

# SQL Server 2008 R2

## Clase 3

### MANIPULACIÓN DE DATOS

o Datos Numéricos: Operadores Aritméticos

o Conversión de datos

### FILTRANDO DATOS A TRAVES DE CONDICIONES DE BÚSQUEDA

o Cláusula WHERE

o Cláusula BETWEEN

o Cláusula IN

o Cláusula LIKE

o Valores NULL

o Operadores Lógicos AND y OR

o Cláusula DISTINCT

o Cláusula ORDER

### RELACIONANDO Y/O RECUPERARANDO INFORMACIÓN RELACIONANDO DOS O MAS TABLAS

o JOINS

o INNER JOINS

o OUTER JOINS

o LEFT OUTER JOINS

o RIGHT OUTER JOINS

o FULL JOINS

o CROSS JOINS

o JOINS con más de dos tablas

o SELF JOINS

# SQL Server 2008 R2

## Clase 3... antes un repaso

**SELECT** es el comando utilizado para recuperar datos de una o más {tablas y/o vistas}

Tabla Clientes

Nombre de columna	Tipo de datos	Permitir valores NULL
IdCliente	int	<input type="checkbox"/>
ApellidoRzSocial	varchar(255)	<input type="checkbox"/>
Nombre	varchar(255)	<input checked="" type="checkbox"/>
FechaAlta	date	<input type="checkbox"/>
FechaBaja	date	<input checked="" type="checkbox"/>
Telefono	varchar(255)	<input checked="" type="checkbox"/>
Calle	varchar(255)	<input checked="" type="checkbox"/>
CodigoPostal	varchar(10)	<input checked="" type="checkbox"/>
IdProvincia	char(3)	<input checked="" type="checkbox"/>

La estructura básica de un SELECT tiene los siguientes elementos:

**SELECT** [\* | enumeración de columnas separadas por coma]  
**FROM** nombreDeTabla

**SELECT** \*  
**FROM** Clientes /\* recupera todos los datos de la tabla con su nombre de columna original\*/

**SELECT** ApellidoRzSocial      **AS** 'Apellido o Razon Social',  
IdCliente                      **AS** [Numero de Cliente],  
FechaBaja                      'Fecha de Baja',  
'Valor fijo'                      Literal  
**FROM** Clientes

- Puedo crear alias con la palabra reservada AS (no es obligatorio)
- Si uso alias con palabras que contienen espacios debo usar [] o ''

Importante:

Siempre se debe buscar de usar una indentación clara  
Para favorecer la lectura de la consulta

# SQL Server 2008 R2

Datos numéricos exactos:

<b>Numeric</b> De- $10^{38} + 1$ a $10^{38} - 1$ Numeric (precision,escala) NUMERIC(5,2) → 5 digitos como maximo de los cuales 2 reservo para decimal	<b>Int</b> De $-2^{31}$ a $2^{31}-1$ Entero simple con signo
<b>Bit</b> [1 0 true  false]	Bigint De $-2^{63}$ a $2^{63}-1$
Decimal idem <b>Numeric</b>	Smallint De $-2^{15}$ a $2^{15}-1$
Money y Smallmoney tienen una precisión de una diezmilésima de las unidades monetarias que representan. Por ejemplo, 2.15 puede especificar 2 dólares y 15 centavos.	
Tinyint De 0 a 255 entero sin signo	

Datos numéricos inexactos:

**Float y Real:** Se utilizan con datos numéricos de coma flotante. No se pueden representar todos los valores del rango con exactitud. Ej float(53) donde 53 es la mantisa del numero en notación científica.

# SQL Server 2008 R2

Operadores Aritméticos:

+ operador suma

- operador resta

\* operador multiplicación

/ operador división

% operador resto

```
SELECT 20 + 5 SUMA,  
       20 - 5 RESTA,  
       20 * 5 MULTIPLICACION,  
       20 / 5 DIVISION,  
       20 % 5 RESTO
```



	SUMA	RESTA	MULTIPLICACION	DIVISION	RESTO
1	25	15	100	4	0

```
SELECT 20.538 + 5 SUMA,  
       20.538 - 5 RESTA,  
       20 * 5.00 MULTIPLICACION,  
       20.538 / 5 DIVISION,  
       20.538 % 5.00 RESTO
```



	SUMA	RESTA	MULTIPLICACION	DIVISION	RESTO
1	25.538	15.538	100.00	4.107600	0.538

```
SELECT IdProducto,  
       Descripcion,  
       Precio,  
       (Precio * 1.5) + 2 'Precio de venta'  
FROM Productos
```



	IdProducto	Descripcion	Precio	Precio de Venta
1	1000	ACEITE DE GIRASOL	1.65	4.475
2	1001	MAYONESA CON OLIVA NATURA FRA 500 CM3	1.34	4.010
3	1002	ACEITUNAS VERDES DOY PACK	5.55	10.325
4	1010	JUGO EN POLVO DIET POMELO ROSADO SER X 9.5 GRS	0.35	2.525

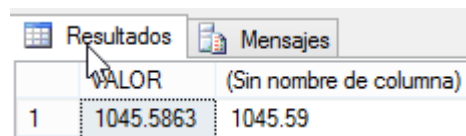
★ SqlServer lleva las operaciones al tipo de datos de mayor precision de los operandos ★

# SQL Server 2008 R2

Conversión de datos: Sql permite realizar gran variedad de conversiones de datos, desde correcciones de precisión hasta convertir datos numéricos y fechas en String, etc. Los comandos utilizados a tal efecto son: CAST y CONVERT, en menor medida también STR

CAST: **CAST**(ExpresiónAConvertir AS TipoDeDatoDeseado[longitud])

```
SELECT 1045.5863 AS VALOR,  
        CAST(1045.5863 AS NUMERIC(13,2))
```



VALOR	(Sin nombre de columna)
1045.5863	1045.59

CONVERT: **CONVERT**(TipoDeDatoDeseado[longitud], Expresión a Convertir [, estilo]) \*\*\*\*El estilo se usa mayormente para fechas,

```
SELECT IdProducto,  
       Descripcion,  
       Precio,  
       (Precio * 1.5253) + 2 AS 'P.de Venta',  
       CAST((Precio * 1.5253) + 2 AS NUMERIC(5,2)) AS 'P.de Venta 2',  
       CONVERT(NUMERIC(10,4), (Precio * 1.5253) + 2) AS 'P.de Venta 3',  
       CAST((Precio * 1.5253) + 2 AS INT) AS 'P.de Venta 4',  
FROM Productos
```

	IdProducto	Descripcion	Precio	P.de Venta	P.de Venta 2	P.de Venta 3	P.de Venta 4
1	1000	ACEITE DE GIRASOL	1.65	4.516745	4.52	4.5167	4
2	1001	MAYONESA CON OLIVA NATURA F...	1.34	4.043902	4.04	4.0439	4
3	1002	ACEITUNAS VERDES DOY PACK	5.55	10.465415	10.47	10.4654	10

```
SELECT IdProducto,  
       Descripcion,  
       FechaAlta,  
       CONVERT(VARCHAR(12), FechaAlta, 113) AS Fecha1,  
       CONVERT(VARCHAR(15), FechaAlta, 103) AS Fecha2,  
FROM Productos
```

	IdProducto	Descripcion	FechaAlta	Fecha1	Fecha2
1	1000	ACEITE DE GIRASOL	1999-04-13	13 Apr 1999	13/04/1999
2	1001	MAYONESA CON OLIVA NATUR...	2004-09-06	06 Sep 2004	06/09/2004
3	1002	ACEITUNAS VERDES DOY PACK	2010-02-24	24 Feb 2010	24/02/2010



# SQL Server 2008 R2

Clausula **WHERE** es la aliada y complemento de toda instrucción donde se necesite incorporar filtros “condiciones” a las consultas o manipulación de datos.

