

SQL - DML
Data Manipulation Language
Subqueries – UNION – INTERSECT – EXCEPT
SQL DDL
Índices

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

Prof.: Ing. Juan Zaffaroni

SQL – Hoy Veremos

- DDL – Data Definition Language
 - INDICES
- DML – Data Manipulation Language
 - SUBQUERIES en
 - SELECT
 - DELETE
 - UPDATE
 - INSERT
 - Operadores Multi Select
 - UNION
 - INTERSECT
 - EXCEPT

Indices

Los índices son estructuras opcionales asociadas a una tabla.

La función de los índices es la de permitir un acceso más rápido a los datos de una tabla, se pueden crear distintos tipos de índices sobre uno o más campos.

Los índices son lógica y físicamente independientes de los datos en la tabla asociada. Se puede crear o borrar un índice en cualquier momento sin afectar a las tablas base o a otros índices.

Indices (Cont.)

TIPOS DE INDICES

Btree Index

Estructura de índice estándar y más utilizada.

Btree Cluster Index

Este tipo de índice provoca al momento de su creación que físicamente los datos de la tabla sean ordenados por el mismo. (Informix / SQLServer / DB2)

Bitmap Index (Oracle)

Son utilizados para pocas claves con muchas repeticiones

Cada bit en el Bitmap corresponde a una fila en particular.

Si el bit esta en on significa que la fila con el correspondiente rowid tiene el valor de la clave.

Hash Index (MySql)

Están implementados en tablas de hash y se basan en otros indices Btree existentes para una tabla. Si una tabla entra íntegramente en memoria, la manera más rápida de ejecutar consultas sobre ella es usando un hash index.

Indices (Cont.)

TIPOS DE INDICES (Cont.)

Functional Index / Function based Index

Son indices cuya clave deriva del resultado de una función.

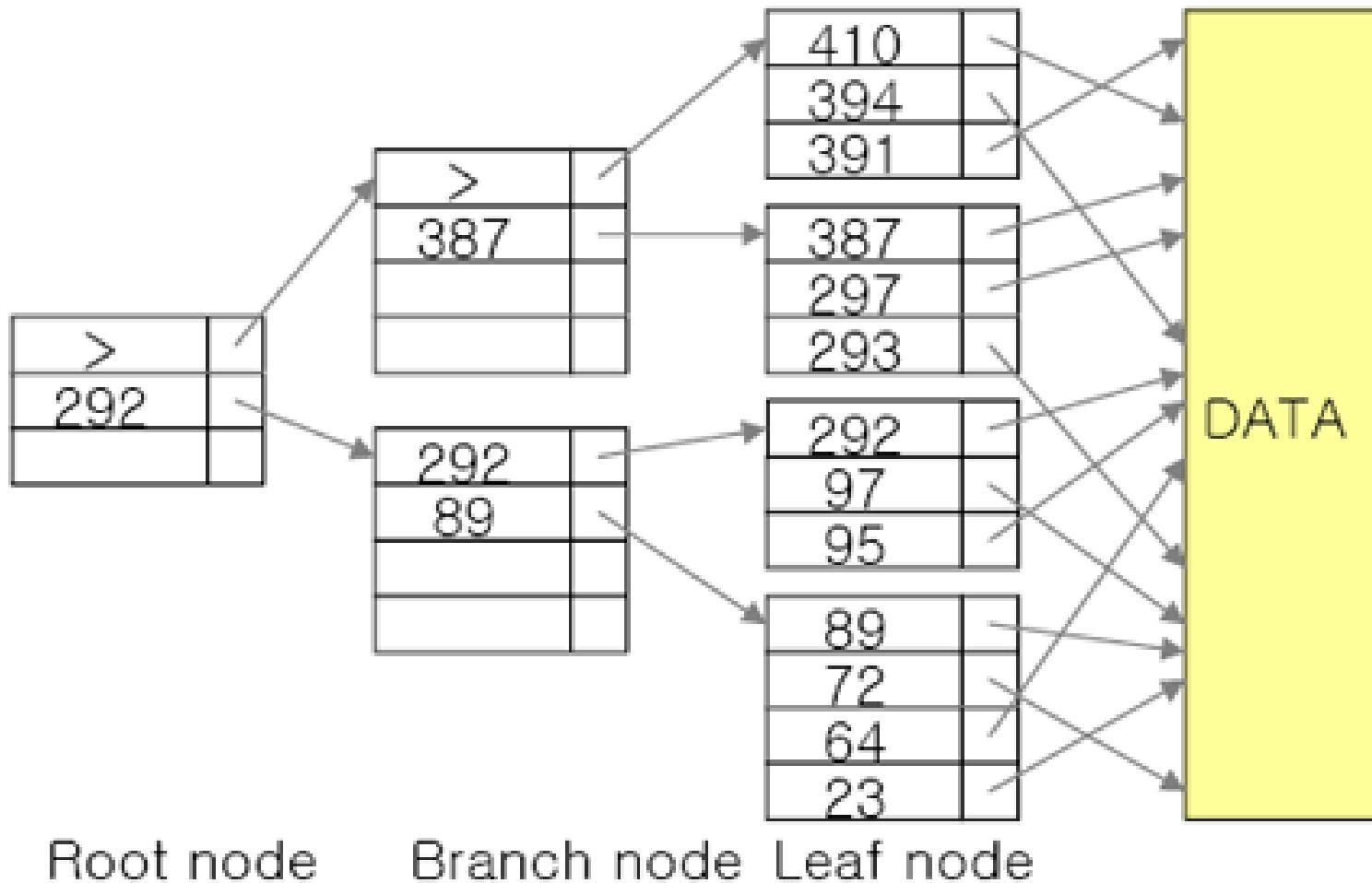
En general las funciones deben ser funciones definidas por un usuario.

Reverse Key Index (Oracle)

Invierte los bytes de la clave a indexar. Esto sirve para los índices cuyas claves son una serie constante con por ej. Crecimiento ascendente. para que las inserciones se distribuyan por todas las hojas del árbol de índice.

Indices (Cont.)

