

SQL - DML
Data Manipulation Language
Insert-Update-Delete-Merge

SQL DDL
Secuencias

UTN - FRBA
Ing. en Sistemas de Información
Gestión de Datos

Prof.: Ing. Juan Zaffaroni

SQL – Hoy Veremos

- DDL – Data Definition Language
 - SECUENCIAS
- DML – Data Manipulation Language
 - INSERT
 - UPDATE
 - DELETE
 - MERGE

Secuencias

Los generadores de secuencias proveen una serie de números secuenciales, especialmente usados en entornos multiusuarios para generar una números secuenciales y únicos sin el overhead de I/O a disco o el lockeo transaccional.

Los motores de base de datos proveen diferentes formas de implementar secuencias a través de:

- Tipo de Dato de una columna (Informix, PostgreSQL)
- Propiedades de una columna (SqlServer, Mysql, DB2)
- Objeto Sequence (Oracle, Informix, PostgreSQL, DB2, SqlServer)

Secuencias

- Tipo de Dato de una columna
 - Motor BD Informix SERIAL.
- Propiedades de una columna
 - Motor SqlServer IDENTITY
 - Motor Mysql AUTO_INCREMENT
 - Motor DB2 IDENTITY
- Objeto Sequence
 - Motores Oracle, Informix, PostgreSQL, DB2, SqlServer.
 - CREATE SEQUENCE

Secuencias

Tipo de Dato de una columna

El motor Informix posee varios tipo de datos SERIAL, SERIAL8 y BIGSERIAL que permiten realizar lo mismo que un objeto secuencia. Al insertar una fila en dicha tabla y asignarle un valor cero, el motor va a buscar el próximo nro. del más alto existente en la tabla.

```
CREATE TABLE ordenes (
    N_orden SERIAL,
    N_cliente INTEGER,
    F_orden DATE,
    I_Total DECIMAL(15 , 2),
    C_estado SMALLINT,
    F_alta_audit DATETIME YEAR TO SECOND,
    D_usuario VARCHAR(20)
);
```

```
INSERT INTO ordenes
VALUES (0, 17, '15/05/2006', 1, CURRENT, "user2")
```

Secuencias

Propiedades de una columna

Existen motores que poseen propiedades de columna que permite realizar lo mismo que una secuencia. Al insertar una fila en dicha tabla, el motor va a buscar el próximo nro. del más alto existente en la tabla.

Ej. SQLServer IDENTITY

```
CREATE TABLE ordenes (
    N_orden int IDENTITY (1, 1),
    N_cliente int NULL ,
    F_orden datetime NULL);
```

```
INSERT INTO ordenes
(n_cliente, f_orden)
```

```
VALUES (114,'2020-03-03')
```

Ej. MySQL AUTO_INCREMENT

```
CREATE TABLE animales
( animal_id INT AUTO_INCREMENT,
  nombre CHAR(30) NOT NULL);

INSERT INTO animales ('perro'), ('gato');
```

Secuencias

Objeto SEQUENCE

Una secuencia debe tener un nombre, debe ser ascendente o descendente, debe tener definido el intervalo entre números, tiene definidos métodos para obtener el próximo número ó el actual (entre otros).

Ej. SEQUENCE ORACLE

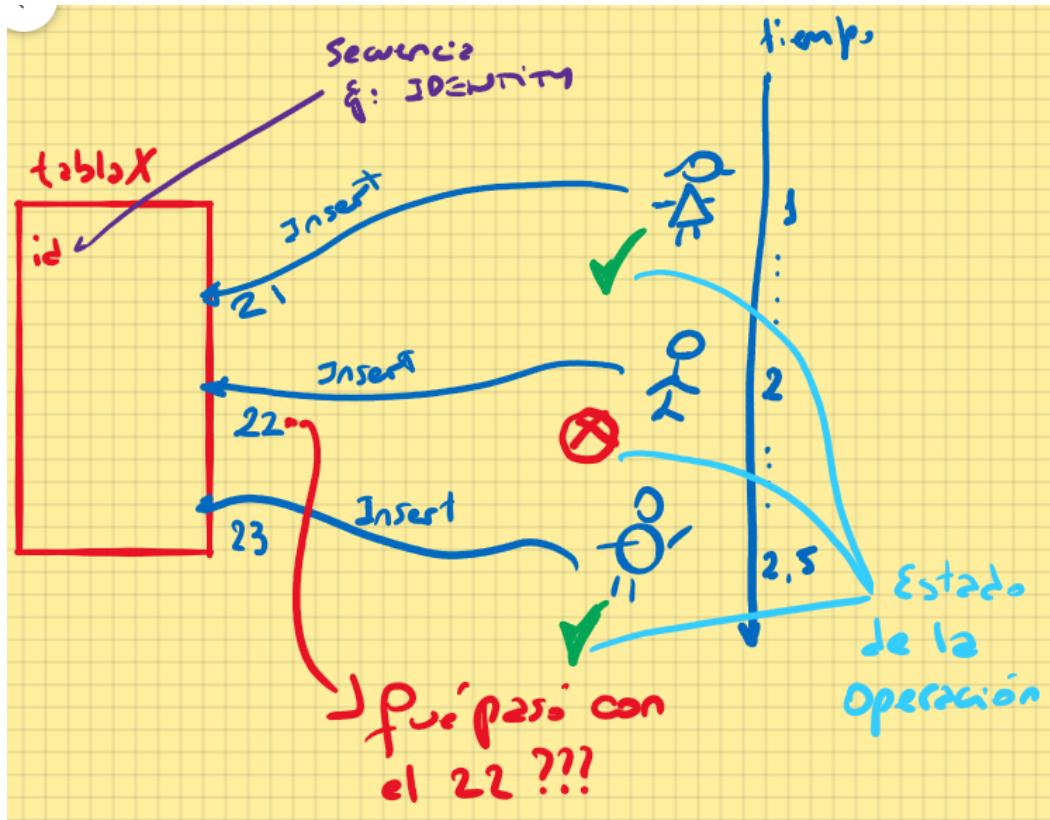
```
CREATE SEQUENCE sqc_orden_nro
INCREMENT BY 1
START WITH 10
MAXVALUE 9999
NOCYCLE
NOCACHE;
```

```
INSERT INTO ordenes
VALUES (sqc_orden_nro.NEXTVAL, 17,
to_date('15/05/2006', 'dd/mm/yyyy'),
1, SYSDATE, USER);
```

