

SQL - DML

Data Manipulation Language

Select Inner y Outer Join - Transactions

SQL DDL

Views - Snapshots

UTN - FRBA

Ing. en Sistemas de Información

Gestión de Datos

Prof.: Ing. Juan Zaffaroni

SQL – Hoy Veremos

- **DDL** – Data Definition Language
 - **VIEWS - SNAPSHOTS**
- **DML** – Data Manipulation Language
 - **SELECT**
 - INNER JOIN
 - OUTER JOIN
 - **TRANSACCIONES**

Views

Una view es un conjunto de columnas, ya sea reales o virtuales, de una misma tabla o no, con algún filtro determinado o no.

De esta forma, es una presentación adaptada de los datos contenidos en una o más tablas, o en otras vistas. Una vista toma la salida resultante de una consulta y la trata como una tabla.

Se pueden usar vistas en la mayoría de las situaciones en las que se pueden usar tablas.

Views

- Tiene un nombre específico
- No aloca espacio de almacenamiento.
- No contiene datos almacenados.
- Está definida por una consulta que consulta datos de una o varias tablas.

Views

Las vistas se pueden utilizar para:

- Suministrar un **nivel adicional de seguridad** restringiendo el acceso a un conjunto predeterminado de filas o columnas de una tabla.
- **Ocultar la complejidad** de los datos.
- **Simplificar sentencias** al usuario.
- Presentar los datos desde una **perspectiva diferente**.
- **Aislar a las aplicaciones** de los cambios en la tabla base.

Views

RESTRICCIONES:

- No se pueden crear índices en las Views (Depende el motor de BD)
- Una view depende de las tablas a las que se haga referencia en ella, si se elimina una tabla todas las views que dependen de ella se borrarán o se pasará a estado INVALIDO, dependiendo del motor. Lo mismo para el caso de borrar una view de la cual depende otra view. (depende del motor de BD)
- Algunas views tienen restringido los: Inserts, Deletes, Updates.
 - Aquellas que tengan joins
 - Una función agregada
 - Trigger INSTEAD OF (Lo vemos cuando veamos triggers)
- Al crear la view el usuario debe tener permiso de select sobre las columnas de las tablas involucradas.
- No es posible adicionar a una View las cláusulas de: ORDER BY y UNION. (depende del motor de BD)

Views

RESTRICCIONES:

- Tener en cuenta ciertas restricciones para el caso de Actualizaciones:
 - Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los inserts fallarían.
 - Si en la view no aparece la primary key los inserts podrían fallar.
 - Se puede borrar filas desde una view que tenga una columna virtual.
 - Con la opción WITH CHECK OPTION, se puede actualizar siempre y cuando el chequeo de la opción en el where sea verdadero.

Views

Ejemplo SQLSERVER

```
CREATE VIEW V_clientes_california  
(codigo, apellido, nombre)  
AS
```

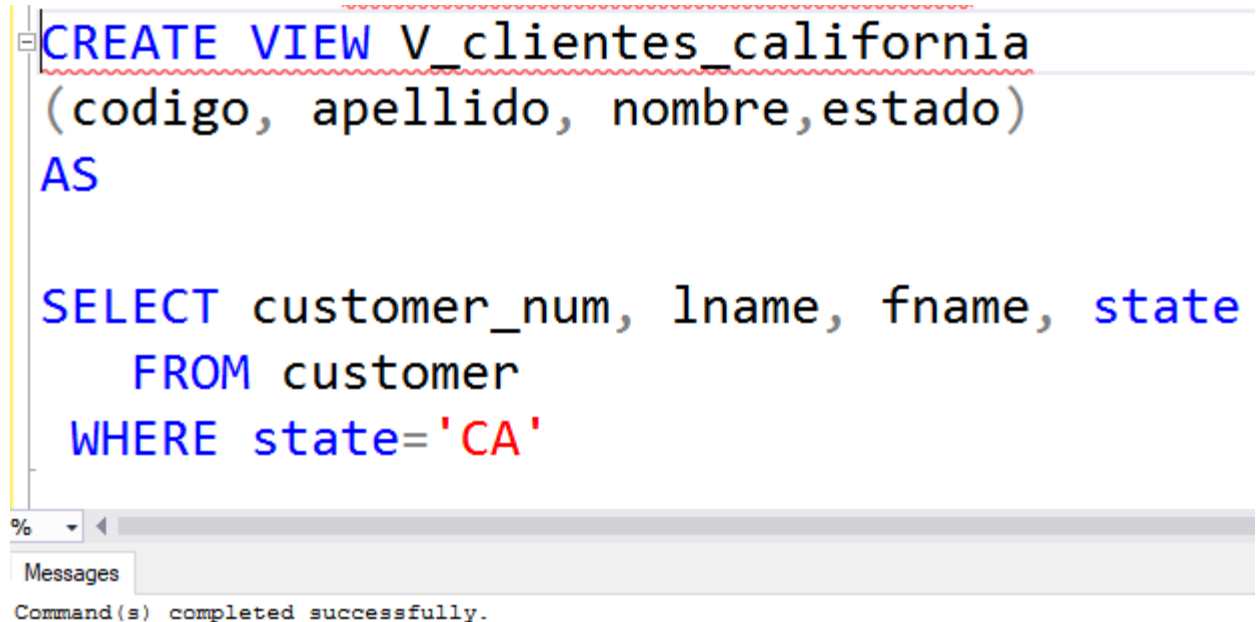
```
SELECT customer_num, lname, fname  
FROM customer  
WHERE state='CA'
```


Views

Ejemplo SQLSERVER

CREATE VIEW V_clientes_california
(codigo, apellido, nombre)
AS

SELECT customer_num, lname, fname
FROM customer
WHERE state='CA'