Indices B Tree



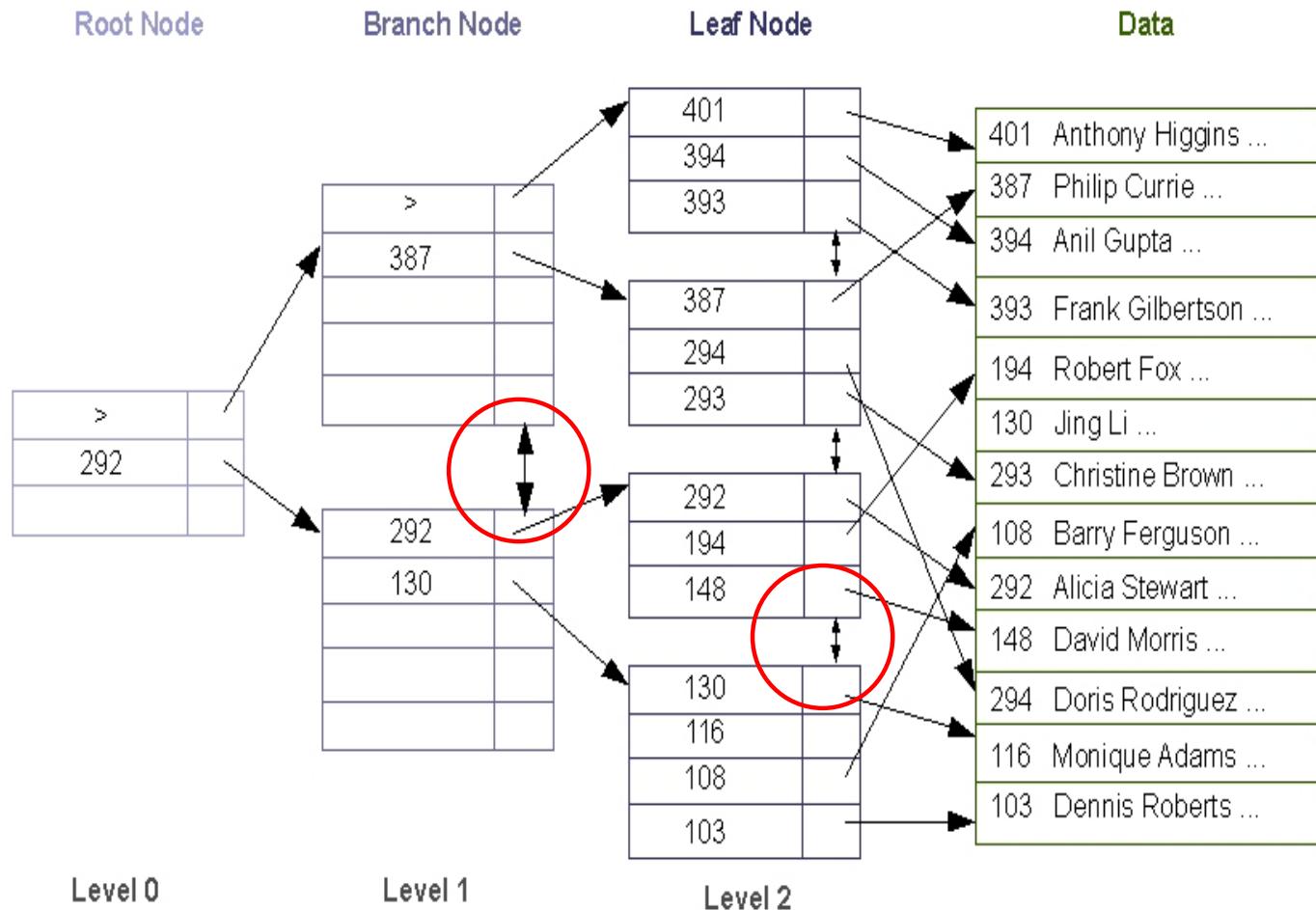
Root node

Branch node

Leaf node

Indices (Cont.)

Indices B Tree +



Indices (Cont.)

CARACTERÍSTICAS DIFERENCIADORAS PARA LOS ÍNDICES

Unique Índice de clave única. Sólo admite una fila por clave.

Duplicado Permite múltiples filas para una misma clave.

Simple La clave está integrada por una sola columna.

Compuesto La clave se compone de varias columnas.

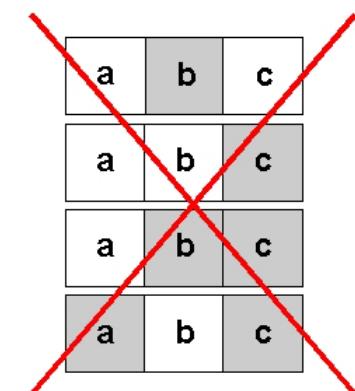
Las principales funciones de un índice compuesto son:

Facilitar múltiples joins entre columnas

Incrementar la unicidad del valor de los índices

```
CREATE INDEX ix_sample  
ON sample_table (a, b, c);
```

a	b	c
a	b	c
a	b	c



Indices (Cont.)

BENEFICIOS DE LA UTILIZACIÓN DE INDICES:

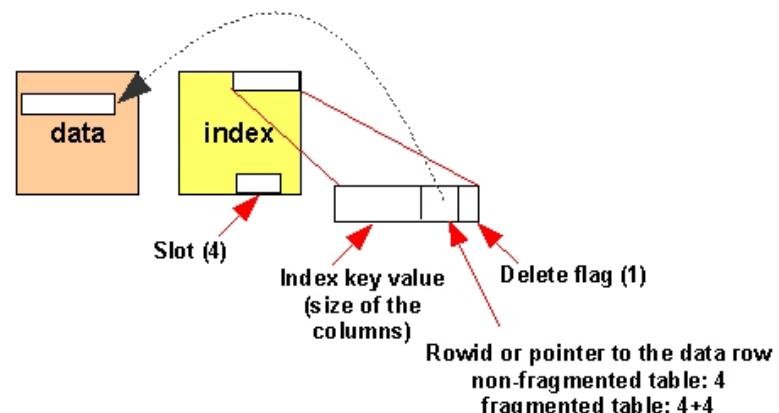
- Se le provee al sistema mejor performance al equipo ya que no debe hacer lecturas secuenciales sino accede a través de los índices, sólo en los casos que las columnas del Select no formen parte del índice.
- Mejor performance en el ordenamiento de filas
- Asegura únicos valores para las filas almacenadas
- Cuando las columnas que intervienen en un JOIN tienen índices se le da mejor performance si el sistema logra recuperar los datos a través de ellas
- Asegura el cumplimiento de constraints y reglas de negocio.
 - Primary key, foreign keys, unique values

Indices (Cont.)

COSTO DE LA UTILIZACIÓN DE INDICES

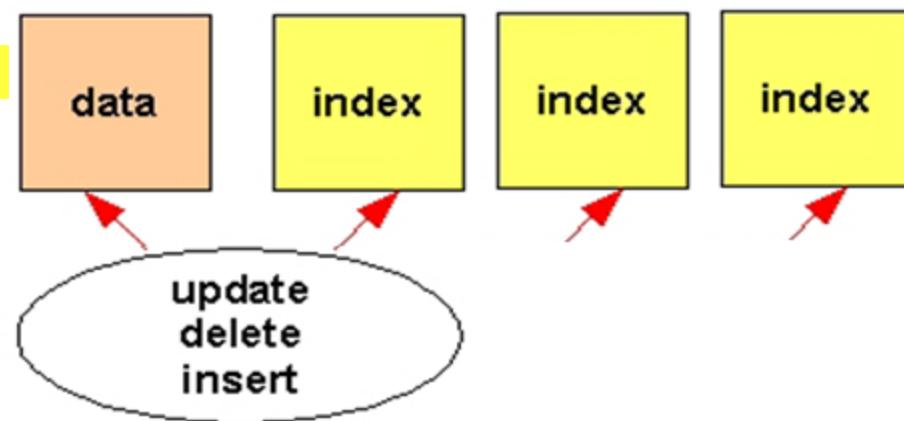
COSTO DE ESPACIO DE DISCO

El primer costo asociado es el espacio que ocupa en disco, que en algunos casos suele ser mayor al que ocupan los datos.



COSTO DE PROCESAMIENTO Y MANTENIMIENTO

El segundo costo es el de procesamiento, hay que tener en cuenta que cada vez que una fila es insertada o modificada o borrada, el índice debe estar bloqueado, con lo cual el sistema deberá recorrer y actualizar los distintos índices.



Indices (Cont.)

GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR

- Indexar columnas que intervienen en Joins
- Indexar las columnas donde frecuentemente se realizan filtros
- Indexar columnas que son frecuentemente usadas en orders by
- Evitar duplicación de índices
 - Sobre todo en columnas con pocos valores diferentes Ej: Sexo, Estado Civil, Etc.
- Verificar que el tamaño de índice debería ser pequeño comparado con la fila
 - Tratar sobre todo en crear índices sobre columnas cuya longitud de atributo sea pequeña
 - No crear índices sobre tablas con poca cantidad de filas, no olvidar que siempre se recupera de a páginas. De esta manera evitaremos que el sistema lea el árbol de índices

(????)
NO ENTENDI

Indices (Cont.)

GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR (Cont.)

- Limitar la cantidad de índices en tablas que son actualizadas frecuentemente
 - Porque sobre estas tablas se estarán ejecutando Selects extras
- Tratar de usar índices compuestos para incrementar los valores únicos
 - Tener en cuenta que si una o más columnas intervienen en un índice compuesto el optimizador podría decidir acceder a través de ese índice aunque sea solo para la búsqueda de los datos de una columna, esto se denomina "*partial key search*"
- Usando **cluster index** se agiliza la recuperación de filas
 - Uno de los principales objetivos de la Optimización de bases de datos es reducir la entrada/salida de disco. Reorganizando aquellas tablas que lo necesiten, se obtendría como resultado que las filas serían almacenadas en bloques contiguos, con lo cual facilitaría el acceso y reduciría la cantidad de accesos ya que recuperaría en menos páginas los mismos datos.

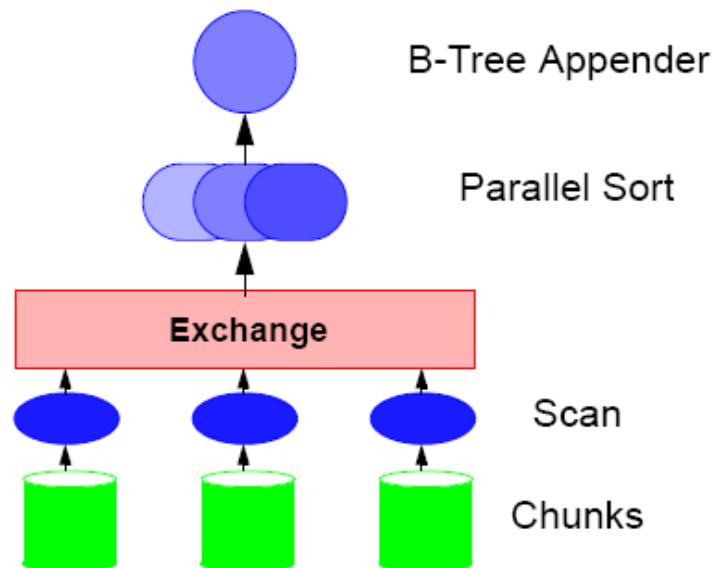
Indices (Cont.)

CONSTRUCCIÓN DE ÍNDICES EN PARALELO

Los motores de BD usan en general métodos de construcción de índices en paralelo.

El arbol B+ es construido por 2 o más procesos paralelos. Para esto el motor realiza una muestra de la filas a Indexar (aproximadamente 1000) y luego decide como separar en grupos. Luego scanea las filas y las ordena usando el mecanismo de sort en paralelo.

Las claves ordenadas son colocadas en los grupos apropiados para luego ir armado en paralelo un subárbol por cada grupo. Al finalizar los subárboles se unen en un único Arbol B+.



Indices (Cont.)

Motor SQL SERVER

Sql Server utiliza una estructura de Arbol B+.

Máxima cantidad de campos para la clave de un índice compuesto: 16.

Creación de un índice único y simple:

CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);

Creación de Indice duplicado y compuesto.

CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);

Creación de Indice clustered

CREATE CLUSTERED INDEX ix3_ordenes ON ordenes (N_orden);

Indices (Cont.)

Motor SQL SERVER (cont.)

Manejo del Load Factor

FILLFACTOR – Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.

Por ej. Si el FILLFACTOR=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

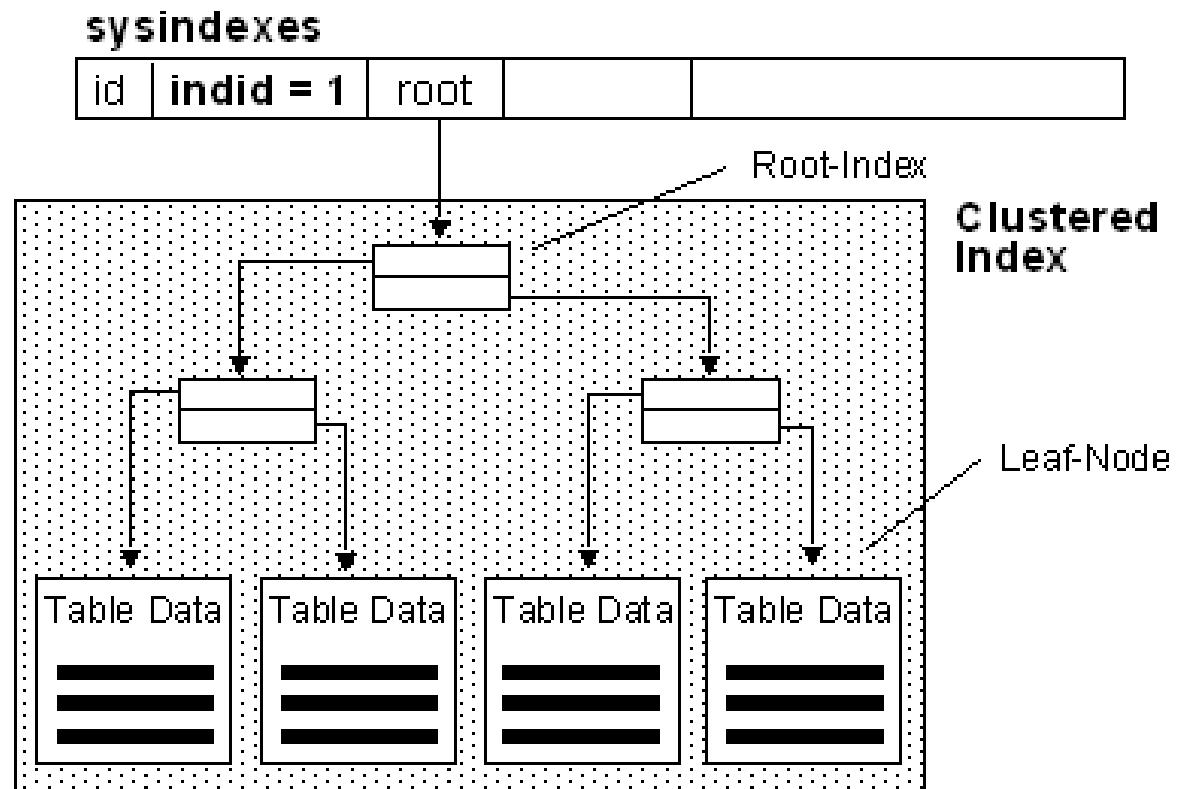
(???)
NO ENTENDI

```
CREATE UNIQUE INDEX ix1_ordenes ON ordenes(N_order)  
WITH FILLFACTOR = 20;
```

Indices (Cont.)