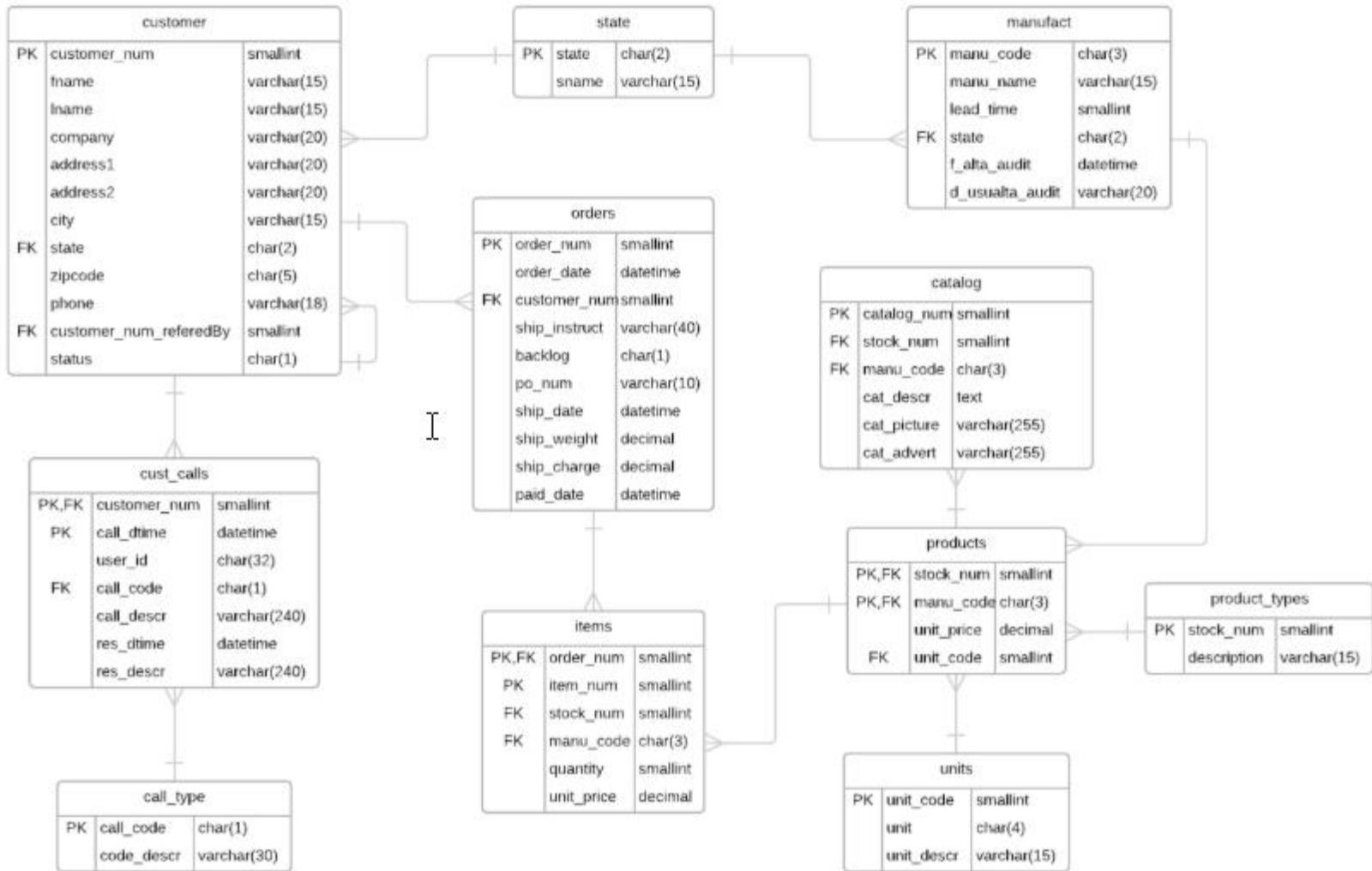
```
INSERT INTO items_ordenes
VALUES (sqc_orden_nro.CURRVAL, 1,
176, 20, 10.57);
```

Secuencias

Cuidado con las secuencias, cuando requerimos números consecutivos sin huecos.