The screenshot shows a SQL Server Enterprise Manager interface. The main window displays the following SQL code:

```
CREATE VIEW V_clientes_california
(codigo, apellido, nombre, estado)
AS

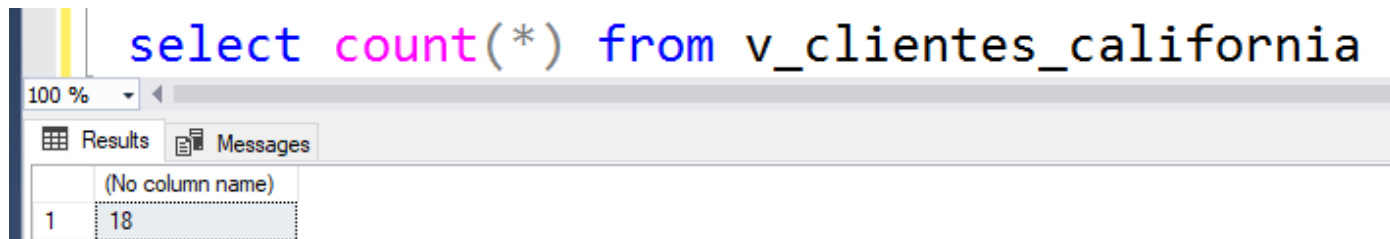
SELECT customer_num, lname, fname, state
FROM customer
WHERE state='CA'
```

Below the code editor, there is a "Messages" pane that displays the message: "Command(s) completed successfully."

Views

Ejemplo SQLSERVER

Hacemos una consulta sobre la View v_clientes_california



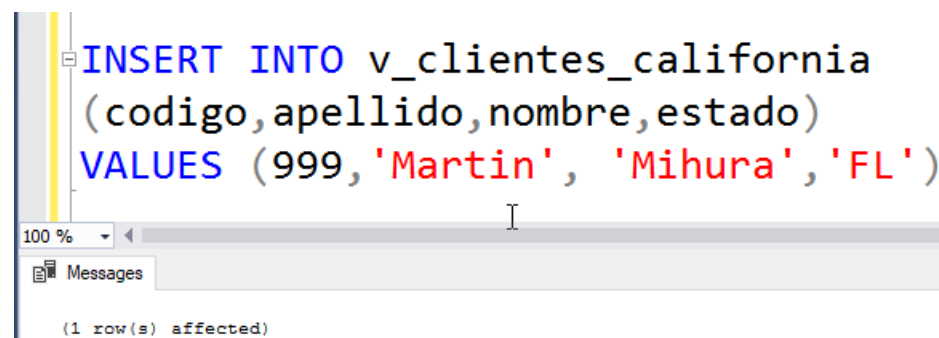
The screenshot shows a SQL query window with the following text:

```
select count(*) from v_clientes_california
```

Below the query, the 'Results' tab is active, displaying a single row with the value 18.

(No column name)
18

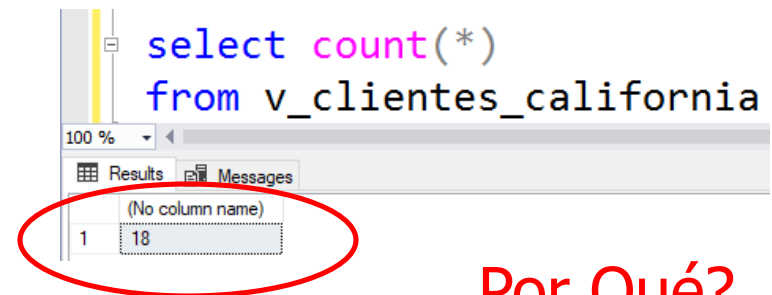
Insertamos a través de la View una fila en la tabla original customer los siguientes valores (999,'Martin', 'Mihura','FL').



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

```
INSERT INTO v_clientes_california  
(codigo,apellido,nombre,estado)  
VALUES (999,'Martin', 'Mihura','FL')
```

Below the query, the 'Messages' tab is active, displaying the message: (1 row(s) affected)



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

```
select count(*)  
from v_clientes_california
```

Below the query, the 'Results' tab is active, displaying a single row with the value 18. This row is circled in red.

(No column name)
18

Por Qué?

Views

Ejemplo SQLSERVER

Hacemos una consulta sobre la View v_clientes_california

```
select *  
from v_clientes_california where codigo=101
```

100 %

Results Messages

	codigo	apellido	nombre	estado
1	101	Pauli	Ludwig	CA

Haremos un Update sobre el cliente 101 de California cambiando su estado a NewYork.

```
UPDATE v_clientes_california  
set estado='NY'  
WHERE codigo=101
```

100 %

Messages

(1 row(s) affected)

```
select *  
from v_clientes_california  
where codigo=101
```

100 %

Results Messages

codigo	apellido	nombre	estado
--------	----------	--------	--------

Por Qué?

Views

Ejecutamos un insert a traves de una vista de una fila que la vista no puede mostrar por sus condiciones en el WHERE.

Ejecutamos un UPDATE en una fila a través de la vista cambiando un valor de un campo que es condicion en el WHERE.

A traves de la vista manipulamos datos que actualmente no pueden ser accedidos a traves de la vista.

```
select customer_num, lname, fname, state
from customer
where customer_num in (101,999)
```

	customer_num	lname	fname	state
1	101	Pauli	Ludwig	NY
2	999	Martin	Mihura	FL

```
select codigo, apellido, nombre, estado
from v_clientes_california
where codigo in (101,999)
```

	codigo	apellido	nombre	estado

Es esto correcto, no les hace ruido?? Discutamos un poco

Views

```
CREATE VIEW V_clientes_california_WCK  
(codigo, apellido, nombre, estado)  
AS
```

```
SELECT customer_num, lname, fname, state  
FROM customer  
WHERE state='CA'
```

WITH CHECK OPTION
realiza un chequeo de
integridad de los datos a
insertar o modificar, los
cuales deben cumplir
con las condiciones del
WHERE de la vista.

WITH CHECK OPTION

```
INSERT INTO v_clientes_california_WCK  
(codigo, apellido, nombre, estado)  
VALUES (999, 'Martin', 'Mihura', 'FL')
```

100 %
Messages
Msg 550, Level 16, State 1, Line 36
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.