MOTOR SQL SERVER

Indices Clustered



Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

Una IOT tiene una organización del almacenamiento que es una variante del Árbol-B. A diferencia de una tabla común, en la que los datos se guardan como un conjunto desordenado, en una IOT se guardan en una estructura de índice de Árbol-B, ordenado a la manera de una clave primaria. Sin embargo, además de almacenar los valores de las columnas de clave primaria de cada tabla, cada entrada en el índice almacena además los valores de las columnas no clave.

De esta forma, en vez de mantener dos estructuras separadas de almacenamiento, una para la tabla y una para el índice de Árbol-B, el sistema mantiene sólo un índice Árbol-B, porque además de almacenar el rowid de las filas, también se guardan las columnas no clave.

Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

Motor Oracle

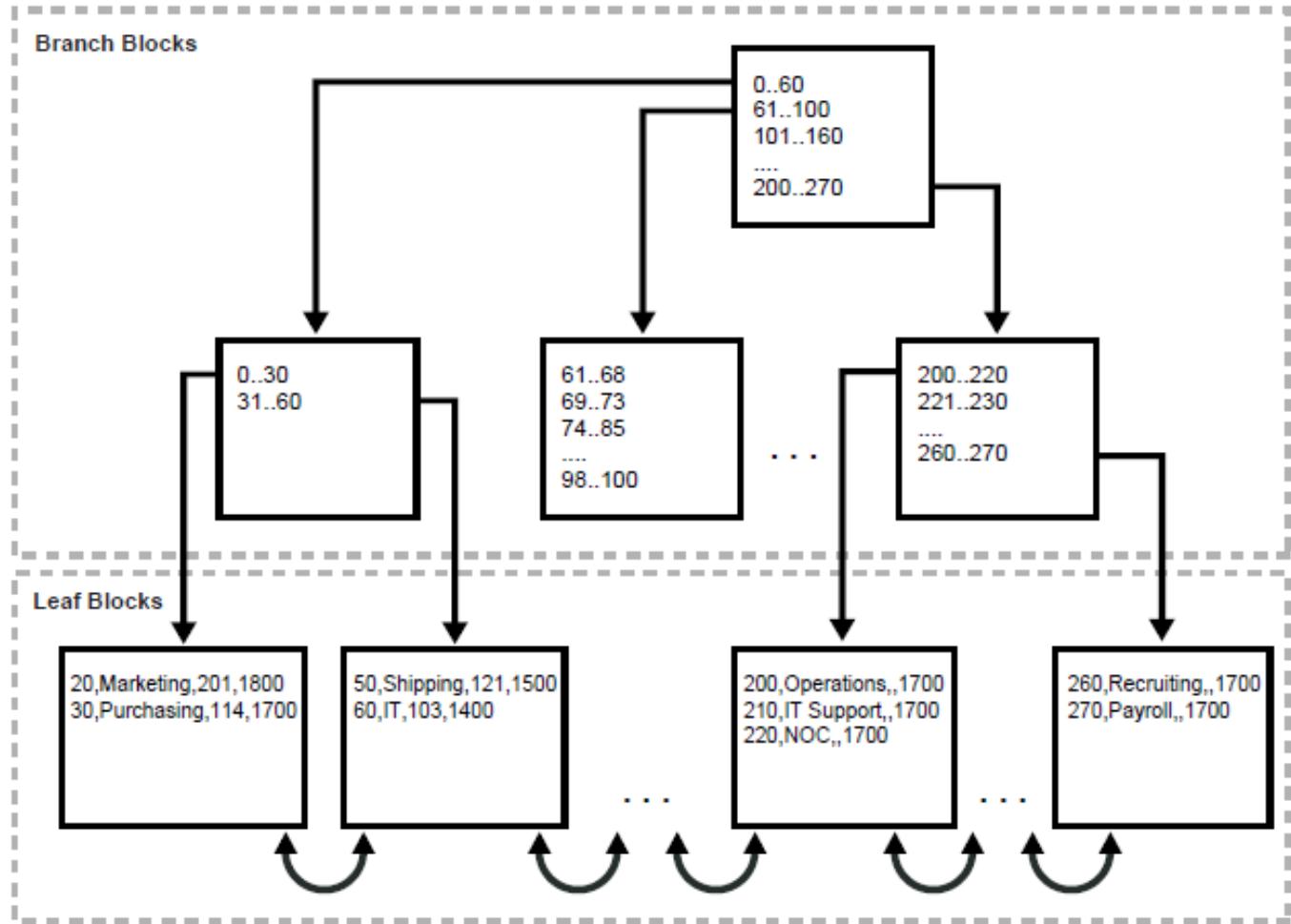
La siguiente sentencia crea la tabla ordenes, como una tabla organizada por índice, indicando que la columna N_cliente divide el área del índice, de las columnas no clave.

```
CREATE TABLE ordenes
( N_orden NUMBER PRIMARY KEY,
  N_cliente NUMBER,
  F_orden DATE,
  C_estado NUMBER,
  F_alta_audit DATE,
  D_usuario VARCHAR2(20) )
```

ORGANIZATION INDEX INCLUDING N_CLIENTE OVERFLOW

Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Ora)