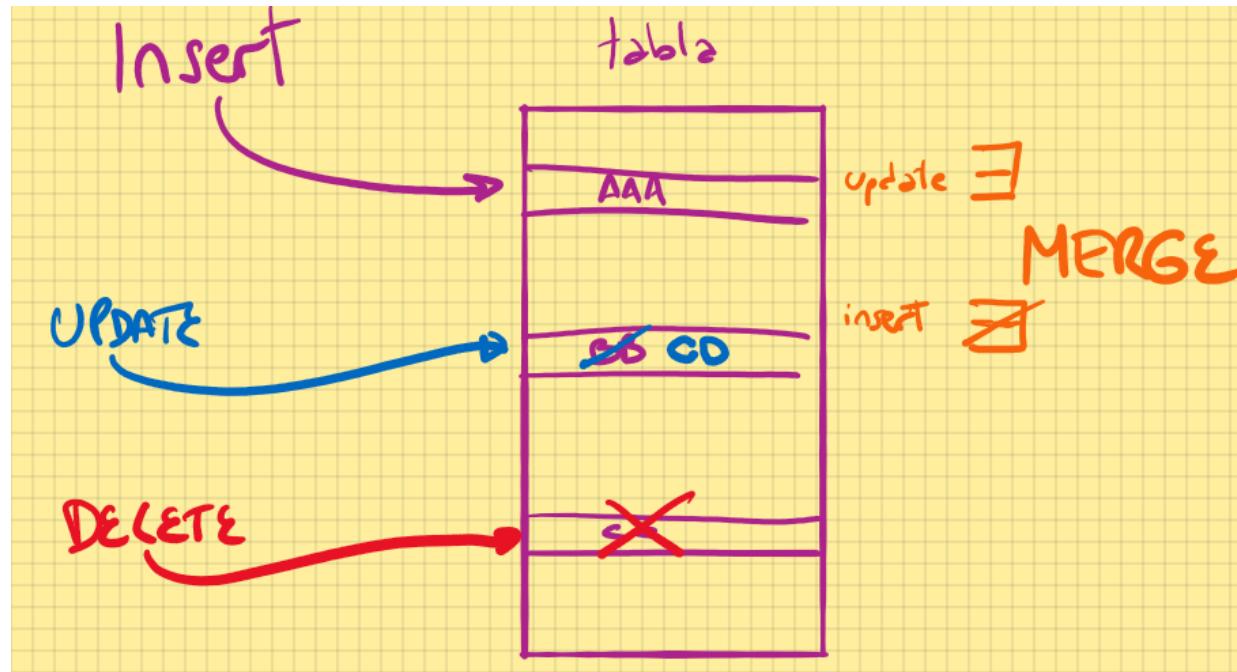
Base de Datos de Ejemplo



SQL – DML

- DML – Data Manipulation Language

- INSERT
- UPDATE
- DELETE
- MERGE



SQL – Operador INSERT

Se inserta una sola fila en la tabla

INSERT INTO nom_tabla [(lista de columnas)] (*) ()**
VALUES (lista de valores) ()**

Se insertan multiples filas en la tabla

INSERT INTO nom_tabla [(lista de columnas)]
SELECT (*)**

(*) La lista de columnas entre corchetes significa que es opcional.

(**) la lista de columnas y valores tienen como separador una coma.

(***) El select debe devolver una lista columnas similar a la que espera recibir el INSERT.

SQL – Operador INSERT

**INSERT INTO product_types
VALUES (375,'Short Baño')**

Inserta en la tabla tipo de productos un nuevo tipo de producto.

The screenshot shows two separate SQL query windows in the SQL Server Management Studio interface.

Top Query Window:

```
SQLQuery1.sql - D...HQEMP6\zaffa (53)* ↗ X
INSERT INTO product_types VALUES (375, 'Short Baño')

100 % <
Messages
(1 row(s) affected)
```

Bottom Query Window:

```
SQLQuery1.sql - D...HQEMP6\zaffa (53)* ↗ X
SELECT * FROM product_types WHERE stock_num=375

100 % <
Results Messages
stock_num description
1 375 Short Baño
```

The top window shows the execution of an **INSERT INTO** statement, which inserted a new row into the **product_types** table with **stock_num = 375** and **description = 'Short Baño'**. The message pane indicates **(1 row(s) affected)**.

The bottom window shows the execution of a **SELECT** statement, which retrieves all columns for rows where **stock_num = 375**. The results pane displays a single row with **stock_num = 375** and **description = 'Short Baño'**.

SQL – Operador INSERT

```
INSERT INTO product_types  
VALUES ('Short Baño', 375)
```

El problema de no poner la lista de columnas a insertar es que mi lista de valores debe respetar 100 el orden de las columnas en la tabla, acoplando mi consulta al modelo.

The screenshot shows a SQL query window titled "SQLQuery1.sql - D...HQEMP6\zaffa (53)*". The query is:

```
INSERT INTO product_types VALUES ('Short Baño', 375)
```

In the "Messages" pane, there is an error message:

```
Msg 245, Level 16, State 1, Line 2  
Conversion failed when converting the varchar value 'Short Baño' to data type smallint.
```

En este ejemplo, vemos que al querer insertar los valores fuera de orden, el comando falla debido a que espera un valor smallint en la primer posición.

Si la tabla hubiese tenido dos campos varchar, no nos habríamos dado cuenta del error, ya que el Motor de BD habría aceptado los datos erróneos.

SQL – Operador INSERT

INSERT INTO customer **VALUES**

```
(266,'Godoy','Estela','Pintos 3325',
'Ituza SA',null,'Buenos Aires',null,null,
null,null,null)
```

El problema de no poner la lista de columnas a insertar es que mi lista de valores debe respetar el orden de las columnas en la tabla, acoplando mi consulta al modelo.

