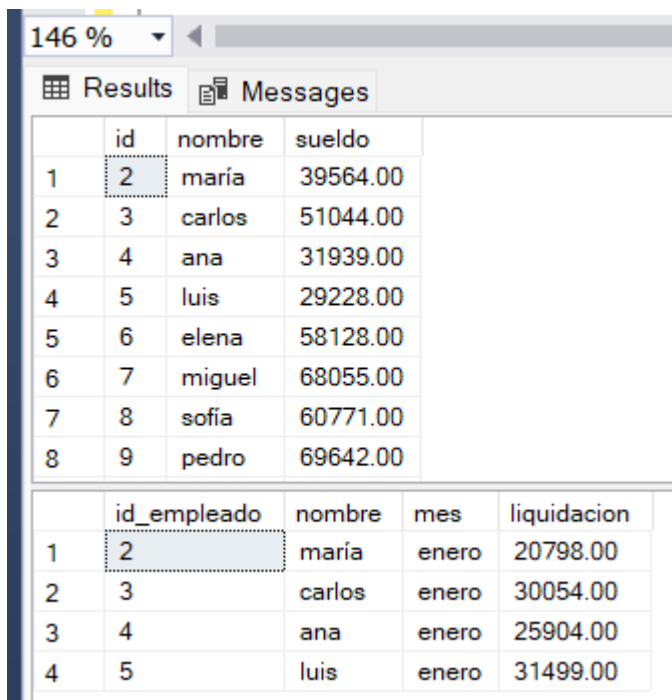
-- CONSULTA DE REGISTROS

```
SELECT * FROM sueldos;
```

```
SELECT * FROM liquidaciones;
```

-- PRUEBA DEL TRIGGER

```
INSERT INTO liquidaciones (nombre, mes,liquidacion) VALUES ('juan','febrero',35.25);
```



	id	nombre	sueldo
1	2	maría	39564.00
2	3	carlos	51044.00
3	4	ana	31939.00
4	5	luis	29228.00
5	6	elena	58128.00
6	7	miguel	68055.00
7	8	sofia	60771.00
8	9	pedro	69642.00

	id_employado	nombre	mes	liquidacion
1	2	maría	enero	20798.00
2	3	carlos	enero	30054.00
3	4	ana	enero	25904.00
4	5	luis	enero	31499.00

Se agrega un registro en la tabla “liquidaciones”

```
INSERT INTO liquidaciones (nombre, mes,liquidacion) VALUES ('juan','febrero',35.25);
```

177 %

Messages

(1 row affected)

(1 row affected)

177 %

Results Messages

15	16	natalia	44724.00
16	17	diego	60459.00
17	18	isabel	28358.00
18	19	roberto	34898.00
19	20	patricia	41099.00
20	21	juan	35.25

	id_empleado	nombre	mes	liquidacion
1	2	maría	enero	20798.00
2	3	carlos	enero	30054.00
3	4	ana	enero	25904.00
4	5	luis	enero	31499.00
5	6	juan	febr...	35.25

C

-- CREACION DEL TRIGGER

CREATE TRIGGER modificarNombre

ON sueldos

AFTER UPDATE

AS

BEGIN

UPDATE liquidaciones SET nombre=(SELECT nombre FROM inserted) WHERE nombre
=(SELECT nombre FROM deleted)

END;

-- CONSULTA DE REGISTROS

SELECT * FROM sueldos;

SELECT * FROM liquidaciones;

-- PRUEBA DEL TRIGGER

UPDATE sueldos SET nombre='ana maria de los angeles' WHERE id=2;

177 %

Results		Messages	
	id	nombre	sueldo
1	2	ana	39564.00
2	3	carlos	51044.00
3	4	ana	31939.00
4	5	luis	29228.00
5	6	elena	58128.00
6	7	miguel	68055.00
7	8	sofia	60771.00
8	9	pedro	69642.00
9	10	laura	55409.00

	id_empleado	nombre	mes	liquidacion
1	1	juan	enero	40889.00
2	2	maría	enero	32021.00
3	3	carlos	enero	36618.00
4	4	ana	enero	36035.00
5	5	luis	enero	56266.00

Se modifica el nombre de maria → ana maria de los angeles en la tabla “sueldos”

`UPDATE sueldos SET nombre='ana maria de los angeles' WHERE id=2;`

177 %

Messages	
(1 row affected)	
(1 row affected)	