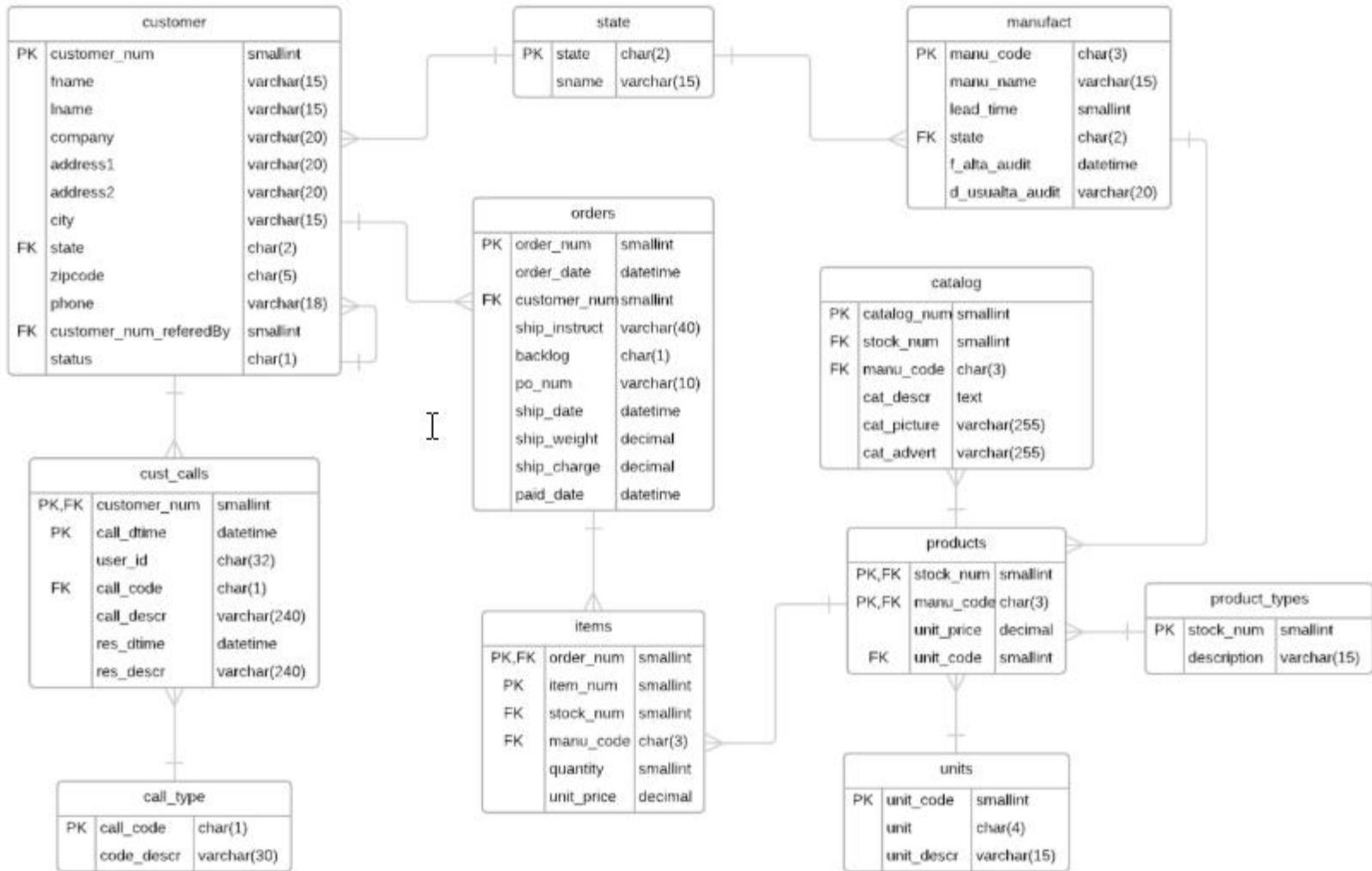
Motor Oracle



SQL – Hoy Veremos

- **DDL** – Data Definition Language
 - INDICES
- **DML** – Data Manipulation Language
 - SUBQUERIES en
 - SELECT
 - DELETE
 - UPDATE
 - INSERT
 - Operadores Multi Select
 - UNION
 - INTERSECT
 - EXCEPT

Base de Datos de Ejemplo



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 three tabs: 'SQLQuery2.sql - D...HQEMP6\zaffa (53)', 'Object Explorer Details', and 'SQLQuery1.sql - D...HQEMP6\zaffa (54)*'. The 'SQLQuery1.sql' tab contains the following code:

```
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 status bar at the bottom left shows '100 %' and 'Messages'. The message pane shows '(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.

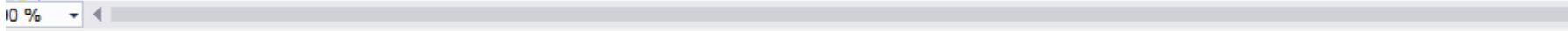
SQL – SubQuery en **DELETE**

```
DELETE FROM customer
WHERE customer_num NOT IN
  (SELECT DISTINCT customer_num FROM cust_calls) 1
AND customer_num NOT IN
  (SELECT DISTINCT customer_num FROM orders) 2
AND customer_num NOT IN
  (SELECT DISTINCT customer_num_referredBy FROM customer c2
   WHERE customer_num_referredBy IS NOT NULL) 3
```

Utilizando tres subqueries como condiciones del DELETE, podríamos borrar todos los clientes de la Tabla customer, que no posean ordenes de compra asociadas y que no hayan referenciado a otro cliente y que no tengan llamados telefónicos.

SQL – SubQuery en **DELETE**

```
SELECT * INTO #clientesParaBorrar FROM customer
DELETE FROM #clientesParaBorrar
WHERE customer_num NOT IN
    (SELECT DISTINCT customer_num FROM cust_calls)
    AND customer_num NOT IN
    (SELECT DISTINCT customer_num FROM orders)
    AND customer_num NOT IN
    (SELECT DISTINCT customer_num_referredBy FROM customer c2
    WHERE customer_num_referredBy IS NOT NULL)
```

10 % 
Messages

(8 row(s) affected)

Para hacer la prueba y no borrar datos de nuestra tabla real
corrimos el ejemplo en la tabla temporal creada
#clientesParaBorrar

SQL – SubQuery en **DELETE**

```
SELECT customer_num FROM customer  
EXCEPT  
SELECT customer_num FROM #clientesParaBorrar
```

VEREMOS ESTE OPERADOR LUEGO

The screenshot shows a SQL query window with two SELECT statements separated by an EXCEPT operator. The first statement selects customer numbers from the 'customer' table, and the second statement selects them from a temporary table '#clientesParaBorrar'. The results are displayed in a table with one column labeled 'customer_num'. The values listed are 102, 105, 108, 109, 113, 114, 118, and 128.

customer_num
102
105
108
109
113
114
118
128

En este ejemplo hacemos la comparación entre las tablas para ver que filas se borraron.

SQL – SubQuery en **UPDATE**

UPDATE FROM customer

SET columna = Subquery

WHERE columna (= / IN / NOT IN) Subquery

(EXISTS / NOT EXISTS) Subquery

En un update vemos distintos lugares y formas de ejecutar un subquery.

SQL – SubQuery en UPDATE

Suquery en CLAUSULA SET

```
UPDATE #clientesParaBorrar  
SET state = (SELECT state FROM state WHERE sname='Florida')  
WHERE customer_num=101
```

(???)
NO HACE FALTA
UN DISTINCT?

Subquery

(1 row(s) affected)

```
SELECT * FROM #clientesParaBorrar  
WHERE customer_num=101
```

Results Messages

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	FL	94086	408-789-8075	NULL	NULL

El subquery puede retornar solo una única columna y fila. O sea un valor escalar.

SQL – SubQuery en UPDATE

Suquery en **CLAUSULA WHERE** con Subquery que devuelve valor escalar

(???)
NO HACE FALTA
UN DISTINCT?

```
UPDATE #clientesParaBorrar
SET state = 'CA'
WHERE customer_num=(SELECT customer_num FROM customer
                      WHERE fname='Ludwig' AND lname='Pauli')
```

Subquery

(1 row(s) affected)

```
SELECT * FROM #clientesParaBorrar
WHERE customer_num=101
```

Results											
customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075	NULL	NULL

El subquery puede retornar solo una única columna y fila. O sea un valor escalar.

Por qué?

SQL – SubQuery en UPDATE

Suquery en **CLAUSULA WHERE** con Subquery que **devuelve una columna con multiples filas.**

The screenshot shows an SQL query window in SSMS. The main query is:

```
UPDATE manufact SET lead_time=15  
WHERE manu_code IN (SELECT DISTINCT manu_code FROM items)
```

A red arrow points from the text "Subquery" to the subquery part of the WHERE clause. Below the main query, there is a --COMPROBACION section with a SELECT query:

```
--COMPROBACION  
SELECT DISTINCT i.manu_code,lead_time  
FROM items i INNER JOIN manufact m ON (i.manu_code=m.manu_code)
```

The results grid shows the following data:

	manu_code	lead_time
1	ANZ	15
2	HRO	15
3	HSK	15
4	KAR	15
5	NKL	15
6	NRG	15
7	PRC	15
8	SHM	15
9	SMT	15

Modificamos el lead_time (tiempo de entrega) a 15 días para todos los proveedores de los cuales ya hayamos entregado productos.

SQL – SubQuery en UPDATE

Que sucede si corremos este UPDATE?

UPDATE manufact SET lead_time=15

WHERE manu_code = (SELECT DISTINCT manu_code FROM items)