Sintaxis: WHERE <condición de búsqueda>

a) Condición simple:

```
SELECT *  
FROM Productos  
WHERE IdProducto = 1014  
  
SELECT *  
FROM Productos  
WHERE Marca = 'CANALE'
```

	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1014	ANANA LIGHT	CANALE	2011-12-14	2015-01-01	9.02

	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1014	ANANA LIGHT	CANALE	2011-12-14	2015-01-01	9.02
2	1015	DULCE DIET MEMBRILL CANALE FRA 610 GRM	CANALE	2010-10-04	NULL	7.33
3	1016	PERAS E/MITADES CANALE LAT 410 GRM	CANALE	2014-02-16	NULL	9.12
4	1207	FIDEOS CODITOS	CANALE	2010-06-15	NULL	8.71
5	1208	FIDEOS BAVETAS	CANALE	2011-12-14	NULL	2.13

b) B. Buscar las filas que contienen un valor como una parte de una cadena

```
SELECT TOP 2 *  
FROM Productos  
WHERE Descripcion LIKE '_HAM%'  
  
SELECT TOP 2 *  
FROM Productos  
WHERE Descripcion LIKE '%FIDEO%'
```

	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1028	SHAMPOO CERAMIDAS PLUS KERATINA	PLUSBELLE	2011-02-20	NULL	2.07
2	1352	shampoo PANTENE PVC-750-ml.	PANTENE	2010-06-15	NULL	2.38


  

	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1207	FIDEOS CODITOS	CANALE	2010-06-15	NULL	8.71
2	1208	FIDEOS BAVETAS	CANALE	2011-12-14	NULL	2.13

# SQL Server 2008 R2

c) Buscar filas utilizando un operador de comparación


```
SELECT TOP 2 *  
FROM Productos  
WHERE PRECIO >= 9
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1014	ANANA LIGHT	CANALE	2011-12-14	2015-01-01	9.02
2	1016	PERAS E/MITADES CANALE LAT 410 GRM	CANALE	2014-02-16	NULL	9.12

d). Buscar las filas que tienen un valor comprendido entre dos valores


```
SELECT *  
FROM Productos  
WHERE IdProducto BETWEEN 1028 AND 1034
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1028	SHAMPOO CERAMIDAS PLUS KERATINA	PLUSBELLE	2011-02-20	NULL	2.07
2	1029	ACONDICIONADOR CERAMIDAS PLUS KERATINA	PLUSBELLE	2013-02-26	2015-01-01	0.42

e) Buscar las filas que están en una lista de valores

```
SELECT *  
FROM Productos  
WHERE IdProducto IN(1040,1041,1043)
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1040	DURAZNOS EN MITAD 820 gr	NOEL	2010-10-04	NULL	0.42
2	1041	MERMELADA DE DAMASCO LIGHT	NOEL	2014-02-16	NULL	6.11

# SQL Server 2008 R2

Una Decisión Inteligente en Recursos Humanos

Búsqueda y selección de personal / Capacitación /  
Evaluación de personal / Asesoramiento Integral.




OPERADORES LOGICOS

**OR** → INCLUYENTE

**AND** → EXCLUYENTE

f) Buscar las filas que cumplen alguna de dos condiciones


```
SELECT *  
FROM Productos  
WHERE marca = 'ZORRO' or idProducto= 1032
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1027	ZORRO ESFERAS ACTIVAS 42 X 2 X 150 GR	ZORRO	2010-02-24	NULL	5.72
2	1030	Detergenten en polvo Zorro Natural Fresh Alta Esp...	ZORRO	2012-06-14	NULL	6.12
3	1031	Detergenten en polvo Zorro Natural Fresh Baja Es...	ZORRO	2010-06-15	NULL	5.83
4	1032	HARINA DE MAIZ # 500 gr	PRESTOPRONTA	2011-12-14	NULL	2.18

g) Buscar las filas que deben cumplir varias condiciones

```
SELECT *  
FROM Productos  
WHERE marca = 'ZORRO' AND  
Descripcion LIKE '%DETER%' AND  
Precio>6
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1030	Detergenten en polvo Zorro Natural Fresh Alta Es...	ZORRO	2012-06-14	NULL	6.12



# SQL Server 2008 R2

Casos especiales: BUSQUEDA DE VALORES **NULL**.

El valor **NULL** significa la no existencia de dato. No se debe confundir con vacío que es un dato de tipo char.

**NULL** puede estar en todos los tipos de datos numéricos, fechas, cadenas de texto, por citar algunos. Por esta razón tiene un operador propio para hacer las comparaciones: **IS NULL** para saber si un valor es nulo o **IS NOT NULL** para saber que no lo es.

```
SELECT TOP 2 *  
FROM Productos  
WHERE FechaBaja IS NULL  
  
SELECT TOP 2 *  
FROM Productos  
WHERE FechaBaja IS NOT NULL
```



	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1000	ACEITE DE GIRASOL	NATURA	1999-04-13	NULL	1.65
2	1001	MAYONESA CON OLIVA NATURA FRA 500 CM3	NATURA	2004-09-06	NULL	1.34


	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
1	1010	JUGO EN POLVO DIET POMELO ROSADO SER X 9.5 GRS	SER	2011-02-20	2015-01-01	0.35
2	1014	ANANA LIGHT	CANALE	2011-12-14	2015-01-01	9.02

# SQL Server 2008 R2

## Casos especiales: FILTROS DISTINCT / TOP

EL filtro **DISTINCT** se usa para evitar informar repeticiones de registros duplicados al momento de obtener los datos consultados (filtro pos consulta).

```
SELECT idcliente  
FROM PedidoCabecera  
where idcliente = 29
```



Resultados		Mensajes	
idCliente			
1	29		
2	29		

Para evitar el problema anterior puedo usar **DISTINCT**, este modificador de la consulta siempre tiene que anteceder a la primer columna del **SELECT**.

```
SELECT DISTINCT idcliente  
FROM PedidoCabecera  
where idcliente = 29
```



idCliente	
1	29

Otra herramienta que tenemos es el **TOP** que nos sirve para indicarle al motor de la base de datos que cantidad de registros devolvernos.

```
SELECT TOP 5 *  
FROM Clientes
```



	IdCliente	ApellidoRzSocial	Nombre	FechaAlta	FechaBaja	Telefono	Calle	CodigoPostal	IdProvincia
1	29	NOBUA	EDUARDO ANTONIO	2015-01-01	NULL	(0306) 737650	MAIPU 823	1722	ARU
2	90	BOTTO	HUGO ALBERTO	1995-01-01	NULL	(0306) 737650	PADRE BARBELLO 311	1742	ARC
3	284	PRINA	JOSE RICARDO	2015-01-01	NULL	(0306) 737650	MARIANO MORENO 1460	6700	ARB
4	716	MARTIN	CARLOS JULIAN	2013-08-01	NULL	(0306) 737650	CONSTANCIO C. VIGIL 251	1744	ARC
5	791	LANUSSE	PABLO	2007-01-01	NULL	(0306) 737650	MARCELO T. DE ALVEAR 1624 5º	1060	ARS

# SQL Server 2008 R2

Una Decisión Inteligente en Recursos Humanos

Búsqueda y selección de personal / Capacitación /  
Evaluación de personal / Asesoramiento Integral.



Definiendo Ordenamiento, cláusula **ORDER BY**

Tiene como objetivo ordenar el conjunto de resultados de una consulta por la lista de columnas especificada y, opcionalmente, limitar las filas devueltas a un intervalo especificado. **El orden en que se devuelven las filas en un conjunto de resultados no se puede garantizar, a menos que se especifique una cláusula ORDER BY.**

```
SELECT TOP 2 ApellidoRzSocial AS 'Apellido o Razon Social',
             IdCliente AS [Numero de Cliente],
             FechaBaja AS 'Fecha de Baja',
             'Valor fijo' AS Literal
FROM clientes
ORDER BY ApellidoRzSocial DESC

SELECT TOP 2 ApellidoRzSocial AS 'Apellido o Razon Social',
             IdCliente AS [Numero de Cliente],
             FechaBaja AS 'Fecha de Baja',
             'Valor fijo' AS Literal
FROM clientes
ORDER BY 2 ASC