The screenshot shows a SQL query window with the following content:

```
1
INSERT INTO customer VALUES
(266, 'Godoy', 'Estela', 'Pintos 3325', 'Ituza SA', null, 'Buenos Aires', null, null, null, null, null)
```

Below the query, the results are displayed:

1 row(s) affected

dbo.customer

customer_num	fname	Iname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status
266	Godoy	Estela	Pintos 3325	Ituza SA		Buenos Aires					

The table structure for the customer table is also shown:

- customer_num (PK, smallint, not null)
- fname (varchar(15), null)
- Iname (varchar(15), null)
- company (varchar(20), null)
- address1 (varchar(20), null)
- address2 (varchar(20), null)
- city (varchar(15), null)
- state (FK, char(2), null)
- zipcode (char(5), null)
- phone (varchar(18), null)
- customer_num_referredBy (FK, smallint, n
- status (char(1), null)

En este caso vemos que el motor de BD acepto los datos erróneos, porque son del mismo tipo.

El apellido y nombre estan invertidos y la direccion1 y la compañía también.

SQL – Operador INSERT

INSERT INTO customer **VALUES**

(266,'Godoy','Estela','Pintos 3325',
'Ituza SA',null,'Buenos Aires')

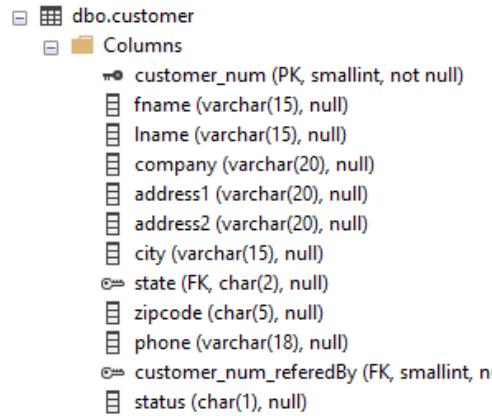
El problema de no poner la lista de columnas a insertar es que mi lista de valores debe respetar el orden de las columnas en la tabla, acoplando mi consulta al modelo.

The screenshot shows a SQL query window with the following code:

```
INSERT INTO customer VALUES
(267, 'Godoy', 'Javier', 'Pintos 3325', 'Ituza SA', null, 'Buenos Aires')
```

The status bar indicates "100 %". Below the query window is a "Messages" pane showing the following error message:

Msg 213, Level 16, State 1, Line 2
Column name or number of supplied values does not match table definition.



En este caso vemos que el motor de BD no acepto los datos erróneos, debido a que la cantidad de valores difiere con la cantidad de columnas de la tabla.

SQL – Operador INSERT

```
INSERT INTO product_types  
(stock_num, description)  
VALUES (376,'Short Rugby')
```

```
INSERT INTO product_types  
(stock_num, description)  
VALUES (376, 'Short Rugby')
```

100 %

Messages

(1 row(s) affected)

Inserta en la tabla tipo de productos un nuevo tipo de producto.

Agregando la lista de atributos podemos desacoplar la instrucción de la definición de la tabla.

```
SELECT * FROM product_types  
WHERE stock_num=376
```

	stock_num	description
1	376	Short Rugby

SQL – Operador INSERT

INSERT INTO customer

(stock_num, lname, fname, address1,
Company, address2, city)

VALUES

(266,'Godoy','Estela','Pintos 3325',
'Ituza SA',null,'Buenos Aires')

```
100 % < > 
[ ] Messages
(1 row(s) affected)

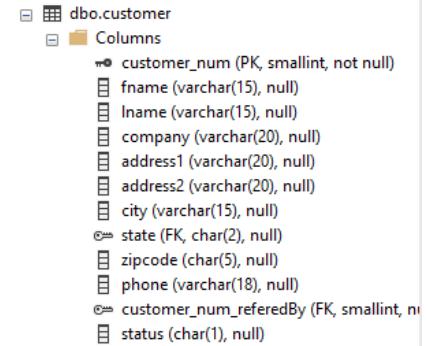
INSERT INTO customer (customer_num, lname, fname, address1, company, address2, city)
VALUES (266, 'Godoy', 'Estela', 'Pintos 3325', 'Ituza SA', null, 'Buenos Aires')
```

Se observa que se pudo insertar la fila en la tabla.

Obviamente que las columnas no incluidas en el INSERT deben aceptar valores nulos o tener un DEFAULT definido.

Al agregar la lista de columnas, podemos insertar filas con una cantidad menor de valores a los que tiene la definición de la tabla y sin necesitar respetar el orden de los mismos en la tabla real.

Siempre y cuando la lista de columnas y la lista de valores coincidan



SQL – Operador INSERT

INSERT INTO closed_orders

SELECT * FROM orders

WHERE paid_date IS NOT null

En una tabla creada previamente con la misma estructura que la tabla ordenes, insertamos las filas que devuelve el SELECT.

Es fundamental que el select devuelva las mismas filas y en el mismo orden las columnas que la tabla destino.

The screenshot shows a SQL Server Management Studio window with two tabs: 'Object Explorer Details' and 'SQL Query1.sql - D...HQEMP6\zaffa (54)*'. The code in the query editor is as follows:

```
CREATE TABLE closed_orders(
    [order_num] [smallint] NOT NULL PRIMARY KEY,
    [order_date] [datetime] NULL,
    [customer_num] [smallint] NOT NULL,
    [ship_instruct] [varchar](40) NULL,
    [backlog] [char](1) NULL,
    [po_num] [varchar](10) NULL,
    [ship_date] [datetime] NULL,
    [ship_weight] [decimal](8, 2) NULL,
    [ship_charge] [decimal](6, 2) NULL,
    [paid_date] [datetime] NULL,
);
INSERT INTO closed_orders SELECT * FROM orders WHERE paid_date IS NOT null
```

The 'Messages' pane at the bottom shows the result: '(16 row(s) affected)'.