```
UPDATE manufact SET lead_time=15
WHERE manu_code = (SELECT DISTINCT manu_code FROM items)
```

A screenshot of a SQL query editor window. The main pane shows the SQL code. Below it is a 'Messages' pane displaying an error message:

100 %

Msg 512, Level 16, State 1, Line 10
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression.
The statement has been terminated.

Break



SQL – SubQuery en **SELECT**

SELECT col, col, **Subquery**

FROM table **JOIN** **Subquery**

WHERE columna (**=/≤/≥/IN/NOT IN**) **Subquery**
(EXISTS / NOT EXISTS) **Subquery**

GROUP BY ...

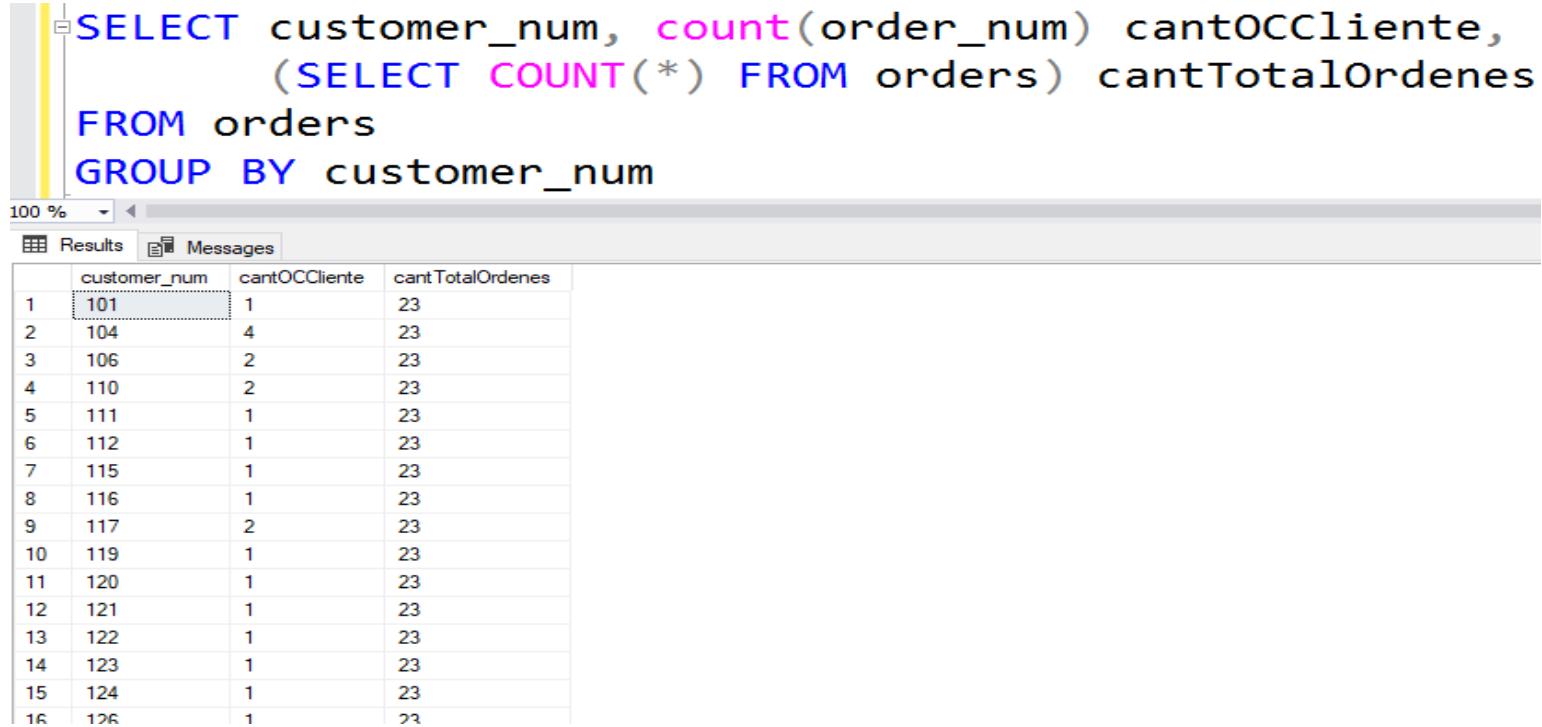
HAVING fxAgregada (**=/≤/≥/IN/NOT IN**) **Subquery**
(EXISTS / NOT EXISTS) **Subquery**

ORDER BY **Subquery**

En un **SELECT** vemos distintos lugares y formas de ejecutar un subquery.

SQL – SubQuery en **SELECT**

Subquery en Lista de columnas.



The screenshot shows a SQL query being run in SSMS. The query is:

```
SELECT customer_num, count(order_num) cantOCCliente,
       (SELECT COUNT(*) FROM orders) cantTotalOrdenes
  FROM orders
 GROUP BY customer_num
```

The results grid displays the following data:

	customer_num	cantOCCliente	cantTotalOrdenes
1	101	1	23
2	104	4	23
3	106	2	23
4	110	2	23
5	111	1	23
6	112	1	23
7	115	1	23
8	116	1	23
9	117	2	23
10	119	1	23
11	120	1	23
12	121	1	23
13	122	1	23
14	123	1	23
15	124	1	23
16	126	1	23

El subquery devuelve la cantidad de ordenes que existen y como si fuese una variable lo muestra en cada fila

SQL – SubQuery en SELECT

Subquery en el **FROM**.

```
SELECT lname apellido, fname nombre, cliente, cantidad
FROM customer c1 JOIN
(SELECT customer_num cliente, count(order_num) cantidad
FROM orders
GROUP BY customer_num) c2
on (c1.customer_num = c2.cliente)
WHERE cantidad > 3
```

Results Messages

	apellido	nombre	cliente	cantidad
1	Higgins	Anthony	104	4

El subquery devuelve un conjunto de filas se asocian al alias c2 como si fuese el result set de una tabla y luego estos datos son JOINeados con la tabla customer c1.

SQL – SubQuery en **SELECT**

Subquery en el **WHERE**.

The screenshot shows a SQL query in the 'Results' tab of SSMS. The query is:

```
SELECT lname+', '+fname, customer_num  
FROM customer  
WHERE customer_num IN (SELECT customer_num FROM cust_calls  
GROUP BY customer_num HAVING COUNT(*)>1)
```

A red box highlights the subquery part of the WHERE clause. The results table below shows one row:

	(No column name)	customer_num
1	Parmelee, Jean	116

La consulta muestra los **clientes que posean mas de una llamada**. Muchas veces este tipo de subqueries **se pueden resolver mediante joins**.

Resuelvanlo en 5 minutos!!

SQL – SubQuery en **SELECT**

Subquery en el **WHERE** con múltiples filas.