SELECT TOP 2 ApellidoRzSocial AS 'Apellido o Razon Social',
             IdCliente AS [Numero de Cliente],
             FechaBaja AS 'Fecha de Baja',
             'Valor fijo' AS Literal
FROM clientes
ORDER BY 'Fecha de Baja' DESC
```

Resultados				
	Apellido o Razon Social	Numero de Cliente	Fecha de Baja	Literal
1	ZUK	60806	NULL	Valor fijo
2	ZUCCARDI	76786	NULL	Valor fijo
	Apellido o Razon Social	Numero de Cliente	Fecha de Baja	Literal
1	NOBUA	29	NULL	Valor fijo
2	BOTTO	90	NULL	Valor fijo
	Apellido o Razon Social	Numero de Cliente	Fecha de Baja	Literal
1	CAMPI	16005	2015-01-29	Valor fijo
2	ROBLEDO	23849	2015-01-29	Valor fijo

Como vemos se puede ordenar especificando:

- Nombre de columna,
- Número de columna en SELECT
- Por nombre del Alias
- Combinación de las tres anteriores

# SQL Server 2008 R2

## JOINS..... UNIENDO TABLAS Y CONSULTAS

Las combinaciones permiten recuperar datos de dos o más tablas según las relaciones lógicas entre ellas.

Las combinaciones indican cómo debe usar Microsoft SQL Server los datos de una tabla para seleccionar las filas de otra tabla.

Una condición de combinación define la forma en la que dos tablas se relacionan en una consulta al Especificar la columna de cada tabla que debe usarse para la combinación.

Una condición de combinación típica especifica una clave externa de una tabla y su clave asociada en otra tabla.

**OTRA VISION QUE PUEDE AYUDAR A PENSAR EN COMO UNIR LAS TABLAS ES PENSARLOS COMO INTERSECCIONES, UNIONES Y COMPLEMENTOS DE CONJUNTOS....**

# SQL Server 2008 R2

## INNER JOIN → LA INTERSECCION

Es el tipo de combinación mas usada. Sólo devuelve las filas en las que haya un valor igual en la/s columna/s de la combinación. COINCIDENCIA TOTAL ENTRE TODAS LAS COLUMNAS DE COMBINACION.

Para resolver las combinaciones en general es necesario incorporar alias de tablas para saber determinar a que tabla pertenece cada columna. En este ejemplo usamos alias **c** para la tabla Clientes y **pr** para la tabla Provincias.

```
SELECT      pr.Descripcion,  
            c.ApellidoRzSocial  
FROM Clientes c  
INNER JOIN Provincias pr  
ON c.IdProvincia = pr.IdProvincia  
WHERE c.ApellidoRzSocial LIKE '%MART%'
```

Alias de tabla

Columnas de combinación (siempre conviene  
Igualarlas según el orden de aparición de las tablas  
Desde el FROM hasta el ultimo JOIN)

	Descripcion	ApellidoRzSocial
1	Ciudad Autónoma de Buenos Aires	MARTIN
2	Ciudad Autónoma de Buenos Aires	DEMARTA
3	Ciudad Autónoma de Buenos Aires	MARTINEZ
4	La Pampa	MARTINO
5	Ciudad Autónoma de Buenos Aires	MARTIELLO



# SQL Server 2008 R2

**LEFT JOIN** → CONSERVA LA TABLA PRECEDENTE (A IZQUIERDA)

Este tipo de combinación se usa cuando se necesita unir una tabla con otra que probablemente no tiene todos los datos que se pretenden combinar, posibilitando mantener los datos de las tablas o combinaciones que anteceden a su uso (Izquierda).

```
SELECT p.IdProducto, p.Descripcion, P.marca
FROM Productos p
WHERE Marca LIKE '%CELUSAL%'
```

	IdProducto	Descripcion	marca
1	1179	SAL FINA SALERO CELUSAL X 100 GRS	CELUSAL
2	1180	SAL FINA LIGHT SALERO CELUSAL X 500 GRS	CELUSAL
3	1181	SAL FINA CON HIERRO SALERO CELUSAL X 500 GRS	CELUSAL
4	1182	SAL ENTREFINA SALERO CELUSAL X 1000 GRS	CELUSAL
5	1183	SAL GRUESA SALERO CELUSAL X 1000 GRS	CELUSAL

La última fila, en las últimas dos columnas contiene valores nulos debido a que no existen pedidos para el producto 1183

```
SELECT p.IdProducto,
       p.Descripcion,
       p.marca,
       pd.idpedido,
       pd.CantidadSolicitada
FROM Productos p
LEFT JOIN PedidoDetalle pd
ON p.IdProducto = pd.IdProducto
WHERE Marca LIKE '%CELUSAL%'
```

	IdProducto	Descripcion	marca	idpedido	CantidadSolicitada
1	1179	SAL FINA SALERO CELUSAL X 100 GRS	CELUSAL	38	1
2	1180	SAL FINA LIGHT SALERO CELUSAL X 500 GRS	CELUSAL	38	1
3	1181	SAL FINA CON HIERRO SALERO CELUSAL X 500 GRS	CELUSAL	38	1
4	1182	SAL ENTREFINA SALERO CELUSAL X 1000 GRS	CELUSAL	38	1
5	1183	SAL GRUESA SALERO CELUSAL X 1000 GRS	CELUSAL	NULL	NULL

# SQL Server 2008 R2

**RIGHT JOIN** → CONSERVA LA TABLA SIGUIENTE (A DERECHA)

Este tipo de combinación se usa cuando se necesita unir una tabla con otra que probablemente no tiene mas datos que se los que se pueden combinar, posibilitando mantener los datos de las tablas o combinaciones que preceden (Derecha).

```
SELECT COUNT(1) AS 'Cantidad Clientes'  
FROM Clientes  
SELECT COUNT(1) AS 'Cantidad Provincias'  
FROM Provincias
```

Cantidad Clientes	
1	991

Cantidad Provincias	
1	24

Vamos a combinar los clientes con las provincias utilizando como relación la columna idProvincia, el resultado deberá Contener por el tipo de consulta **RIGHT JOIN** todos los resultados de la tabla derecha, en este caso **Provincias**. En otras palabras es como decir traeme todas las provincias y los clientes que están en ellas, incluyendo aquellas provincias que no tienen clientes.

```
SELECT c.IdCliente, c.ApellidoRzSocial, p.*  
FROM Clientes c  
RIGHT JOIN Provincias p  
ON c.IdProvincia = p.IdProvincia
```

	IdCliente	ApellidoRzSocial	IdProvincia	Descripcion	FlagVentaHabilitada
990	99598	ISAR	ARB	Provincia de Buenos Aires	1
991	99886	GARCIA	ARU	Chubut	1
992	NULL	NULL	ARN	Misiones	1
993	NULL	NULL	ARH	Chaco	0
994	NULL	NULL	ARO	Formosa	1

# SQL Server 2008 R2

**FULL JOIN** → CONSERVA TODOS LOS DATOS DE AMBAS = (UNION DE CONJUNTOS)

Este tipo de combinación se usa cuando se necesita unir TODOS los registros o filas de ambas tablas de la combinación, completando con valores NULL de una tabla o de la otra en los casos que no hay coincidencia.

```
SELECT COUNT(1) AS 'Cant.Detalle'  
FROM PedidoDetalle  
SELECT COUNT(1) AS 'Cant.Productos'  
FROM Productos
```

Cant.Detalle	
1	182

Cant.Productos	
1	384

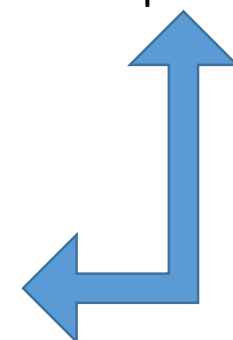
Vamos a combinar los detalles de los pedidos con todos nuestros productos. El resultado es la unión de ambas tablas.

El resultado de esta combinación nunca será mayor que la suma de sus registros.