En este ejemplo el insert esta acoplado a la definición de la tabla closed_orders y el SELECT está acoplado a la definición de la tabla orders.

SQL – Operador INSERT

INSERT INTO closed_orders

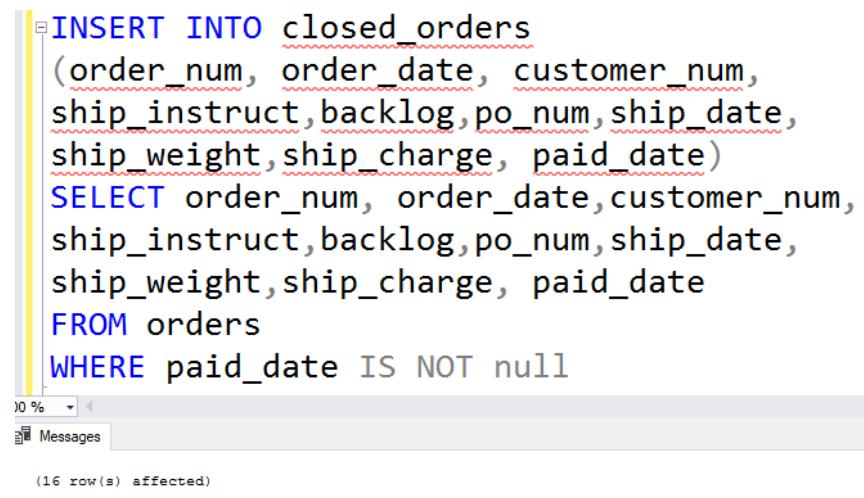
(order_num, order_date, customer_num,
ship_instruct, backlog, po_num, ship_date,
ship_weight, ship_charge, paid_date)

SELECT order_num, order_date, customer_num,
ship_instruct, backlog, po_num, ship_date,
ship_weight, ship_charge, paid_date
FROM orders

WHERE paid_date **IS NOT** null

En una tabla creada previamente con la misma estructura que la tabla ordenes, insertamos las filas que devuelve el SELECT.

Con la lista de columnas en el INSERT y en el SELECT los desacoplamos de la definición de las tablas y viendo las instrucciones sabemos claramente que se está insertando y en donde.



```
INSERT INTO closed_orders
(order_num, order_date, customer_num,
ship_instruct, backlog, po_num, ship_date,
ship_weight, ship_charge, paid_date)
SELECT order_num, order_date, customer_num,
ship_instruct, backlog, po_num, ship_date,
ship_weight, ship_charge, paid_date
FROM orders
WHERE paid_date IS NOT null
```

SQL – Operador INSERT

INSERT INTO manufact

(manu_code, manu_name, lead_time,state)

VALUES ('DBL','DBLANDIT',1,'CA')

```
INSERT INTO manufact
(manu_code, manu_name, lead_time, state)
VALUES ('DBL', 'DBLANDIT', 1, 'CA')
```

100 %

Messages

(1 row(s) affected)

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)*

```
SELECT * FROM manufact WHERE manu_code='DBL'
```

100 %

Results Messages

	manu_code	manu_name	lead_time	state	f_alta_audit	d_usualta_audit
1	DBL	DBLANDIT	1	CA	2020-04-14 23:15:37.330	dbo

DEFAULT CONSTRAINT

Cuando insertamos valores en una tabla y obviamos determinadas columnas, por DEFAULT tendrán valor NULL, salvo que con el constraint de DEFAULT les asignemos otro valor.

```
CREATE TABLE manufact
(
    manu_code char(3) NOT NULL PRIMARY KEY CLUSTERED,
    manu_name varchar(15) NULL,
    lead_time smallint NULL,
    state char(2) NULL,
    f_alta_audit datetime NULL DEFAULT (user_name()),
    d_usualta_audit varchar(20) NULL DEFAULT (getdate())
)
```

Se observa que en los atributos f_alta_audit y d_usualta_audit tienen los datos definidos como DEFAULT en la definición de la tabla.

SQL – Operador INSERT

INSERT INTO empleados (nombre, apellido, cuit)

VALUES ('Lisandro','Ayestaran',20235582487)

IDENTITY

Al insertar una fila en una tabla con un atributo con la **propiedad de IDENTITY**, **NO** se debe incluir dicha columna en la lista de columnas, ni poner valor en la lista de valores.

```
CREATE TABLE empleados
(empleadoId INT IDENTITY (1,1) PRIMARY KEY,
nombre VARCHAR(60),
apellido VARCHAR(60),
CUIT BIGINT);

INSERT INTO empleados (nombre, apellido, cuit)
VALUES ('Lisandro','Ayestaran',20235582487)
```

100 %

Messages

(1 row(s) affected)

100 %

Results Messages

SELECT * FROM empleados

	empleadoId	nombre	apellido	CUIT
1	1	Lisandro	Ayestaran	20235582487

En el ejemplo vemos que la columna empleadoId no es incluida en la lista de columnas y en la lista de valores.

SQL – Operador UPDATE

Se modifica una o varias columnas de las filas que cumplan con la condición.

UPDATE nom_tabla

SET columna=valor[, columna=valor...])