```
SELECT lname+', '+fname, customer_num  
FROM customer  
WHERE customer_num IN (SELECT customer_num FROM cust_calls  
GROUP BY customer_num HAVING COUNT(*)>1)
```

Results

(No column name)	customer_num
1	Parmelee, Jean

La consulta muestra los clientes que posean mas de una llamada. Muchas veces este tipo de subqueries se pueden resolver mediante joins.

Resuelvanlo en 5 minutos!!

SQL – SubQuery en **SELECT**

Subquery en el **WHERE** con un valor escalar.

The screenshot shows a SQL query in the 'Query Editor' window of SQL Server Management Studio. The query is:

```
SELECT COUNT(*) FROM customer
WHERE city = (SELECT city FROM customer WHERE lname = 'Higgins')
```

A red box highlights the subquery part of the WHERE clause: `(SELECT city FROM customer WHERE lname = 'Higgins')`. A red arrow points from the text 'Subquery en el WHERE con un valor escalar.' above to this highlighted subquery.

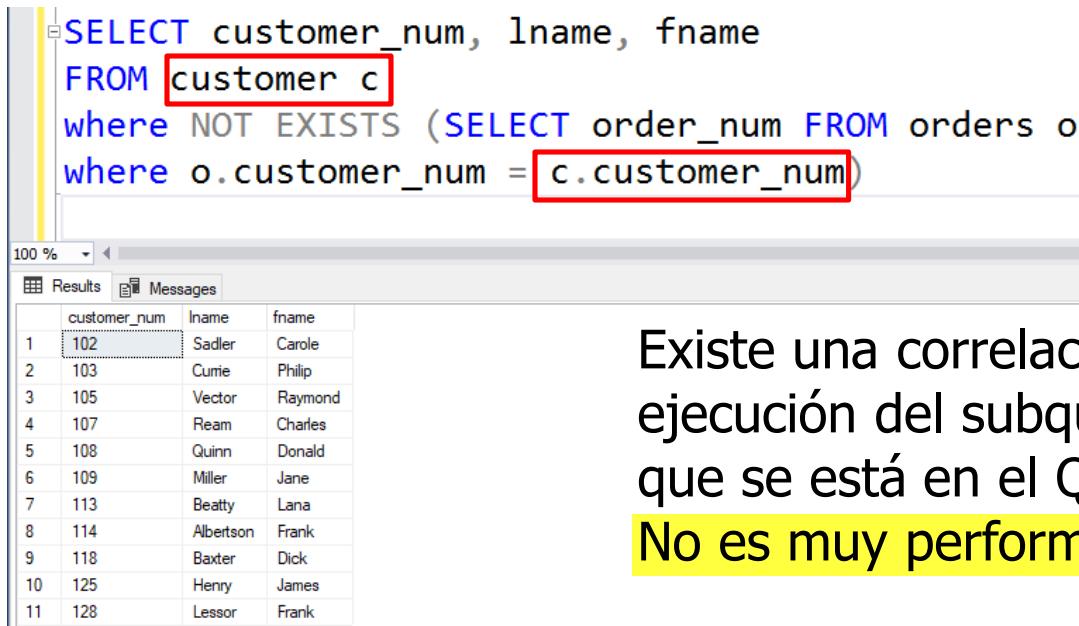
The results pane shows a single row with the value 5.

La consulta devuelve la cantidad de clientes que viven en la misma ciudad en la que vive Higgins, incluso el mismo.
Se puede resolver con joins.

Resuelvanlo en 5 minutos!!

SQL – SubQuery en **SELECT**

Subquery CORRELACIONADO.



```
SELECT customer_num, lname, fname
FROM customer c
WHERE NOT EXISTS (SELECT order_num FROM orders o
WHERE o.customer_num = c.customer_num)
```

customer_num	lname	fname
102	Sadler	Carole
103	Currie	Philip
105	Vector	Raymond
107	Ream	Charles
108	Quinn	Donald
109	Miller	Jane
113	Beatty	Lana
114	Albertson	Frank
118	Baxter	Dick
125	Henry	James
128	Lessor	Frank

Existe una correlación entre la ejecución del subquery y la fila en la que se está en el Query original.
No es muy performante!!

La consulta devuelve los clientes que para los que no existen ordenes de compra.

SQL – SubQuery en SELECT

Subquery CORRELACIONADO.

```
SELECT customer_num, lname, fname  
FROM customer c  
where NOT EXISTS (SELECT order_num FROM orders o  
where o.customer_num = c.customer_num)
```

Results

	customer_num	lname	fname
1	102	Sadler	Carole
2	103	Cumie	Philip
3	105	Vector	Raymond
4	107	Rream	Charles
5	108	Quinn	Donald
6	109	Miller	Jane
7	113	Beatty	Lana
8	114	Albertson	Frank
9	118	Baxter	Dick
10	125	Henry	James
11	128	Lessor	Frank

```
SELECT c.customer_num, lname, fname  
FROM customer c LEFT JOIN orders o  
ON c.customer_num=o.customer_num  
GROUP BY c.customer_num, lname, fname  
HAVING COUNT(order_num)=0  
ORDER BY 1
```

Results

	customer_num	lname	fname
1	102	Sadler	Carole
2	103	Cumie	Philip
3	105	Vector	Raymond
4	107	Rream	Charles
5	108	Quinn	Donald
6	109	Miller	Jane
7	113	Beatty	Lana
8	114	Albertson	Frank
9	118	Baxter	Dick
10	125	Henry	James
11	128	Lessor	Frank

Misma consulta con
Joins y GROUP BY

SQL – SubQuery en SELECT

Subquery CORRELACIONADO.

```
SELECT customer_num, lname, fname  
FROM customer c  
where NOT EXISTS (SELECT order_num FROM orders o  
where o.customer_num = c.customer_num)
```

(???)
LA QUERY DE ABAJO SIGUE
SIENDO POCO PERFORMANTE???

	customer_num	lname	fname
1	102	Sadler	Carole
2	103	Cumie	Philip
3	105	Vector	Raymond
4	107	Ream	Charles
5	108	Quinn	Donald
6	109	Miller	Jane
7	113	Beatty	Lana
8	114	Albertson	Frank
9	118	Baxter	Dick
10	125	Henry	James
11	128	Lessor	Frank

```
SELECT customer_num, lname, fname  
FROM customer  
WHERE customer_num NOT IN  
(SELECT DISTINCT customer_num FROM orders)
```

	customer_num	lname	fname
1	102	Sadler	Carole
2	103	Cumie	Philip
3	105	Vector	Raymond
4	107	Ream	Charles
5	108	Quinn	Donald
6	109	Miller	Jane
7	113	Beatty	Lana
8	114	Albertson	Frank
9	118	Baxter	Dick
10	125	Henry	James
11	128	Lessor	Frank

Misma consulta con
Subquery común

SQL – SubQuery en **SELECT**

listar las ordenes de los clientes con su total comprado pero sólo para los que el total de la orden sea mayor que el **promedio comprado** de dicho cliente.