```
SELECT Pd.*, P.*  
FROM PedidoDetalle pd  
FULL JOIN Productos p  
ON pd.IdProducto = p.IdProducto
```

Como vemos en la fila 77 y 78 se devolvieron productos que no están en ningún pedido.

	IdPedido	IdPedidoDetalle	IdProducto	CantidadSolicitada	IdProducto	Descripcion	Marca	FechaAlta	FechaBaja	Precio
74	70	175	1092	1	1092	Iguana Retomable 970cc	IGUANA	2010-02-24	NULL	6.33
75	70	176	1093	1	1093	gaseosa MIRINDA naranja	MIRINDA	2011-02-20	NULL	8.88
76	70	177	1094	1	1094	QUILMES BAJO CERO RUBIA CERV - 1/1 VIDRIO RET.	QUILM...	2013-02-26	NULL	1.00
77	NULL	NULL	NULL	NULL	1095	PEPSI REGULAR COLA GASE - GRB	PEPSI	2012-06-14	NULL	6.19
78	NULL	NULL	NULL	NULL	1096	7UP REGULAR LIMA LIMON GASE - GRB.	7UP	2010-06-15	NULL	8.32



# SQL Server 2008 R2

**CROSS JOIN** → TODOS CONTRA TODOS...Producto cartesiano de dos tablas.

Este tipo de combinación se usa cuando se necesita unir TODOS los registros o filas de una table contra TODOS los registro de la otra tablas. La cantidad o CARDINALIDAD del resultado se obtiene de multiplicar los registros de la primer tabla por los registros de la segunda.

```
SELECT *  
FROM clientes c  
CROSS JOIN Productos p  
ORDER BY IdCliente
```

	IdCliente	ApellidoRzSocial	Nombre	IdProducto	Descripcion	Precio	
1	29	NOBUA	EDUARDO ANTONIO	1000	ACEITE DE GIRASOL	1.65	^
2	29	NOBUA	EDUARDO ANTONIO	1001	MAYONESA CON OLIVA NATURA FRA 500 CM3	1.34	
3	29	NOBUA	EDUARDO ANTONIO	1002	ACEITUNAS VERDES DOY PACK	5.55	
4	29	NOBUA	EDUARDO ANTONIO	1010	JUGO EN POLVO DIET POMELO ROSADO SER X 9.5 GRS	0.35	▼

✓ Consulta ejecutada correctamente. | NBTRADITUMMET\SQLEXPRESS200... | USUARIOS\metejerina (52) | CursoSql | 00:00:02 | 380544 filas

Como observamos en la consulta anterior, se devolvieron 380544 filas, lo que implica que se hizo el producto cartesiano de 991 clientes y 384 productos

# SQL Server 2008 R2

**JOIN's** → Por lo general cuando necesitamos obtener Información es altamente probable que tengamos que consultar mas de una tabla. En esos casos necesitaremos conocer el MODELO (como están relacionadas las tablas entre sí)  
En el siguiente supuesto queremos traer todos los pedidos cargados, tenemos que tener en cuenta que no todos tienen un detalle asociado. Se pide el número de pedido, el apellido del cliente que lo solicitó, la descripción de la provincial del cliente y en caso que haya detalle la descripción y cantidad de producto solicitada,

```
SELECT pc.IdPedido      'Nro. Pedido',  
       c.ApellidoRzSocial 'Cliente',  
       p.Descripcion    'Provincia',  
       pr.descripcion    'Producto',  
       pd.CantidadSolicitada 'Cantidad'  
FROM PedidoCabecera pc  
INNER JOIN Clientes c  
    ON pc.IdCliente = c.IdCliente  
INNER JOIN Provincias P  
    ON c.IdProvincia = p.IdProvincia  
LEFT JOIN PedidoDetalle pd  
    ON pc.IdPedido = pd.IdPedido  
LEFT JOIN Productos pr  
    ON pd.IdProducto = pr.IdProducto  
ORDER BY pc.IdPedido
```

	Nro. Pedido	Cliente	Provincia	Producto	Cantidad
197	62	ROD...	Ciudad ...	máquina de afeitar GILLETTE women BLI-2-un.	1
198	62	ROD...	Ciudad ...	TOALLA HIG.NORMAL S/A SECA BASICA	1
199	63	CAMPI	Jujuy	NULL	NULL
200	64	GALE...	La Rioja	ACEITE DE GIRASOL	2
201	64	GALE...	La Rioja	CAFÉ DOLCA SUAVE LATA	2
202	64	GALE...	La Rioja	GELATINA DE FRUTILLA	2
203	65	LOMB...	Entre R...	NULL	NULL
204	66	GAW...	Provinc...	NULL	NULL
205	67	SOM...	Provinc...	NULL	NULL
206	68	PALUDI	Entre R...	PAPEL HIGIENICO CLASSIC 36 m2	1
207	68	PALUDI	Entre R...	AMARGO SERRANO BLANCO PET	1
208	68	PALUDI	Entre R...	AMARGO SERRANO PET	1
209	68	PALUDI	Entre R...	AMERICANO	1
210	68	PALUDI	Entre R...	FERNET	1



# SQL Server 2008 R2

**SELF JOIN** → No es una nueva clase de join sino una técnica para combinar una tabla consigo misma. El ejemplo que se muestra a continuación es solo a efectos pedagógicos ya que el mismo resultado podría obtenerse de otra manera más eficiente. (se usa el los diferentes join que se requieran INNER / LEFT / RIGHT / FULL / CROSS)

```
SELECT C.*  
FROM clientes C  
INNER JOIN clientes C2  
ON C.IdCliente = C2.IdCliente AND  
C2.FechaBaja IS NOT NULL  
WHERE C2.FechaAlta <= '20140101'
```

	IdCliente	ApellidoRzSocial	Nombre	FechaAlta	FechaBaja	Telefono	Calle	CodigoPostal	IdProvincia
1	3502	GAWER	SERGIO ENRIQUE	2004-08-01	2014-10-04	(0306) 737650	JUAN XXIII 229	1748	ARB
2	4050	SOMMA	LUIS FERNANDO	2012-10-01	2013-11-29	(0306) 737650	DOCTOR REAL 1124	6700	ARB
3	7951	PALUDI	JUAN CARLOS	2013-04-01	2014-09-04	(0306) 737650	RAFAEL HERNANDEZ 2649	1428	ARE
4	8346	LEGARRETA	CARLOS AROLDI	2007-11-01	2014-08-13	(0306) 737650	CHARCAS 3391 6	1425	ARF
5	8523	PEÑALVA	ALEJANDRO HORACIO	2007-01-01	2012-05-15	(0306) 737650	TRATADO DEL PILAR Y RUTA 8 0	1629	ARB
6	10405	RODRIGUEZ KISSNER	JORGE	2004-01-01	2013-08-10	(0306) 737650	CALLE 143 E/13 Y 14 1357	1884	ARZ
7	13565	RODRIGUEZ BUSSON	ROBERTO	1995-01-01	2014-11-26	(0306) 737650	BAHIA BLANCA 2739 2	1417	ARC

# SQL Server 2008 R2

## CLASE 4

**SUB CONSULTAS** → Se trata de una consulta que se escribe dentro de otra.

La sintáxis de una consulta es idéntica a las consultas “NORMALES”, excepto que las subconsultas por estar dentro de otras siempre van entre paréntesis.

### PRIMER CASO SUB CONSULTA ANIDADA

Este tipo de consulta se hace por cada registro que devuelve la consulta principal “NORMAL”

```
SELECT  c.IdCliente          AS 'NroCliente',  
        c.ApellidoRzSocial  AS 'cliente',  
        (SELECT COUNT(*)  
         FROM PedidoCabecera pc  
         WHERE pc.IdCliente = c.IdCliente)  
        AS 'Pedidos'  
FROM  clientes c
```

	NroCliente	Cliente	Pedidos
1	29	NOBUA	2
2	90	BOYTO	2
3	284	PRINA	2
4	716	MARTIN	2
5	791	LANUSSE	2
6	906	MEDINA	1
7	955	UGARTEMENDIA	1
8	957	ORTIZ	1
9	982	CAMBRES	1
10	1009	KLEIN	1
11	1032	HOI I MANN	1

# SQL Server 2008 R2

## SEGUNDO CASO SUB CONSULTA FILTRADA por IN o NOT IN