The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

Views

```
CREATE VIEW V_clientes_california_WCK  
(codigo, apellido, nombre, estado)  
AS
```

```
SELECT customer_num, lname, fname, state  
FROM customer  
WHERE state='CA'
```

WITH CHECK OPTION
realiza un chequeo de
integridad de los datos a
insertar o modificar, los
cuales deben cumplir
con las condiciones del
WHERE de la vista.

WITH CHECK OPTION

```
UPDATE v_clientes_california_WCK  
set estado='NY'  
WHERE codigo=101
```

Msg 550, Level 16, State 1, Line 15

The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.

The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

Snapshots/Materialized Views /Summarized Tables

Los snapshots, también llamados vistas materializadas o tablas sumariadas, son objetos del esquema de una BD que pueden ser usados para sumarizar, precomputar, distribuir o replicar datos. Se utilizan sobre todo en DataWarehouse, sistemas para soporte de toma de decisión, y para computación móvil y/o distribuida.

Consumen espacio de almacenamiento en disco. Deben ser recalculadas o refrescadas cuando los datos de las tablas master cambian. Pueden ser refrescadas en forma manual, o a intervalos de tiempo definidos dependiendo el motor de BD.

Snapshots/Materialized Views /Summarized Tables

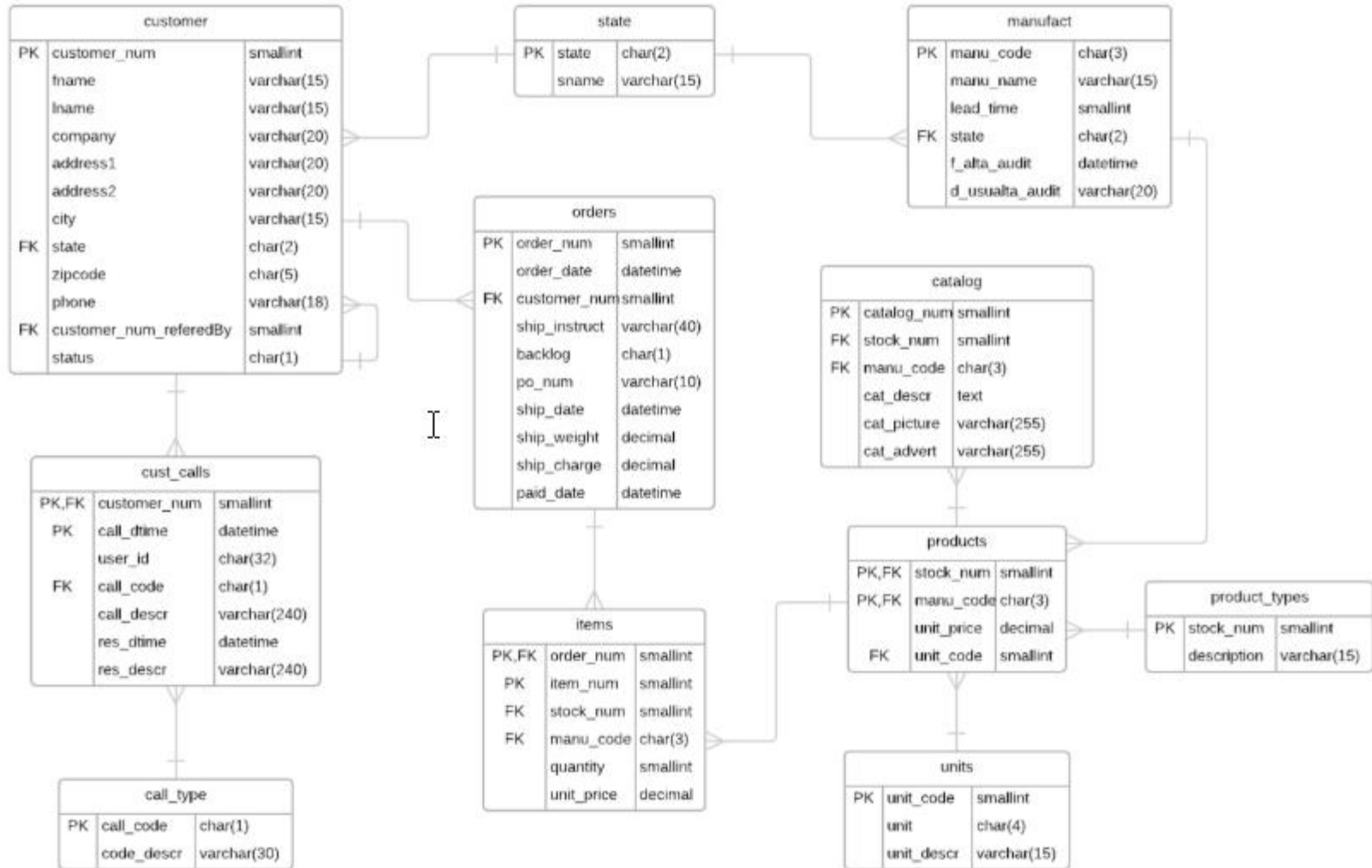
Ej. Objeto Materialized View en Oracle

```
CREATE MATERIALIZED VIEW ST_total_ordenes_clientes  
BUILD DEFERRED  
REFRESH COMPLETE  
AS SELECT A.C_cliente, B.D_apellido, B.D_nombre,  
          SUM(A.I_total) AS Total_ordenes  
FROM ordenes A, clientes B  
WHERE A.C_cliente = B.C_cliente  
GROUP BY A.C_cliente, B.D_apellido, B.D_nombre ;
```

La cláusula **BUILD DEFERRED** causa que en la creación de la vista materializada no se realice el INSERT de los datos, que se realizará en el momento que se ejecute el próximo refresco. El valor por defecto es **BUILD IMMEDIATE**, que sí inserta los datos.

La cláusula **REFRESH COMPLETE** indica que el método de refresco será completo, que ejecuta nuevamente la subconsulta. De otra forma, la opción **FAST** le indica actualizar de acuerdo a los cambios que han ocurrido en la tabla master.