[WHERE condiciones] (*)

(*) La cláusula **WHERE** es opcional, pero **CUIDADO si no se pone condición el UPDATE se realizará sobre la TOTALIDAD DE LAS FILAS DE LA TABLA.**

SQL – Operador UPDATE

UPDATE customer

SET company = 'UTN', phone = '5555-5555'

WHERE customer_num = 112

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)* ✎ X

```
UPDATE customer
SET company = 'UTN', phone = '5555-5555'
WHERE customer_num = 112
```

100 %

Messages

(1 row(s) affected)

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)* ✎ X

```
SELECT * FROM customer
WHERE customer_num = 112
```

100 %

Results Messages

	customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status
1	112	Margaret	Lawson	UTN	234 Wyandotte Way		Los Altos	CA	94022	5555-5555	111	NULL

SQL – Operador UPDATE

UPDATE customer

SET company = 'UTN', phone = '5555-5555'

WHERE customer_num = 112

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)* ✎ X

```
UPDATE customer
SET company = 'UTN', phone = '5555-5555'
WHERE customer_num = 112
```

100 %

Messages

(1 row(s) affected)

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)* ✎ X

```
SELECT * FROM customer
WHERE customer_num = 112
```

100 %

Results Messages

	customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status
1	112	Margaret	Lawson	UTN	234 Wyandotte Way		Los Altos	CA	94022	5555-5555	111	NULL

SQL – Operador UPDATE

WARNING

UPDATE empleados

SET apellido='Salerno'

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)*

```
SELECT * FROM empleados
```

Results Messages

	empleadoid	nombre	apellido	CUIT
1	1	Lisandro	Ayestaran	20235582487
2	2	Heman	Puente	6
3	3	Facundo	Anrieta	5
4	4	Mariela	Diaz	4
5	5	Jorge	Imach	3
6	6	Federico	Hernida	2
7	7	Isabel	Sanchez	1

Este UPDATE no tiene la cláusula WHERE, por lo que se modificarán todas las filas de la tabla asignándole al apellido 'Salerno'.

Object Explorer Details SQLQuery1.sql - D...HQEMP6\zaffa (54)*

```
UPDATE empleados  
SET apellido='Salerno';
```

```
SELECT * FROM empleados;
```

Results Messages

	empleadoid	nombre	apellido	CUIT
1	1	Lisandro	Salerno	20235582487
2	2	Heman	Salerno	6
3	3	Facundo	Salerno	5
4	4	Mariela	Salerno	4
5	5	Jorge	Salerno	3
6	6	Federico	Salerno	2
7	7	Isabel	Salerno	1

SQL – Operador UPDATE

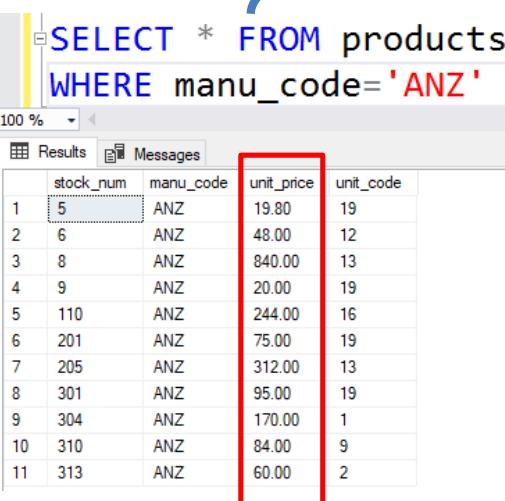
UPDATE products

SET unit_price= unit_price*1,20

WHERE manu_code='ANZ'

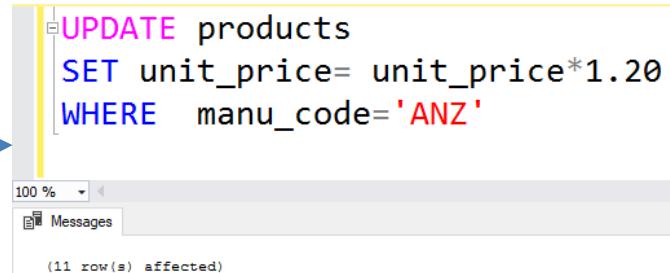
Modificar ciertas filas
cambiando el valor de
una columna por el
resultado de una
formula.

SELECT * FROM products
WHERE manu_code= 'ANZ'

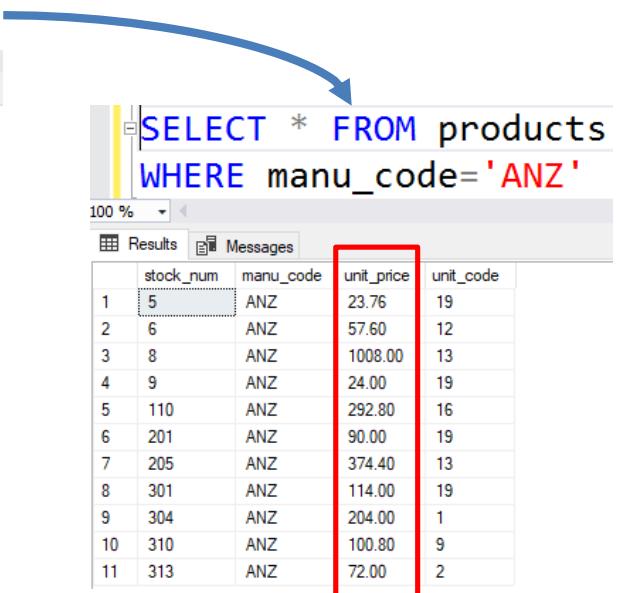


	stock_num	manu_code	unit_price	unit_code
1	5	ANZ	19.80	19
2	6	ANZ	48.00	12
3	8	ANZ	840.00	13
4	9	ANZ	20.00	19
5	110	ANZ	244.00	16
6	201	ANZ	75.00	19
7	205	ANZ	312.00	13
8	301	ANZ	95.00	19
9	304	ANZ	170.00	1
10	310	ANZ	84.00	9
11	313	ANZ	60.00	2

```
UPDATE products
SET unit_price= unit_price*1.20
WHERE manu_code='ANZ'
```



SELECT * FROM products
WHERE manu_code= 'ANZ'



	stock_num	manu_code	unit_price	unit_code
1	5	ANZ	23.76	19
2	6	ANZ	57.60	12
3	8	ANZ	1008.00	13
4	9	ANZ	24.00	19
5	110	ANZ	292.80	16
6	201	ANZ	90.00	19
7	205	ANZ	374.40	13
8	301	ANZ	114.00	19
9	304	ANZ	204.00	1
10	310	ANZ	100.80	9
11	313	ANZ	72.00	2

SQL – Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE.