```
SELECT TOP 5  c.IdCliente      AS 'NroCliente',  
              c.ApellidoRzSocial AS 'cliente'  
FROM clientes c  
WHERE c.IdCliente IN (SELECT IdCliente FROM PedidoCabecera)  
ORDER BY 1 ASC  
  
SELECT TOP 5  c.IdCliente      AS 'NroCliente',  
              c.ApellidoRzSocial AS 'cliente'  
FROM clientes c  
WHERE c.IdCliente NOT IN (SELECT IdCliente FROM PedidoCabecera)  
ORDER BY 1 ASC
```

	NroCliente	Cliente
1	29	NOBUA
2	90	BOTTO
3	284	PRINA
4	716	MARTIN
5	791	LANUSSE

	NroCliente	Cliente
1	5758	SOLARI
2	6024	SABAN
3	6097	LAIZ GARCIA
4	6138	CALCAGNI
5	6326	RIOS VELAR

La primer consulta devuelve los primeros 5 clientes que hicieron un pedido

La segunda consulta devuelve los primeros 5 clientes que no realizaron un pedido

# SQL Server 2008 R2

## TERCER CASO SUB CONSULTA PARA COMPROBACION DE EXISTENCIA – EXISTS | NOT EXISTS

```
SELECT TOP 5  C.IdCliente          AS 'NroCliente',  
              C.ApellidoRzSocial  AS 'Cliente'  
FROM clientes C  
WHERE EXISTS(SELECT 1 FROM PedidoCabecera pc  
              where pc.IdCliente = c.IdCliente AND pc.FechaBajaPedido IS NOT NULL)  
ORDER BY 1 ASC  
  
SELECT TOP 5  C.IdCliente          AS 'NroCliente',  
              C.ApellidoRzSocial  AS 'Cliente'  
FROM clientes C  
WHERE EXISTS(SELECT 1 FROM PedidoCabecera pc  
              where pc.IdCliente = c.IdCliente AND pc.FechaBajaPedido IS NULL)  
ORDER BY 1 ASC
```

	NroCliente	Cliente
1	284	PRINA
2	1032	HOLLMANN
3	1809	MITRE
4	4050	SOMMA
5	4594	ROJAS

	NroCliente	Cliente
1	29	NOBUA
2	90	BOTTO
3	716	MARTIN
4	791	LANUSSE
5	906	MEDINA

La primer consulta devuelve los primeros 5 clientes que EXISTEN en la tabla De pedidos y cuya fecha de baja es diferente a NULL

La segunda consulta devuelve los primeros 5 clientes que EXISTEN en la tabla De pedidos y cuya fecha de baja es NULL

# SQL Server 2008 R2

Una Decisión Inteligente en Recursos Humanos

Búsqueda y selección de personal / Capacitación /  
Evaluación de personal / Asesoramiento Integral.



Servicio de Empleo

## CUARTO CASO – SUB CONSULTAS CORRELACIONADAS

```
SELECT      C.IdCliente      AS 'NroCliente',  
            C.ApellidoRzSocial AS 'Cliente',  
            PC.IDpedido      AS 'Pedido'  
FROM (SELECT * FROM Clientes  
      WHERE FechaBaja IS NOT NULL) C  
INNER JOIN (SELECT * FROM PedidoCabecera  
            WHERE FechaBajaPedido IS NOT NULL) PC  
ON C.IdCliente = PC.IdCliente  
  
SELECT C.ApellidoRzSocial, C.FechaBaja FROM Clientes C  
WHERE IdCliente IN (4050,18381,20103,4050)  
  
SELECT P.IdPedido, P.IdCliente, P.FechaBajaPedido  
FROM PedidoCabecera P  
WHERE IdPedido IN (57,64,65,67)
```

La primer consulta busca los clientes que están dados de baja, una vez que los obtiene pasa a buscar los pedidos de esos clientes que tambien están dados de baja.

Las otras dos consultas son la comprobación de la primera

	NroCliente	Cliente	Pedido
1	4050	SOMMA	57
2	18381	GALENDE	64
3	20103	LOMBRONI	65
4	4050	SOMMA	67

	ApellidoRzSocial	FechaBaja
1	SOMMA	2013-11-29
2	GALENDE	2014-10-03
3	LOMBRONI	2012-12-27

	IdPedido	IdCliente	FechaBajaPedido
1	57	4050	2000-10-16 00:00:00.000
2	64	18381	2000-10-16 00:00:00.000
3	65	20103	2000-10-16 00:00:00.000
4	67	4050	2000-10-16 00:00:00.000



# SQL Server 2008 R2

UNION / UNION ALL / INTERSECT / EXCEPT

## UNION

Especifica que se deben combinar varios conjuntos de resultados para ser devueltos como un solo conjunto de resultados.

## UNION ALL

Agrega todas las filas a los resultados. Incluye las filas duplicadas. Si no se especifica, las filas duplicadas se quitan.

EXCEPT devuelve filas distintas de la consulta de entrada izquierda que no son de salida en la consulta de entrada derecha. (Excluye los resultados de la consulta anterior)

INTERSECT devuelve filas distintas que son de salida en las consultas de entrada izquierda y derecha. (coincidentes)  
Las reglas básicas para combinar los conjuntos de resultados de dos consultas que utilizan EXCEPT o INTERSECT son las siguientes:

El número y el orden de las columnas debe ser el mismo en todas las consultas.

Los tipos de datos deben ser compatibles.

# UPDATE (Transact-SQL)

Es utilizado para modificar registros existentes en una tabla o en un vista (MS SQL Server)

## Sintaxis SQL UPDATE

```
UPDATE nombre_tabla  
SET columna1=valor1,column2=valor2,...  
WHERE alguna_columna=algun_valor;
```

## Base para pruebas

A continuación, una selección de la tabla “Clientes”

	IdCliente	ApellidoRzSocial	Nombre	FechaAlta	FechaBaja	Telefono	Calle	CodigoPostal	IdProvincia
1	29	NOBUA	EDUARDO ANTONIO	2015-01-01	NULL	(0306) 737650	MAIPU 823	1722	ARU
2	90	BOTTO	HUGO ALBERTO	1995-01-01	NULL	(0306) 737650	PADRE BARBELLO 311	1742	ARC
3	284	PRINA	JOSE RICARDO	2015-01-01	NULL	(0306) 737650	MARIANO MORENO 1460	6700	ARB
4	716	MARTIN	CARLOS JULIAN	2013-08-01	NULL	(0306) 737650	CONSTANCIO C. VIGIL 251	1744	ARC
5	791	LANUSSE	PABLO	2007-01-01	NULL	(0306) 737650	MARCELO T. DE ALVEAR 1624 5º	1060	ARS

# UPDATE (Transact-SQL)

## Ejemplo SQL UPDATE

Se supone que se quiere modificar al cliente “JOSE RICARDO PRINA” y actualizar la calle y el código postal.

Se utiliza, entonces, la siguiente declaración:

```
UPDATE Clientes  
SET Calle='CHARCAS 348',CodigoPostal='1325'  
WHERE IdCliente='284';
```

Ahora, una selección de la tabla “Clientes” se verá así:

	IdCliente	ApellidoRzSocial	Nombre	FechaAlta	FechaBaja	Telefono	Calle	CodigoPostal	IdProvincia
1	29	NOBUA	EDUARDO ANTONIO	2015-01-01	NULL	(0306) 737650	MAIPU 823	1722	ARU
2	90	BOTTO	HUGO ALBERTO	1995-01-01	NULL	(0306) 737650	PADRE BARBELLO 311	1742	ARC
3	284	PRINA	JOSE RICARDO	2015-01-01	NULL	(0306) 737650	Charcas 348	1325	ARB

## ¡ATENCIÓN! ¡CUIDADO!

Si se omite la cláusula WHERE, ¿qué sucede?