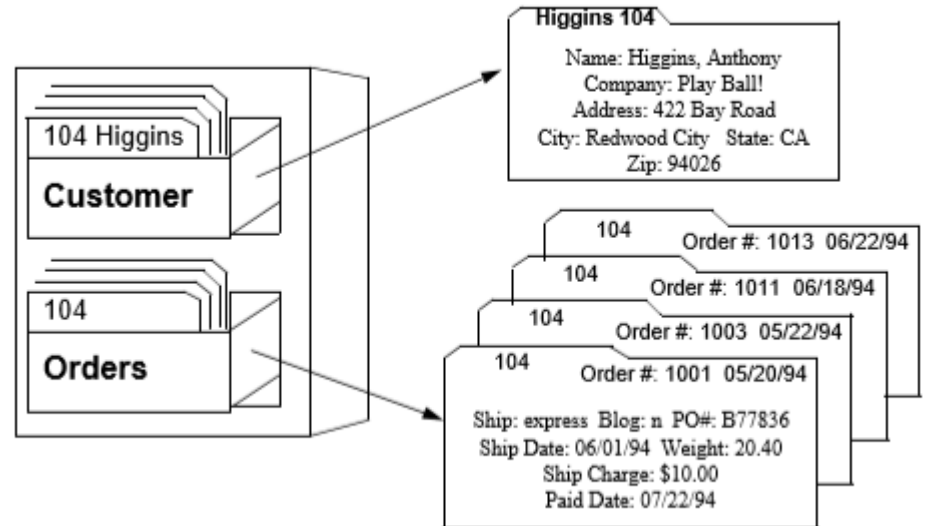
Base de Datos de Ejemplo



SQL – DML

- DML – Data Manipulation Language
 - SELECT
 - INNER JOIN
 - OUTER JOIN

Si queremos hacer coincidir filas de dos o mas tablas a partir de un atributo con valores comunes, por ej.



A partir del customer_num de la tabla **orders** obtener el lname y fname de cada cliente de la tabla **customer**.

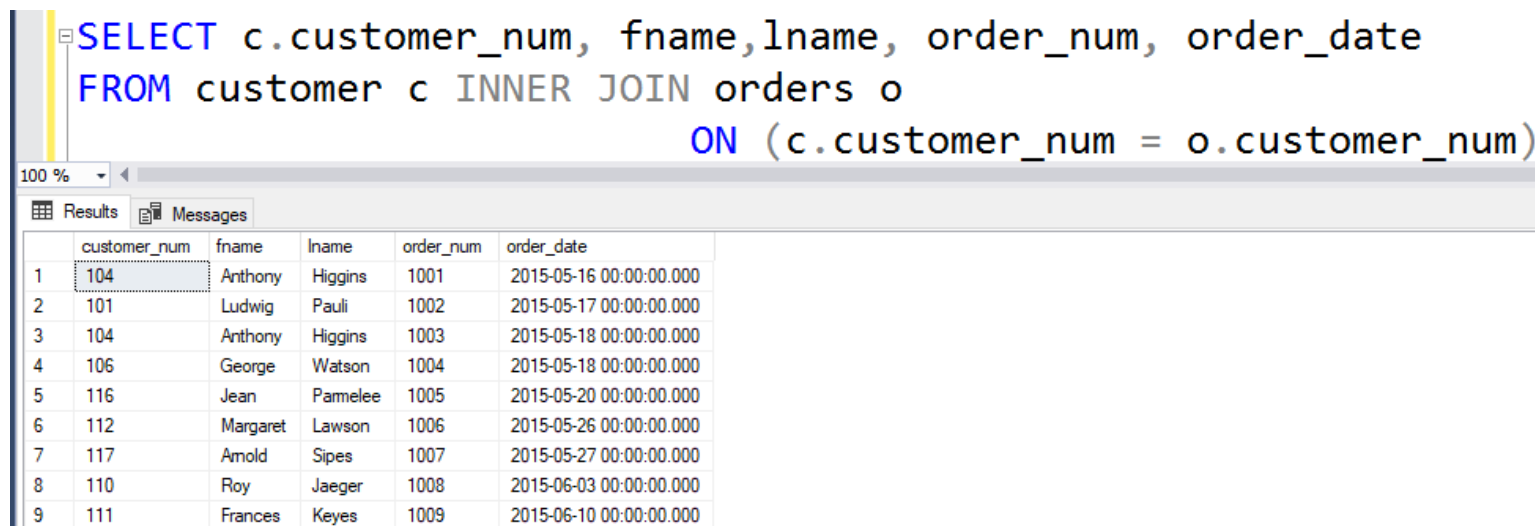
SQL – Operador SELECT de más de 1 tabla

INNER JOIN (clave simple)

Se realiza un **macheo de las filas** de una tabla que coincidan a través de un atributo o combinación de atributos con filas de otras tabla.

El INNER JOIN solo devolverá las filas que coincidan.

```
SELECT c.customer_num, fname, lname, order_num, order_date  
FROM customer c INNER JOIN orders o  
ON (c.customer_num = o.customer_num)
```



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

```
SELECT c.customer_num, fname, lname, order_num, order_date  
FROM customer c INNER JOIN orders o  
ON (c.customer_num = o.customer_num)
```

Below the query window, the 'Results' tab is active, displaying a table with 6 columns: customer_num, fname, lname, order_num, and order_date. The table contains 9 rows of data. The first row is highlighted with a blue selection bar.

	customer_num	fname	lname	order_num	order_date
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000
2	101	Ludwig	Pauli	1002	2015-05-17 00:00:00.000
3	104	Anthony	Higgins	1003	2015-05-18 00:00:00.000
4	106	George	Watson	1004	2015-05-18 00:00:00.000
5	116	Jean	Pamelee	1005	2015-05-20 00:00:00.000
6	112	Margaret	Lawson	1006	2015-05-26 00:00:00.000
7	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000
8	110	Roy	Jaeger	1008	2015-06-03 00:00:00.000
9	111	Frances	Keyes	1009	2015-06-10 00:00:00.000