DELETE FROM nom_tabla

[WHERE condiciones] (*)

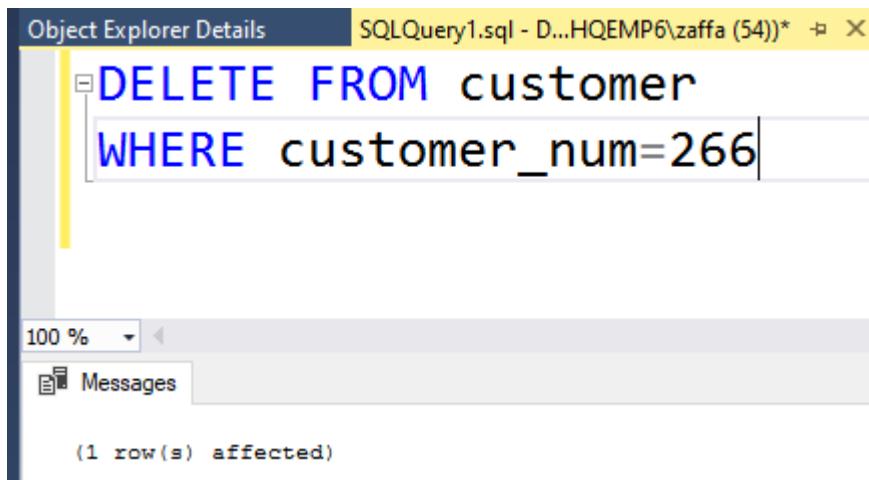
(*) La cláusula **WHERE** es opcional, pero **CUIDADO si no se pone condición el DELETE borrará la TOTALIDAD DE LAS FILAS DE LA TABLA.**

SQL – Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE, o sea cuyo `customer_num` sea 266.

DELETE FROM customer

WHERE customer_num=266



The screenshot shows a SQL Server Management Studio interface. The title bar says "Object Explorer Details" and "SQLQuery1.sql - D...HQEMP6\zaffa (54)*". The main pane contains the following SQL code:

```
DELETE FROM customer
WHERE customer_num=266
```

The status bar at the bottom shows "(1 row(s) affected)".

SQL – Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE, o sea cuyo customer_num sea 104.

```
DELETE FROM customer  
WHERE customer_num=104
```

WARNING

Cuidado con la
Integridad Referencial
resguardada por las
PK y las FK.

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'DESKTOP-MHQEMP6\SQLEXPRESS'. It highlights two specific sections with red boxes: 'Keys' under 'Tables' and 'Keys' under 'Tables' for the 'orders' table. In the center, the Object Explorer Details window shows the following T-SQL code:

```
DELETE FROM customer
WHERE customer_num=104
```

Below the code, the Messages pane displays an error message:

```
Msg 547, Level 16, State 0, Line 1
The DELETE statement conflicted with the REFERENCE constraint "FK__orders__customer__33D4B598". The conflict occurred in database "stores?NewVersion", table "dbo.orders", column 'customer_num'.
The statement has been terminated.
```

Break



Tablas Temporales

Son tablas creadas cuyos datos son de existencia temporal.
No son registradas en las tablas del diccionario de datos.
No es posible alterar tablas temporarias. Si eliminarlas y crear
los índices temporales que necesite una aplicación.
Las actualizaciones a una tabla temporal podrían no generar
ningún log transaccional si así se configurara.

Tipos de Tablas

- De Sesión (locales)
- Globales

Tipos de Creación

- Explícita
- Implícita

Tablas Temporales

Tipos de Tablas Temporales

De Sesión (locales)

Son visibles sólo para sus creadores durante la misma sesión (conexión) a una instancia del motor de BD.

Las tablas temporales locales se eliminan cuando el usuario se desconecta o cuando decide eliminar la tabla durante la sesión.

Globales

Las tablas temporales globales están visibles para cualquier usuario y sesión una vez creadas. Su eliminación depende del motor de base de datos que se utilice.

Tablas Temporales

Tipos de Creación

Creación Explícita.

Este tipo de creación se realizar mediante la instrucción CREATE. De manera explícita se deberá crear la tabla indicando el nombre, sus campos, tipos de datos y restricciones.

Creación Implícita

Se pueden crear tablas temporales a partir del resultado de una consulta SELECT.

Tablas Temporales

Por qué utilizarlas?

Como almacenamiento intermedio de Consultas Muy Grandes:

Por ejemplo, se tiene una consulta SELECT que realiza “**JOINS**” con ocho tablas.

Muchas veces las consultas con varios “**JOINS**” pueden funcionar de manera poco performante.

Una técnica para intentar es la de dividir una consulta grande en consultas más pequeñas. Si usamos tablas temporales, podemos crear tablas con resultados intermedios basados en consultas de menor tamaño, en lugar de intentar ejecutar una consulta única que sea demasiado grande y múltiples “**JOINS**”.

Tablas Temporales

Por qué utilizarlas? (Cont.)