```
SQLQuery1.sql - D:\HQEMP6\zaffa (53)* ↵ X
SELECT o.order_num, o.customer_num, SUM(quantity*unit_price) totalOrden
, promedio
FROM orders o JOIN items i ON o.order_num=i.order_num
JOIN (SELECT customer_num, SUM(quantity*unit_price)
      /COUNT(DISTINCT o2.order_num) promedio
      FROM orders o2 JOIN items i2 ON (o2.order_num=i2.order_num)
      GROUP BY customer_num) tabSQ
      ON (o.customer_num=tabSQ.customer_num)
      GROUP BY o.order_num, o.customer_num, tabSQ.promedio
      HAVING SUM(i.quantity*i.unit_price) > tabSQ.promedio
```

100 %

Results Messages

	order_num	customer_num	totalOrden	promedio
1	1003	104	1355.00	570.950000
2	1008	110	1340.00	895.000000
3	1014	106	1440.00	1428.000000
4	1012	117	2840.00	2268.000000

Resolución con **subquery en FROM**

SQL – Hoy Veremos

- **DDL** – Data Definition Language
 - INDICES
- **DML** – Data Manipulation Language
 - SUBQUERIES en
 - SELECT
 - DELETE
 - UPDATE
 - INSERT
 - Operadores Multi Select
 - UNION
 - INTERSECT
 - EXCEPT

Operador Union

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
UNION
```

```
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

La cantidad de campos en cada consulta debe corresponderse.

Los campos en igual posición deben tener el mismo tipo de datos.

La clausula ORDER BY va al final del ultimo SELECT.

Solo se pude Ordenar referenciando por posición.

En el caso de filas con idénticos valores solo deja una de las dos.

Operador Union

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25
```

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
UNION  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

	stock_num	manu_code
1	5	ANZ
2	5	NRG

```
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5
```

	stock_num	manu_code
1	5	NRG
2	5	SMT
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	NRG
7	5	ANZ
8	5	ANZ
9	5	SMT

Operador Union

```
SELECT stock_num PROD,manu_code FAB FROM products  
WHERE unit_price < 25
```

UNION

```
SELECT stock_num,manu_code FROM items WHERE stock_num=5  
ORDER BY 1,2
```

```
SELECT stock_num PROD,manu_code FAB FROM products  
WHERE unit_price < 25  
UNION  
SELECT stock_num,manu_code FROM items WHERE stock_num=5  
ORDER BY 1,2
```

PROD	FAB
5	ANZ
5	NRG
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

La tabla de salida toma los nombres de columnas del primer SELECT.

Operador Union – Mas de dos tablas

```
SELECT 1 orden, customer_num, order_num, order_date  
FROM orders WHERE customer_num=127  
UNION  
SELECT 3 orden, customer_num, order_num, order_date  
FROM orders WHERE customer_num BETWEEN 111 AND 126  
UNION  
SELECT 2 orden, customer_num, order_num, order_date  
FROM orders WHERE customer_num BETWEEN 1 AND 110  
ORDER BY 1 ASC, 2 DESC
```

En este caso particular vemos que utilizamos un ordenamiento de filas determinado por una constante en cada select que dice que filas irán primero en la salida.

Operador Union – Mas de dos tablas

```
SQLQuery1.sql - D...HQEMP6\zaffa (53)*  ▶ X  
UNION  
SELECT 2 orden, customer_num, order_num, order_date  
FROM orders WHERE customer_num BETWEEN 1 AND 110  
ORDER BY 1 ASC, 2 DESC
```

100 %

Results Messages

	orden	customer_num	order_num	order_date
1	1	127	1023	2015-07-20 00:00:00.000
2	2	110	1008	2015-06-03 00:00:00.000
3	2	110	1015	2015-06-23 00:00:00.000
4	2	106	1004	2015-05-18 00:00:00.000
5	2	106	1014	2015-06-21 00:00:00.000
6	2	104	1001	2015-05-16 00:00:00.000
7	2	104	1003	2015-05-18 00:00:00.000
8	2	104	1011	2015-06-14 00:00:00.000
9	2	104	1013	2015-06-18 00:00:00.000
10	2	101	1002	2015-05-17 00:00:00.000
11	3	126	1022	2015-07-20 00:00:00.000
12	3	124	1021	2015-07-19 00:00:00.000
13	3	123	1020	2015-07-07 00:00:00.000
14	3	122	1019	2015-07-07 00:00:00.000
15	3	121	1018	2015-07-06 00:00:00.000
16	3	120	1017	2015-07-05 00:00:00.000
17	3	119	1016	2015-06-25 00:00:00.000
18	3	117	1007	2015-05-27 00:00:00.000

Ordenamiento particular, primero el 127, luego del 110 al 1, y finalmente del 128 al 111

Operador UNION ALL

```
SELECT stock_num,manu_code FROM products WHERE unit_price < 25  
UNION ALL
```

```
SELECT stock_num,manu_code FROM items WHERE stock_num=5  
ORDER BY 1,2
```

SELECT stock_num,manu_code FROM products WHERE unit_price < 25 UNION ALL SELECT stock_num,manu_code FROM items WHERE stock_num=5 ORDER BY 1,2	
stock_num	manu_code
5	ANZ
5	NRG
5	NRG
5	SMT
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

El Operador UNION con la cláusula ALL, repite en la estructura de salida todas las filas de las tablas involucradas, sean o no iguales.

Operador INTERSECT

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
INTERSECT
```

```
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

Mismas restricciones del UNION.

Devuelve las filas que están en ambas consultas.

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
INTERSECT  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```



The screenshot shows a database interface with a query editor and a results viewer. The query editor contains the code from the previous block. The results viewer shows a single row with two columns: 'stock_num' and 'manu_code'. The value '5' is in the 'stock_num' column and 'ANZ' is in the 'manu_code' column.

stock_num	manu_code
5	ANZ

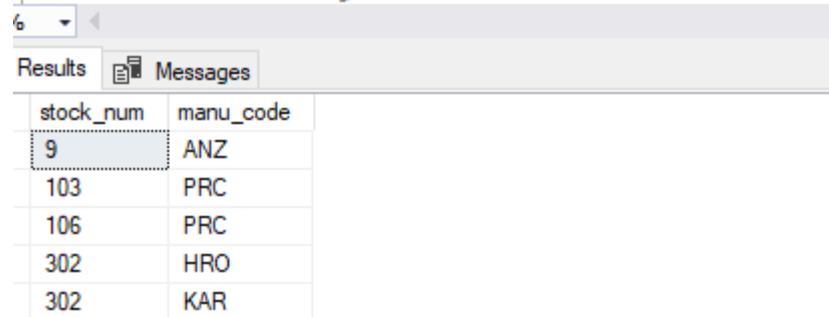
Operador EXCEPT

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
EXCEPT  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

Mismas restricciones del UNION.

Devuelve las filas del Primer SELECT que no están en la intersección con el Segundo SELECT.

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
EXCEPT  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```



The screenshot shows a SQL query results window with two tabs: 'Results' and 'Messages'. The 'Results' tab displays a table with two columns: 'stock_num' and 'manu_code'. The data is as follows:

stock_num	manu_code
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

Operador Union

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25
```

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

UNION

UNION ALL

INTERSECT

EXCEPT

	stock_num	manu_code
	5	ANZ
	5	NRG
	5	SMT
	9	ANZ
	103	PRC
	106	PRC
	302	HRO
	302	KAR

	stock_num	manu_code
	5	ANZ
	5	NRG
	5	NRG
	5	SMT
	5	SMT
	9	ANZ
	103	PRC
	106	PRC
	302	HRO
	302	KAR

	stock_num	manu_code
	5	ANZ

	stock_num	manu_code
	9	ANZ
	103	PRC
	106	PRC
	302	HRO
	302	KAR

Vamos a la Practica!!!