```
UPDATE Clientes  
SET Calle='CHARCAS 348',CodigoPostal='1325';
```

# DELETE (Transact-SQL)

Se utiliza para eliminar filas de una tabla

## Sintaxis SQL DELETE

```
DELETE FROM nombre_tabla  
WHERE alguna_columna=algun_valor;
```

## Ejemplo SQL DELETE

Se busca eliminar al cliente "PRINA" de la table Clientes

```
DELETE FROM Clientes  
WHERE IdCliente='284';
```

## ¡ATENCIÓN! ¡CUIDADO!

Si se omite la cláusula WHERE, ¿qué sucede?  
(y no existe "deshacer")

# TRUNCATE (Transact-SQL)

¿Qué pasa si queremos deshacernos de los datos en una tabla –pero no de la tabla en sí-? Es decir, si buscamos “vaciarla”.

## Sintaxis TRUNCATE

```
TRUNCATE TABLE nombre_table;
```

# DROP (Transact-SQL)

Se utiliza para eliminar una tabla completa

## Sintaxis DROP

```
DROP TABLE nombre_table;
```



# TRUNCATE vs DELETE

- Equivalentes a nivel lógico
- No equivalentes a nivel físico
- Cláusula de condición WHERE
- Eliminación de filas (una por vez vs. todas)
- Performance
- Rollback



# FUNCIONES DE AGREGACIÓN

Son las funciones que devuelven un único valor, calculado a partir de varios valores de una columna.

- **AVG ():** Devuelve el valor promedio
- **COUNT ():** Calcula el número de filas
- **FIRST ():** Devuelve el primer valor
- **LAST ():** Devuelve el ultimo valor
- **MAX ():** Devuelve el valor más grande
- **MIN ():** Devuelve el valor más pequeño
- **SUM ():** Calcula la sumatoria

# FUNCIÓN AVG ()

Devuelve el valor promedio de una columna numérica

## Sintaxis AVG ()

```
SELECT AVG (nombre_columna) FROM Nombre_table;
```

## Ejemplos

A continuación, se obtiene el valor promedio de la columna precio, de la tabla Productos:

```
SELECT AVG(Precio) AS PrecioPromedio FROM Productos
```

El siguiente ejemplo permite conocer los productos cuyo precio está por encima del promedio. El resultado ordenado en forma ascendente de acuerdo a su precio.

```
SELECT Descripcion, Precio FROM Productos  
WHERE Precio > (SELECT AVG(Precio) FROM Productos)  
ORDER BY Precio ASC
```

# FUNCIÓN COUNT ()

Devuelve el número de filas según criterio específico

## Sintaxis COUNT ()

```
SELECT COUNT(nombre_columna) FROM Nombre_table;
```

## Ejemplos

```
SELECT COUNT (IdProducto) FROM Productos  
WHERE Precio>(SELECT AVG(Precio) FROM Productos);
```

```
SELECT COUNT (IdProducto) AS CANTIDAD_PRODUCTOS FROM Productos;
```

# FUNCIONES FIRST & LAST

Devuelven primer y último valor

## Ejemplos

```
SELECT TOP 1 Descripcion, Precio from Productos  
ORDER BY Descripcion
```

```
SELECT TOP 1 Descripcion, Precio from Productos  
ORDER BY Descripcion DESC
```



Ministerio de  
Trabajo, Empleo  
y Seguridad Social

Formación  
Continua

cessi  
Argentina

EMPLEARTEC.ORG.AR



# FUNCIONES MIN & MAX

Devuelven los valores más pequeños o más grandes de una columna seleccionada

## Ejemplos

```
SELECT MAX(Precio) AS PrecioMásAlto FROM Productos;
```

```
SELECT MIN(Precio) AS PrecioMásBajo FROM Productos;
```

# FUNCIÓN SUM ()

Devuelve la suma de todos los valores indicados

## Sintaxis SUM ()

```
SELECT SUM(column_name) FROM nombre_tabla;
```

## Ejemplo

A continuación, se buscar obtener la cantidad de productos solicitados

```
SELECT SUM(CantidadSolicitada) FROM PedidoDetalle;
```

```
SELECT SUM(CantidadSolicitada) AS CantidadTotal FROM PedidoDetalle;
```



# REPASO FUNCIONES AGREGADAS

- ✓ MS SQL Server posee funciones que permiten contar registros, calcular sumas, promedios, obtener valores mínimos y máximos.
- ✓ Se pueden utilizar en una instrucción SELECT y son combinables con la cláusula "GROUP BY"
- ✓ Existen relaciones entre las funciones y los tipos de datos
  - COUNT: se puede emplear con cualquier tipo de dato
  - MIN y MAX: con cualquier tipo de dato
  - SUM y AVG: solo en campos de tipo numérico
- ✓ Tratamiento de los valores nulos
  - COUNT (\*): incluye los valores nulos de los campos
  - Resto de las funciones: excluyen los valores nulos de los campos

# RESUMIENDO DATOS: GROUP BY

- ✓ Las funciones de agregado permiten realizar varios cálculos operando con conjuntos de registros.
- ✓ La declaración GROUP BY es utilizada en conjunto con las funciones agregadas para agrupar el conjunto resultante en una o más columnas.
- ✓ Las funciones de agregado, en soledad, producen un valor de resumen para todos los registros de un campo.
- ✓ ¿Cómo generar valores de resumen para un solo campo? Combinando las funciones de agregado con la cláusula "GROUP BY", que agrupa registros para consultas detalladas.

## Sintaxis GROUP BY

```
SELECT columna, función_agregada(columna)  
      FROM tabla  
      WHERE columna operador valor  
      GROUP BY columna;
```

# RESUMIENDO DATOS: HAVING

Mientras la cláusula WHERE permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite la misma acción pero con grupos de registros.

Entonces:

- ✓ Especifica una condición de búsqueda para un grupo o agregado.
- ✓ HAVING solo se puede utilizar con la instrucción SELECT.
- ✓ Normalmente, HAVING se utiliza en una cláusula GROUP BY.
- ✓ Cuando no se utiliza GROUP BY, HAVING se comporta como una cláusula WHERE (M\$)
- ✓ No confundir las cláusulas WHERE y HAVING. Una establece condiciones para la selección de registros de un SELECT; la segunda condiciones para la selección de registros de una salida GROUP BY.
- ✓ Puede utilizarse con funciones de agrupamiento (la cláusula WHERE no puede hacer esto)

## Sintaxis HAVING

```
SELECT columna1, FUNCIÓN(columna2)  
FROM tabla  
WHERE columna2 operador valor  
GROUP BY columna1  
HAVING FUNCIÓN(columna) operador valor;
```

Entonces, se usa la cláusula HAVING para restringir las filas que devuelve una salida GROUP BY.  
(Va siempre después de la cláusula GROUP BY antes de la cláusula ORDER BY si la hubiere.)

# COMPUTE & COMPUTE BY

- ✓ Son cláusulas que generan totales que aparecen en columnas extras al final del resultado.
- ✓ Se utilizan con las funciones de agrupamiento: avg(), count(), max(), min(), sum().
- ✓ En la misma instrucción se puede incluir varias cláusulas "compute?"

## Sintaxis COMPUTE

```
SELECT columna1, columna2  
FROM tabla  
COMPUTE FUNCION(columna1)
```

Entonces, se usa la cláusula HAVING para restringir las filas que devuelve una salida GROUP BY.  
(Va siempre después de la cláusula GROUP BY antes de la cláusula ORDER BY si la hubiere.)



# COMPUTE & COMPUTE BY

- ✓ Son cláusulas que generan totales que aparecen en columnas extras al final del resultado.
- ✓ Se utilizan con las funciones de agrupamiento: avg(), count(), max(), min(), sum().
- ✓ En la misma instrucción se puede incluir varias cláusulas 'compute'

## Sintaxis COMPUTE & COMPUTE BY

```
SELECT columna1, columna2 FROM tabla  
order by columna1, columna2  
COMPUTE FUNCION1(columna1), FUNCION2(columna2)  
by columna1, columna2
```

- ✓ "Compute by" genera cortes de control y subtotales.
- ✓ Con "compute by" se DEBE usar también la cláusula "order by" y utilizar los mismos campos (o menos) y en mismo orden.
- ✓ Listando varios campos luego del "by" corta un grupo en subgrupos y aplica la función de agregado en cada nivel de agrupamiento.

# IDENTITY

- Un campo numérico puede tener un atributo extra "identity". Los valores de un campo con este atributo genera valores secuenciales que se inician en 1 y se incrementan en 1 automáticamente.
- Se usa en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta
- El campo debe ser entero
- Cuando un campo tiene el atributo "identity" no se puede ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.
- Si se elimina el último registro ingresado (por ejemplo 3) y luego se inserta otro registro, SQL Server seguirá la secuencia, es decir, colocará el valor "4".