Para optimizar accesos a una consulta varias veces en una aplicación:

Por ejemplo, usted está utilizando una consulta que tarda varios segundos en ejecutarse, pero sólo muestra un conjunto acotado de resultados, el cual desea utilizar en varias áreas de su procedimiento almacenado, pero cada vez que se llama se debe volver a ejecutar la consulta general.

Para resolver esto, puede ejecutar la consulta una sola vez en el procedimiento, llenando una tabla temporal, de esta manera se puede hacer referencia a la tabla temporal en varios lugares en su código, sin incurrir en una sobrecarga de resultados adicional.

Tablas Temporales

Por qué utilizarlas? (Cont.)

Para almacenar resultados intermedios en una aplicación:

Por ejemplo, usted está necesitando en un determinado proceso generar información que se irá actualizando y/o transformando en distintos momentos de la ejecución, sin querer actualizar o impactar a tablas reales de la BD hasta el final del procedimiento.

Para resolver esto, puede crear una tabla temporal de sesión durante la ejecución del procedimiento, realizando en ella inserciones, modificaciones, borrado y/o transformación de datos. Al llegar al final del procedimiento, con los datos existentes en la tabla temporal se actualizará la o las tablas físicas de la BD que corresponda.

Tablas Temporales

Ejemplo SQLServer – Tabla de Sesión

Creación Explícita

```
CREATE TABLE #ordenes_pendientes (
    N_orden INTEGER,
    N_cliente INTEGER,
    F_orden DATE,
    I_Total DECIMAL(15 , 2),
    C_estado SMALLINT,
    F_alta_audit TIMESTAMP,
    D_usuario VARCHAR(20) )
WITH NO LOG;
```

```
INSERT INTO #ordenes_Pendientes
SELECT * FROM ordenes WHERE c_estado = 1
```

Creación Implícita

```
SELECT *
INTO #ordenes_Pendientes
FROM ordenes
WHERE c_estado = 1
```

Tanto el ejemplo de creación explícita y el de implícita generarán la misma tabla temporal con los mismos datos.

SQL – DML Merge

La sentencia MERGE se utiliza esencialmente para realizar procesamientos batch (migraciones, carga de datos, apareos, etc.) de tablas.

La sintaxis (resumida) de la sentencia tiene el siguiente formato:

```
MERGE    <target_table>
        USING <table_source>
        ON <merge_search_condition>
        [ WHEN MATCHED [ AND <clause_search_condition> ]
            THEN <merge_matched> ]
        [ 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> ]
        [ <output_clause> ] ;
```

SQL – DML Merge

Pero pasemos a un ejemplo para entender mejor su funcionamiento y supongamos tener los siguientes requerimientos:

Dada una tabla **mergeFuente** y una tabla **mergeDestino** cuyas claves primarias en ambas es el atributo **código**:

Si el **código** de la tabla **mergeFuente** existe en la tabla **mergeDestino** y las **direcciones** son diferentes entonces actualizar la dirección en la tabla **mergeDestino**

Si el **código** de la tabla **mergeFuente** NO existe en la tabla **mergeDestino** entonces insertar en la tabla **mergeDestino** el registro de la tabla **mergeFuente**.

Si el **código** de la tabla **mergeDestino** NO existe en la tabla **mergeFuente** entonces borrar el registro de la tabla **mergeDestino**

SQL – DML Merge

Para ello, primeramente creamos las dos tablas, **mergeFuente** y **mergeDestino**.

```
create table mergeFuente
(codigo smallint PRIMARY KEY,
 nombre varchar(30) not null,
 direccion varchar(50)
);
create table mergeDestino
(codigo smallint PRIMARY KEY,
 nombre varchar(30) not null,
 direccion varchar(50),
 estado char(1) default 'A',
 observaciones varchar(50)
);

-- insertamos filas en la tabla fuente
insert into mergeFuente (codigo, nombre, direccion) values
(2, 'Ricardo Ruben', 'Paraguay 1888'),      -- modifica la direccion
(3, 'Juan Jose Jacinto', 'Terranova 765'),   -- no modificado
(8, 'Carola Sampietro', 'Arenales 1265');    -- nuevo

-- insertamos filas en la tabla destino
insert into mergeDestino (codigo, nombre, direccion) values
(1, 'Pepe', 'Venezuela 3456'),
(2, 'Ricardo Ruben', 'Cucha Cucha 234'),
(3, 'Juan Jose Jacinto', 'Terranova 765'),
(4, 'Violeta Rivarola', 'Quito 2112');
```

SQL – DML Merge

Ahora escribimos la sentencia MERGE según los requerimientos:

```
MERGE mergeDestino d
USING mergeFuente f
ON d.codigo = f.codigo
WHEN MATCHED AND d.direccion <> f.direccion THEN
    UPDATE
        SET d.direccion = f.direccion
WHEN NOT MATCHED BY TARGET THEN
    INSERT (codigo, nombre, direccion, estado, observaciones)
        VALUES (f.codigo, f.nombre, f.direccion, 'A', 'Nuevo')
WHEN NOT MATCHED BY SOURCE THEN
    DELETE;
```

Select * from mergeDestino;

codigo	nombre	direccion	estado	observaciones
2	Ricardo Ruben	Paraguay 1888	A	NULL
3	Juan Jose Jacinto	Terranova 765	A	NULL
8	Carola Sampietro	Arenales 1265	A	Nuevo

Podemos observar el resultado esperado.

Se actualizó la dirección de la fila de la tabla mergeDestino con código = 2;
Se insertó la fila con código = 8 y
Se borraron las filas con códigos 1 y 4.

Vamos a la Practica!!!