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (clave compuesta)

Ejemplo 2: código de producto, total vendido y el código de unidad de medida. Para esto deberemos tomar la unit_code de la tabla products.

```
SELECT i.stock_num, i.manu_code, SUM(i.unit_price*quantity) tot_producto  
      , unit_code  
from items i INNER JOIN products p  
      ON (i.stock_num = p.stock_num AND i.manu_code=p.manu_code)  
GROUP BY i.stock_num, i.manu_code, unit_code
```

100 %

Results Messages

	stock_num	manu_code	tot_producto	unit_code
1	1	HRO	750.00	6
2	1	HSK	800.00	6
3	1	SMT	900.00	6
4	2	HRO	252.00	13
5	3	HSK	720.00	9
6	4	HRO	960.00	13
7	4	HSK	1920.00	13
8	5	ANZ	3484.80	19
9	5	NRG	3500.00	19
10	5	SMT	1035.00	19

ER diagram showing relationships between items, products, and units tables:

- items** table: PK,FK order_num (smallint), PK item_num (smallint), FK stock_num (smallint), FK manu_code (char(3)), quantity (smallint), unit_price (decimal).
- products** table: PK,FK stock_num (smallint), PK,FK manu_code (char(3)), unit_price (decimal), FK unit_code (smallint).
- units** table: PK unit_code (smallint), unit (char(4)), unit_descr (varchar(15)).

Relationships: items to products (1:M), items to units (1:M), products to units (1:M).

Por qué utilizamos alias de tabla en algunas columnas?

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (mas de dos tablas)

Ejemplo 3: código de producto, total vendido y la descripción de la unidad de medida. Para esto deberemos tomar la unit_code de la tabla products y el unit_descr de la tabla units.

```
SQLQuery1.sql - D:\HQEMP6\zaffa (54))*  
SELECT i.stock_num, i.manu_code, SUM(i.unit_price*quantity) tot_producto  
      , unit_descr  
from items i INNER JOIN products p  
      ON (i.stock_num = p.stock_num AND i.manu_code=p.manu_code)  
      INNER JOIN units u ON (p.unit_code=u.unit_code)  
GROUP BY i.stock_num, i.manu_code, unit_descr
```

100 %

Results

Messages

	stock_num	manu_code	tot_producto	unit_descr
1	1	HRO	750.00	10 gloves/case
2	1	HSK	800.00	10 gloves/case
3	1	SMT	900.00	10 gloves/case
4	2	HRO	252.00	24/case
5	3	HSK	720.00	12/case
6	4	HRO	960.00	24/case
7	4	HSK	1920.00	24/case

```
graph TD
    items -- "PK,FK order_num, smallint; PK item_num, smallint; FK stock_num, smallint; FK manu_code, char(3)" --> products
    products -- "PK,FK stock_num, smallint; PK,FK manu_code, char(3); unit_price, decimal; FK unit_code, smallint" --> units
    units -- "PK unit_code, smallint; unit, char(4); unit_descr, varchar(15)"
```

The diagram illustrates the relationships between three tables: items, products, and units. The items table has primary keys order_num and item_num, and foreign keys stock_num and manu_code. The products table has primary keys stock_num and manu_code, and a foreign key unit_code. The units table has a primary key unit_code and attributes unit and unit_descr. Red boxes highlight the foreign key relationships between items and products, and between products and units.

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (mas de dos tablas)

Ejemplo 3: cod_cliente, apellido y nombre de cliente, orden de compra, fecha emisión, nro. de stock, código fabricante, nombre_fabricante, descripción tipo producto, nro de ítem, cantidad y precio unitario.

```
SELECT c.customer_num, fname, lname, o.order_num, order_date, item_num,  
       i.stock_num, i.manu_code, manu_name, pt.description, item_num,  
       quantity, i.unit_price  
FROM orders o JOIN customer c ON (o.customer_num = c.customer_num)  
              JOIN items i ON (o.order_num=i.order_num)  
              JOIN product_types pt ON (i.stock_num=pt.stock_num)  
              JOIN manufact m ON (i.manu_code=m.manu_code)
```

100 %													
Results Messages													
	customer_num	fname	lname	order_num	order_date	item_num	stock_num	manu_code	manu_name	description	item_num	quantity	unit_price
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
2	101	Ludwig	Pauli	1002	2015-05-17 00:00:00.000	1	4	HSK	Husky	football	1	1	960.00
3	104	Anthony	Higgins	1003	2015-05-18 00:00:00.000	1	9	ANZ	Anza	volleyball net	1	1	20.00
4	106	George	Watson	1004	2015-05-18 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
5	116	Jean	Pamelee	1005	2015-05-20 00:00:00.000	1	5	NRG	Norge	tennis racquet	1	10	280.00
6	112	Margaret	Lawson	1006	2015-05-26 00:00:00.000	1	5	SMT	Smith	tennis racquet	1	5	125.00
7	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
8	110	Roy	Jaeger	1008	2015-06-03 00:00:00.000	1	8	ANZ	Anza	volleyball	1	1	840.00
9	111	Frances	Keyes	1009	2015-06-10 00:00:00.000	1	1	SMT	Smith	baseball gloves	1	1	450.00
10	115	Alfred	Grant	1010	2015-06-13 00:00:00.000	1	6	SMT	Smith	tennis ball	1	1	36.00

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (mas de dos tablas con condiciones adicionales en WHERE)