```
create table libros(  
codigo int identity,  
titulo varchar(40) not null,  
autor varchar(30),  
editorial varchar(15),  
precio float  
);
```

# SUMARIZANDO: ROLL UP

- ✓ El operador "rollup" resume valores de grupos.
- ✓ Es posible incluir varias funciones de agrupamiento
- ✓ Por cada agrupación aparece una fila extra con valores de resumen
- ✓ Con "rollup" se puede emplear "where" y "having«
- ✓ Entonces, es un modificador para GROUP BY

## Sintaxis ROLL UP

```
SELECT columna, FUNCION(columna) FROM tabla  
group by columna WITH ROLL UP
```

# SUMARIZANDO: CUBE

- ✓ Genera filas de resumen de subgrupos para todas las combinaciones posibles de los valores de los campos por los que agrupamos
- ✓ Se pueden colocar hasta 10 campos en el "group by".
- ✓ Con "cube" se puede emplear "where" y "having"

## Sintaxis ROLL UP

```
SELECT columna, FUNCION(columna) FROM tabla  
group by columna WITH ROLL UP
```

# SUMARIZANDO: CUBE

- ✓ Mientras que “rollup” agrega filas extras resumiendo resultados por grupo y subgrupo...
- ✓ “cube” genera filas de resumen de subgrupos para todas las combinaciones posibles de valores de campos
- ✓ Se pueden colocar hasta 10 campos en el “group by”
- ✓ Se pueden emplear “where” y “having”

## Sintaxis CUBE

```
SELECT columna, FUNCION(columna) FROM tabla  
group by columna WITH CUBE
```

- ✓ Genera un conjunto de resultados que es un cubo multidimensional
- ✓ Cuya expansión se basa en las columnas que se deseen analizar

# GROUPING

- ✓ Función utilizada con operadores “rollup” y “cube”
- ✓ Se utiliza para distinguir, en el resultado, valores de detalle y de resumen
- ✓ Permite diferenciar si valores “null” son valores de tablas o filas generadas por los operadores “rollup” o “cube”
- ✓ Genera una nueva columna por cada “grouping”
- ✓ Valor 1 -> valores de resumen “rollup” o “cube”

## Sintaxis Grouping

```
SELECT columna, FUNCION(columna)  
GROUPING(columna) as alias  
FROM tabla  
group by columna with rollup;
```

- ✓ Se utiliza cuando no se puede distinguir la fila generada por “rollup” o “cube” de otras (null)
- ✓ Por lo tanto, si se utiliza “cube” o “rollup” y los campos empleados para agrupar admiten valores null -> utilizar “grouping”



# ÍNDICES

- ✓ SQL Server accede a los datos de dos maneras: Recorriendo tablas o empleando índices
- ✓ Es decir, registro x registro vs. estructura de árbol del índice
- ✓ Una tabla se indexa por un campo (o varios)
- ✓ Eficientizan las búsquedas -> Obejtivo es mejorar performance
- ✓ Genera espacio en disco
- ✓ Mejores índices=creados con campos con valores únicos
- ✓ Campos a indexar: pk, fk o campos que combinan tablas
- ✓ SQL Server los crea automáticamente bajo ciertas restricciones

## Dos tipos: agrupados y no agrupados

- ✓ Permiten el acceso directo
- ✓ Aceleren búsquedas, consultas y otras operaciones
- ✓ Optimizan rendimiento general

# ÍNDICES AGRUPADOS (cluster)

- ✓ Similar a guía telefónica
- ✓ Para campos utilizados por búsquedas con frecuencia
- ✓ Sólo puede haber 1 índice agrupado
- ✓ Modifican el orden físico de los registros, ordenándolos secuencialmente
- ✓ -> El orden lógico de los valores de clave determina el orden físico de las filas de la tabla

# ÍNDICES NO AGRUPADOS

- ✓ Índice de un libro
- ✓ No están ordenados físicamente -> estructura adicional
- ✓ Los punteros indican el lugar de almacenamiento
- ✓ Para cuando se realizan distintos tipos de búsqueda frecuentes
- ✓ Pueden existir hasta 249 índices no agrupados

**Dif.básica->** ✓ Los agrupados están ordenados y almacenados en forma secuencial en función de su clave

# CONSIDERACIONES (al crearlos)

- ✓ Evitar crear demasiada cantidad (sobre todo en tablas que se actualizan con mucha frecuencia)
- ✓ Tablas con poca actualización & gran volumen de datos -> mayor número de índices
- ✓ Gran número de índices -> SELECT

## Sintaxis INDICES (creación)

```
CREATE INDEX nombre_indice  
ON tabla (columna)
```

# VISTAS

- ✓ Es una alternativa para mostrar datos de varias tablas
- ✓ Tabla virtual que almacena una consulta
- ✓ Sus datos no son objeto de la DB
- ✓ En gral., se puede nombrar cualquier consulta y almacenarla como una vista

## permiten

- ✓ Ocultar información
- ✓ Simplificar la administración de la seguridad
- ✓ Mejorar el rendimiento (evitando tipear instrucciones repetidamente)
- ✓ Vista= subconjunto de registros y campos de una tabla, unión de varias tablas, combinación, subconjunto de otra vista, combinación de vistas y tablas

# VISTAS (cont.)

## Sintaxis VISTAS (creación)

```
create view NOMBRE_VISTA as  
    SETENCIA_SELECT  
from TABLA
```

# VISTAS (cont.)

- ✓ Cómo obtener información
- ✓ Cifrado (with encryption)
- ✓ Eliminación



# Exists & Not Exists

- ✓ Operadores para determinar si hay o no datos en una lista de valores
- ✓ Se suelen utilizar en subconsultas (interior) para restringir el resultado de la consulta (exterior).

## Sintaxis (ejemplo)

```
select cliente,numero
from facturas as f
where exists
(select *from Detalles as d
 where f.numero=d.numeroofactura
 and d.articulo='lapiz');
```

# Store Procedures

SQL ofrece dos alternativas para asegurar la integridad de los datos:

- ✓ Declarativa (constraints, defaults, rules)
- ✓ Procedimental (sp, trigger)

**SP: Conjunto de instrucciones que constituyen una unidad y que están almacenados en el servidor.**

- ✓ Encapsulan tareas repetitivas
- ✓ Tipos: **del sistema, locales, temporales, y extendidos**
- ✓ **Ventajas:** reducen tráfico entre cliente & servidor, seguridad, aúnan el acceso y las modificaciones.

# Store Procedures: Creación

- ✓ SQL analiza las instrucciones desde la sintaxis -> nombre en tabla “sysobjects”, contenido en “syscomments”. Si hay error, no se crea.
- ✓ Se pueden referenciar objetos no creados...pero deben existir al momento de ejecutar!
- ✓ Simulan conjunto de referencias a tablas, vistas, funciones definidas por el usuario.
- ✓ Pueden incluir todo tipo de instrucción **menos**: create default, create rule, etc.
- ✓ Tablas temporales y variables con duración = a la ejecución del SP.
- ✓ Ya hemos utilizado SPs (sp\_help, sp\_columns, etc.)

## Sintaxis (crear)

Create procedure **NOMBREPROCEDIMIENTO**  
As **INSTRUCCIONES**

# Store Procedures (parámetros de entrada)

- ✓ Permiten pasar info a un proc
- ✓ Se deben declarar variables como parámetros en la creación
- ✓ Son locales al procedimiento
- ✓ Se pueden declarar varios
- ✓ El valor por defecto puede ser null o una constante
- ✓ Al ejecutar el proc, los valores puede pasarme por posición o por nombre (por ej., segundo valor)

## Sintaxis

**Create procedure NOMBREPROCEDIMIENTO**

**@NOMBREPARÁMETRO TIPO =VALOR POR DEFECTO**

**As INSTRUCCIONES**

# Store Procedures (parámetros de salida)

- ✓ Se utilizan para devolver información
- ✓ Se debe declarar la variable como tipo “output”

## Sintaxis