177 %

Results		Messages	
	id	nombre	sueldo
1	2	ana maria de los angeles	39564.00
2	3	carlos	51044.00
3	4	ana	31939.00
4	5	luis	29228.00
5	6	elena	58128.00
6	7	miguel	68055.00
7	8	sofia	60771.00
8	9	pedro	69642.00
9	10	laura	55409.00

	id_empleado	nombre	mes	liquidacion
1	1	juan	enero	40889.00
2	2	maría	enero	32021.00
3	3	carlos	enero	36618.00
4	4	ana maria de los angeles	enero	36035.00
5	5	luis	enero	56266.00

Ejercicio Práctico 3.22:

Asuma que tiene en una tabla PERSONAL con campos NOMBRE y FECHAINGRESO los datos correspondientes al personal de su empresa.

Alguien le dijo que necesita agregar un identificador único correlativo que será el número de legajo. Usted ya ha creado el campo LEGAJO pero está vacío y quiere llenarlo

a. Escriba un cursor que recorra la tabla PERSONAL y vaya cargando los legajos en forma correlativa de acuerdo al orden de ingreso a la empresa (Al más antiguo el 1, al siguiente el 2, y así siguiendo adelante)

b. Usted ya hizo el punto a pero ahora ha comprado a un competidor y tiene que consolidar las tablas de personal. Le insertará todos los nombres y luego necesitará un cursor que recorra solo los registros que tienen LEGAJO en NULL y les inserte números de legajo a partir del último número pre-existente.

A

```
CREATE TABLE personal(  
id int identity(1,1) primary key,  
nombre varchar(50),  
fechaIngreso date);
```

```
INSERT INTO personal (nombre, fechaIngreso) VALUES ('juan', '02-10-2017');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('ana', '05-12-2019');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('sofia', '10-01-2024');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('pedro', '10-12-1999');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('nicolas', '11-04-2020');
```

```
select * from personal;
```

```
ALTER TABLE personal ADD legajo int;
```

```
DECLARE c CURSOR FOR SELECT id FROM personal ORDER BY fechaIngreso ASC;  
OPEN c  
DECLARE @variable INT  
FETCH NEXT FROM c INTO @variable  
DECLARE @CONTADOR INT
```

```

SET @CONTADOR=1
WHILE @@FETCH_STATUS=0
BEGIN
UPDATE personal SET legajo=@CONTADOR WHERE id=@variable
SET @CONTADOR=@CONTADOR+1
FETCH NEXT FROM c INTO @variable
END
CLOSE c
DEALLOCATE c

```

Se genera la tabla personal y se insertan los registros aleatorios

Results		Messages	
	id	nombre	fechaIngreso
1	1	juan	2017-02-10
2	2	ana	2019-05-12
3	3	sofia	2024-10-01
4	4	pedro	1999-10-12
5	5	nicolas	2020-11-04

Se agrega la columna legajo

Results

Messages

	id	nombre	fechaIngreso	legajo
1	1	juan	2017-02-10	NULL
2	2	ana	2019-05-12	NULL
3	3	sofia	2024-10-01	NULL
4	4	pedro	1999-10-12	NULL
5	5	nicolas	2020-11-04	NULL

Se inserta el número de legajo de acuerdo a la fecha de ingreso

Results

Messages

	id	nombre	fechaIngreso	legajo
1	1	juan	2017-02-10	2
2	2	ana	2019-05-12	3
3	3	sofia	2024-10-01	5
4	4	pedro	1999-10-12	1
5	5	nicolas	2020-11-04	4

B

```
INSERT INTO personal (nombre, fechaIngreso) VALUES ('ariel', '10-12-1986');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('eliana', '05-12-2021');  
INSERT INTO personal (nombre, fechaIngreso) VALUES ('maria', '10-01-1995');
```

```
DECLARE c CURSOR FOR SELECT id FROM personal WHERE legajo IS NULL ORDER  
BY fechaIngreso ASC;
```

```
OPEN c
```

```
DECLARE @variable2 INT
```

```
FETCH NEXT FROM c INTO @variable2
```

```
DECLARE @CONTADOR2 INT
```

```
SET @CONTADOR2=(select max(legajo)from personal)+1
```

```
WHILE @@FETCH_STATUS=0
```

```
BEGIN
```

```
UPDATE personal SET legajo=@CONTADOR2 WHERE id=@variable2
```

```
SET @CONTADOR2=@CONTADOR2+1
```

```
FETCH NEXT FROM c INTO @variable2
```

```
END
```

```
CLOSE c
```

```
DEALLOCATE c
```