Ejemplo 4: Mismo ejemplo anterior pero con Condiciones, cliente de nombre Husky, órdenes del mes de mayo del 2015.

```
SQLQuery1.sql - D...HQEMP6\zaffa (54))*
SELECT c.customer_num, fname, lname, o.order_num, order_date, item_num,
       i.stock_num, i.manu_code, manu_name, pt.description, item_num,
       quantity, i.unit_price
FROM orders o JOIN customer c ON (o.customer_num = c.customer_num)
              JOIN items i ON (o.order_num=i.order_num)
              JOIN product_types pt ON (i.stock_num=pt.stock_num)
              JOIN manufact m ON (i.manu_code=m.manu_code)
WHERE manu_name='Hero' AND MONTH(order_date)=5 AND YEAR(order_date)=2015
```

	customer_num	fname	lname	order_num	order_date	item_num	stock_num	manu_code	manu_name	description	item_num	quantity	unit_price
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
2	106	George	Watson	1004	2015-05-18 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
3	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
4	106	George	Watson	1004	2015-05-18 00:00:00.000	2	2	HRO	Hero	baseball	2	1	126.00
5	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	2	2	HRO	Hero	baseball	2	1	126.00
6	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	4	4	HRO	Hero	football	4	1	480.00
7	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	5	7	HRO	Hero	basketball	5	1	600.00

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (mas de dos tablas con condiciones adicionales en WHERE)

Ejemplo 4: Mismo ejemplo anterior pero con Condiciones, cliente de nombre Husky, órdenes del mes de mayo del 2015 y precio unitario mayor que 126.

```
SQLQuery1.sql - D:\HQEMP6\zaffa (54))*
```

```
SELECT c.customer_num, fname, lname, o.order_num, order_date, item_num,
       i.stock_num, i.manu_code, manu_name, pt.description, item_num,
       quantity, i.unit_price
FROM orders o JOIN customer c ON (o.customer_num = c.customer_num)
              JOIN items i ON (o.order_num=i.order_num)
              JOIN product_types pt ON (i.stock_num=pt.stock_num)
              JOIN manufact m ON (i.manu_code=m.manu_code)
WHERE manu_name='Hero' AND MONTH(order_date)=5 AND YEAR(order_date)=2015
      AND unit_price>126
```

100 %

Results Messages

	customer_num	fname	lname	order_num	order_date	item_num	stock_num	manu_code	manu_name	description	item_num	quantity	unit_price
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
2	106	George	Watson	1004	2015-05-18 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
3	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
4	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	4	4	HRO	Hero	football	4	1	480.00
5	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000	5	7	HRO	Hero	basketball	5	1	600.00

Break



SQL – Operador SELECT de más de 1 tabla

OUTER JOIN

El Outer join, mostrara todas las filas de la **tabla dominante** macheen o no con la otra tabla.

El Outer puede ser **LEFT**. (tabla izquierda dominante), **RIGHT** (tabla derecha dominante) o **FULL** (ambas tablas dominantes)

OUR CUSTOMER table has :

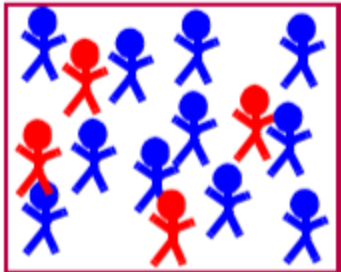


Customers that have placed orders



Customers that have NOT placed orders!

CUSTOMER



■ Join



```
SELECT c.customer_num, fname,lname,COUNT(order_num)
cantOrdenes
FROM customer c INNER JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

■ Outer Join



```
SELECT c.customer_num, fname,lname,COUNT(order_num)
cantOrdenes
FROM customer c LEFT JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

SQL – Operador SELECT de más de 1 tabla

OUTER JOIN

Comparemos resultados.

SQLQuery1.sql - D:\...HQUEMP6\zaffa (54)*

```
SELECT c.customer_num, fname,lname,COUNT(order_num) cantOrdenes
FROM customer c INNER JOIN orders o
      ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

100 %

	customer_num	fname	lname	cantOrdenes
1	101	Ludwig	Pauli	1
2	104	Anthony	Higgins	4
3	106	George	Watson	2
4	110	Roy	Jaeger	2
5	111	Frances	Keyes	1
6	112	Margaret	Lawson	1
7	115	Alfred	Grant	1
8	116	Jean	Parmelee	1
9	117	Arnold	Sipes	2
10	119	Bob	Shorter	1
11	120	Fred	Jewell	1
12	121	Jason	Wallack	1
13	122	Cathy	O Brian	1
14	123	Marvin	Hanlon	1
15	124	Chris	Putnum	1
16	126	Eileen	Neele	1
17	127	Kim	Satfer	1

Query executed successfully.

Inner Join 17 clientes que compraron.

SQLQuery1.sql - D:\...HQUEMP6\zaffa (54)*

```
SELECT c.customer_num, fname,lname,COUNT(order_num) cantOrdenes
FROM customer c LEFT JOIN orders o
      ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

100 %

	customer_num	fname	lname	cantOrdenes
1	101	Ludwig	Pauli	1
2	102	Carole	Sadler	0
3	103	Philip	Cumie	0
4	104	Anthony	Higgins	4
5	105	Raymond	Vector	0
6	106	George	Watson	2
7	107	Charles	Ream	0
8	108	Donald	Quinn	0
9	109	Jane	Miller	0
10	110	Roy	Jaeger	2
11	111	Frances	Keyes	1
12	112	Margaret	Lawson	1
13	113	Lana	Beatty	0
14	114	Frank	Albertson	0
15	115	Alfred	Grant	1
16	116	Jean	Parmelee	1
17	117	Arnold	Sipes	2

Query executed successfully.

Left Join 28 clientes, 17 con cantOrdenes mayor que 0 y 11 con cantOrdenes=0

SQL – Operador SELECT de más de 1 tabla

JOIN AUTO REFERENCIADO (SELF REFERENCING JOIN)

La tabla customer tiene un atributo customer_num_referredBy, que nos indica quien fue el cliente que lo referencio.

Podríamos armar una consulta que nos diga nombre y apellido del referido y de quien lo referencio.

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

```
SELECT c2.lname+', '+c2.fname Padrino, c1.lname+', '+c1.fname Referido
FROM customer c1 JOIN customer c2
ON (c1.customer_num_referredBy =c2.customer_num)
```

Below the query, the results are displayed in a table with two columns: 'Padrino' and 'Referido'. The results are as follows:

	Padrino	Referido
1	Pauli, Ludwig	Sadler, Carole
2	Pauli, Ludwig	Curie, Philip
3	Curie, Philip	Higgins, Anthony
4	Curie, Philip	Vector, Raymond
5	Curie, Philip	Watson, George
6	Ream, Charles	Quinn, Donald
7	Ream, Charles	Miller, Jane
8	Keyes, Frances	Lawson, Margaret
9	Keyes, Frances	Beatty, Lana
10	Keyes, Frances	Albertson, Frank
11	Grant, Alfred	Pamelee, Jean
12	Grant, Alfred	Sipes, Arnold
13	Grant, Alfred	Baxter, Dick
14	Grant, Alfred	Shorter, Bob
15	Jewell, Fred	Wallack, Jason
16	Wallack, Jason	O Brian, Cathy
17	Wallack, Jason	Hanlon, Marvin

At the bottom of the window, a status bar indicates: 'Query executed successfully. DESKTOP-MHQEMP6\SQLEXPRESS ... DESKTOP-MHQEMP6\zaffa ... stores7NewVersion 00:00:00 21 rows'.

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (COLUMNAS AMBIGUAS)

Cuando un atributo existe en mas de una tabla del SELECT, es obligatorio identificar de que tabla lo estamos tomando, utilizando el punto como separador (dot notation) **tabla.columna** o **alias.columna**.

```
SQLQuery1.sql - D...HQEMP6\zaffa (54))*  
  
SELECT customer_num, fname,lname,order_num  
FROM customer c LEFT JOIN orders o  
ON (c.customer_num = o.customer_num)
```

100 %
Messages
Msg 209, Level 16, State 1, Line 4
Ambiguous column name 'customer_num'.

```
SQLQuery1.sql - D...HQEMP6\zaffa (54))*  
  
SELECT c.customer_num, fname,lname,order_num  
FROM customer c LEFT JOIN orders o  
ON (c.customer_num = o.customer_num)
```

100 %
Results Messages

	customer_num	fname	lname	order_num
1	101	Ludwig	Pauli	1002
2	102	Carole	Sadler	NULL
3	103	Philip	Curie	NULL
4	104	Anthony	Higgins	1001
5	104	Anthony	Higgins	1003
6	104	Anthony	Higgins	1011
7	104	Anthony	Higgins	1013
8	105	Raymond	Vector	NULL
9	106	George	Watson	1004
10	106	George	Watson	1014
11	107	Charles	Ream	NULL
12	108	Donald	Quinn	NULL
13	109	Jane	Miller	NULL
14	110	Roy	Jaeger	1008
15	110	Roy	Jaeger	1015
16	111	Frances	Keyes	1009
17	112	Margaret	Lawson	1006

SQL – Operador SELECT de más de 1 tabla

PRODUCTO CARTESIANO

El producto cartesiano es muy costoso para el motor de base de datos. En el caso que deban realizar alguno, deben tratar de achicar el working set lo máximo que puedan, proyectando solo las columnas que necesiten y condicionando con WHERE lo que pudiesen. **RECOMENDACIÓN NO LO UTILICEN.**

```
SELECT *
FROM customer c, orders o
```

Cientes (12 cols)+Ordenes(10 cols)= 22 columnas
28 clientes * 23 ordenes =644 filas
Que pasaría si fuesen 1000 clientes con 100000 ordenes?

	customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status	order_num	order_date
1	101	Ludwig	Pauli	All Sports Supplies	213 Erstwld Court		Sunnyvale	CA	94086	408-789-8075	NULL	NULL	1001	2015-05-16 00:00:00.00
2	102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117	415-822-1289	101	NULL	1001	2015-05-16 00:00:00.00
3	103	Philip	Curie	Phil s Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543	101	NULL	1001	2015-05-16 00:00:00.00
4	104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100	103	NULL	1001	2015-05-16 00:00:00.00
5	105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249	103	NULL	1001	2015-05-16 00:00:00.00
6	106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789	103	NULL	1001	2015-05-16 00:00:00.00
7	107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	415-356-9876	NULL	NULL	1001	2015-05-16 00:00:00.00
8	108	Donald	Quinn	Quinn s Sports	587 Alvarado		Redwood City	CA	94063	415-544-8729	107	NULL	1001	2015-05-16 00:00:00.00
9	109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789	107	NULL	1001	2015-05-16 00:00:00.00
10	110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	415-743-3611	NULL	NULL	1001	2015-05-16 00:00:00.00
11	111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245	NULL	NULL	1001	2015-05-16 00:00:00.00
12	112	Margaret	Lawson	UTN	234 Wyandotte Way		Los Altos	CA	94022	5555-5555	111	NULL	1001	2015-05-16 00:00:00.00
13	113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	415-356-9982	111	NULL	1001	2015-05-16 00:00:00.00
14	114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	415-886-6677	111	NULL	1001	2015-05-16 00:00:00.00
15	115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	415-356-1123	NULL	NULL	1001	2015-05-16 00:00:00.00
16	116	Jean	Pamellee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	415-534-8822	115	NULL	1001	2015-05-16 00:00:00.00
17	117	Arnold	Spice	Kids Kapers	950 Lutter Court		Redwood City	CA	94062	415-345-4578	115	NULL	1001	2015-05-16 00:00:00.00

Query executed successfully.

DESKTOP-MHQEMP6\SQLXEXPRESS ... | DESKTOP-MHQEMP6\zaffa ... | stores7NewVersion | 00:00:00 | 644 rows

SQL – Hoy Veremos

- DDL – Data Definition Language
 - VIEWS - SNAPSHTS
- DML – Data Manipulation Language
 - SELECT
 - INNER JOIN
 - OUTER JOIN
 - TRANSACCIONES

Funciones del motor de base de datos y del DBMS en su conjunto

Mecanismos para garantizar la consistencia de datos

El DBMS cuenta con distintos mecanismos para poder asegurar la consistencia de los datos que existentes en la/s Base/s de Datos.

Conceptos relacionados con la consistencia de datos

- **TRANSACCIONES**

- Es un conjunto de sentencias SQL que se ejecutan atómicamente en una unidad lógica de trabajo. Partiendo de que una transacción lleva la base de datos de un estado correcto a otro estado correcto, el motor posee mecanismos de manera de garantizar que la operación completa se ejecute o falle, no permitiendo que queden datos inconsistentes.

- Cada sentencia de alteración de datos (insert, update o delete) es una transacción en sí misma (singleton transaction).

- **LOGS TRANSACCIONALES:** Es un registro donde el motor almacena la información de cada operación llevada a cabo con los datos

- **RECOVERY:** Método de recuperación ante caídas.

Funciones del motor de base de datos y del DBMS en su conjunto

Transacciones (Cont.)

Para definir una transacción debemos definir un conjunto de instrucciones precedidas por la sentencia **BEGIN TRANSACTION**, de esta manera las sentencias a continuación se ejecutarán de forma atómica. Pero para lo que es el concepto de transacción, una transacción puede finalizar correctamente o puede fallar; en el caso de finalizar correctamente, todos los datos se actualizarían en la base y en caso de fallar se deberían deshacer todos los cambios hasta el comienzo de la transacción. Para manejar estas acciones contamos con la sentencia **COMMIT TRANSACTION** para actualizar los datos en la base de datos, y con la sentencia **ROLLBACK TRANSACTION** para deshacer nuestra transacción.

(Ej de Sintaxis en SqlServer)

Funciones del motor de base de datos y del DBMS en su conjunto

Transacciones (Cont.)

Ejemplo:

```
--Declaración de variables
DECLARE @IMPORTE REAL;
DECLARE @FACTURA INT;
DECLARE @FECHA VARCHAR(8);
DECLARE @CLIENTE INT;

--Harcodeo de inicalización
SET @FACTURA = 353;
SET @FECHA = '20120601';
SET @CLIENTE = 15;

BEGIN TRANSACTION --Comienzo de transacción

INSERT INTO entregas(numero, fecha, cliente)
VALUES (@FACTURA, @FECHA, @CLIENTE)

SET @IMPORTE = Calcular_Importe(@FACTURA)

IF @IMPORTE > 0 THEN
    INSERT INTO facturas(n_factura, imp_total)
    VALUES (@FACTURA, @IMPORTE)
    COMMIT TRANSACTION --Se insertan los datos en la base
ELSE
    PRINT 'ERROR! IMPORTE NO VALIDO'
    ROLLBACK TRANSACTION --Se vuelven al punto de inicio
END
```

Para nuestro ejemplo, se realiza una inserción en la tabla de entregas con la factura entregada y luego se calcula el importe a partir de una función. Si el importe es mayor a 0 la factura se inserta en la tabla de facturas y la transacción se confirma con un *commit*; caso contrario se imprime el mensaje de error y la transacción se vuelve al estado inicial mediante un *rollback*.

Funciones del motor de base de datos y del DBMS en su conjunto

Transacciones (Cont.)

Algunos motores de base de datos permiten el manejo de **Transacciones Anidadas**.

Ejemplo:

```
CREATE TABLE #numeros (num int)
```

```
BEGIN TRAN T1
```

```
    INSERT INTO #numeros VALUES (1)
```

```
    BEGIN TRAN T2
```

```
        INSERT INTO #numeros VALUES (2)
```

```
    COMMIT TRAN --Confirma T2
```

```
ROLLBACK TRAN--Deshace T1 que contiene a T2; entonces también la deshace
```

En nuestro caso, la transacción T2 finaliza correctamente dentro de T1; pero en el momento de confirmar la transacción T1 se produce un rollback que deshace a T2 volviendo a la tabla a un estado inicial sin valores cargados.

(Ej de Sintaxis en SqlServer)

Funciones del motor de base de datos y del DBMS en su conjunto

Transacciones (Cont.)

Algunos motores de base de datos permiten establecer puntos intermedios de guardado de información.

Ejemplo:

```
CREATE TABLE #numeros (num int)

BEGIN TRAN

INSERT INTO #numeros VALUES (2)

SAVE TRAN N2 --Guardo estado actual a N2

    BEGIN TRAN
        INSERT INTO #numeros VALUES (3)
    ROLLBACK TRAN N2 --Deshago la transacción actual hasta N2

INSERT INTO #numeros VALUES (4)

COMMIT TRAN
```

Es posible realizar más de un *Save Tran* en cada transacción y se puede elegir a cual se desea volver en cual momento.

(Ej de Sintaxis en SqlServer)

Funciones del motor de base de datos y del DBMS en su conjunto

Mecanismos de recuperación (RECOVERY)

Es un mecanismo provisto por los motores de Base de Datos que se ejecuta en cada inicio del motor de forma automática como dispositivo de tolerancia a fallas.

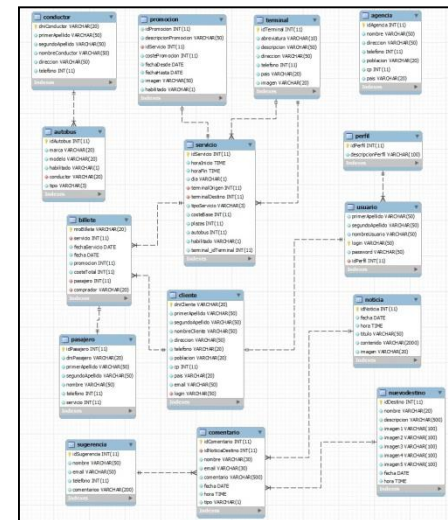
Sus objetivos son los siguientes:

- Retornar al Motor de Base de datos al punto consistente más reciente. (checkpoint = punto en el que el motor sincronizó memoria y disco)
- Utilizar los logs transaccionales para retornar el motor de base de datos a un estado lógico consistente, realizando un "rolling forwards" de las transacciones ocurridas con éxito luego del checkpoint más reciente y realizar un "rolling back" de las transacciones que no hayan sido exitosas.

Propiedades de un RDBMS

- Foco en la ejecución de **Transacciones**

- **Atomicidad**
- **Consistencia**
- **Isolation** (Aislamiento)
- **Durabilidad**



Vamos a la Practica!!!