```
create procedure NOMBREPROCEDIMIENTO  
@PARAMETROENTRADA TIPO =VALORPORDEFEECTO,  
@PARAMETROSALIDA TIPO=VALORPORDEFEECTO output  
as  
SENTENCIAS  
select @PARAMETROSALIDA=SENTENCIAS;
```

# Store Procedures (insert)

- ✓ Se puede ingresar datos en una tabla con el resultado devuelto por un SP

```
insert into TABLA exec SP_NAME
```



# Store Procedures (anidados)

- ✓ Un SP puede invocar a otro SP
- ✓ Si A llama a B, B debe existir al crear A
- ✓ B tiene acceso a todos los objetos que cree A

```
insert into TABLA exec SP_NAME
```

# Triggers

- ✓ Es un tipo de SP que se ejecuta cuando se intenta modificar datos de una tabla (o vista)
- ✓ Se definen para una tabla o vista específicas.
- ✓ Se crean por integridad y coherencia entre los datos entre distintas tablas
- ✓ Se dispara automáticamente cuando se intenta insert, update o delete
- ✓ Vs SP=no pueden ser invocados directamente
- ✓ No reciben ni retornan parametros
- ✓ Se utilizan para integridad de los datos, no para obtener resultados de consultas

## Sintaxis

```
create trigger NOMBREDISPARADOR  
on NOMBRETABLEA  
for EVENTO- insert, update o delete  
as  
SENTENCIAS
```

# Triggers

- ✓ Es un tipo de SP que se ejecuta cuando se intenta modificar datos de una tabla (o vista)
- ✓ Se definen para una tabla o vista específicas.
- ✓ Se crean por integridad y coherencia entre los datos entre distintas tablas
- ✓ Se dispara automáticamente cuando se intenta insert, update o delete
- ✓ Vs SP=no pueden ser invocados directamente
- ✓ No reciben ni retornan parametros
- ✓ Se utilizan para integridad de los datos, no para obtener resultados de consultas

## Sintaxis

```
create trigger NOMBREDISPARADOR  
on NOMBRETABLEA  
for EVENTO- insert, update o delete  
as  
SENTENCIAS
```

# Triggers (cont.) + trigger inserted

- ✓ “Create trigger” es la primer sentencia del bloque y sólo se pueden aplicar a una tabla
- ✓ Se crean solamente en la base de datos actual pero pueden hacer referencia a objetos de otras bases.
- ✓ No está permitido: create/alter/drop/load database, disk init, disk resize y otros.
- ✓ Se pueden crear varios triggers para cada evento (insert,update, delete) para una misma tabla
- ✓ Dentro del trigger, se accede a tabla virtual “inserted” (estructura=tabla en la que se define el trigger) -> guarda los valores nuevos de los registros

# Triggers: ON, Instead Of, After

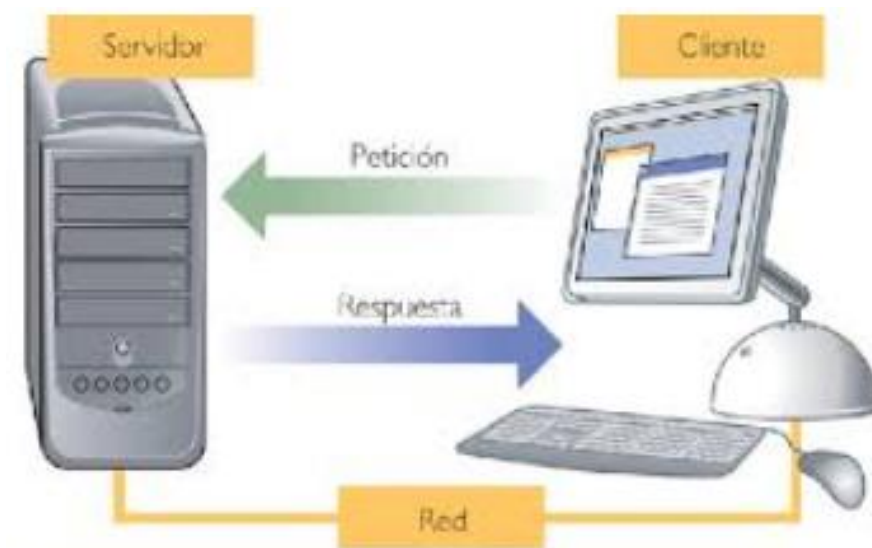
- ✓ Se puede especificar el momento del inicio del trigger
- ✓ Si no -> After (equivalente a FOR)
- ✓ Con "Instead Of" -> sólo un trigger por evento

# Modelo Cliente/Servidor

- ✓ Arquitectura donde un programa (el cliente) realiza peticiones a otro programa (el servidor) que le da respuesta
- ✓ Servidores web, fileserver, servidores de correo, de bases de datos, etc.
- ✓ La arquitectura básica es la misma en todos.

## Características de un cliente

- ✓ Quien inicia, rol activo.
- ✓ Espera y recibe las respuestas del servidor
- ✓ Por lo Gral., puede conectarse a varios servidores a la vez.
- ✓ Interactúa directamente con los usuarios (personas) a través de una interfaz gráfica (GUI)



# Características de un servidor

- ✓ Se inicia y espera a que lleguen peticiones, rol pasivo
- ✓ Recepción, procesamiento y envío de respuesta al cliente
- ✓ Aceptan gran número de conexiones concurrentes

## Ventajas

- ✓ **Escalabilidad.** Se puede aumentar la cant de clientes y servidores, en forma independiente.
- ✓ **Granularidad en Seguridad.** SQL permite administrar permisos en todos los niveles: servidor, tablas, lectura/escritura/ejecución, sp's, etc.

# Herramientas de Administración (SQL Server)

- ✓ SSMS
- ✓ SQL Server Profiler
- ✓ Asistente para la optimización de bases de datos
- ✓ Herramientas para el símbolo de sistema, como sqlcmd.exe o osql.exe
- ✓ Complementos de SQL Server Data Tools (SSDT) para MVStudio

## SQL Server Management Studio (SSMS)

- ✓ Entorno integrado de acceso a todos los componentes de SQL Server (config, admin, develop)
- ✓ Combina herramientas gráficas para desarrolladores y admins
- ✓ Combina Enterprise Manager, Query Analyzer, Analysis Manager (tools inc. en versiones anteriores)



# SQL Server Profiler

- ✓ Permite realizar seguimientos, analizarlos y reproducir resultados de aquéllos.
- ✓ Los eventos registrados se guardan en un archivo (log) de seguimiento que posteriormente se puede analizar o usar para recrear una serie de pasos específicos
- ✓ Se utilizar para diagnosticar situaciones o problemas
- ✓ <http://expressprofiler.codeplex.com/downloads/get/478141>

## Otras herramientas

- ✓ Asistente para la optimización del motor
- ✓ Sqlcmd (utility)
- ✓ osql.exe (utility)
- ✓ Reporting services
- ✓ Import/Export Data

```
Microsoft (R) SQL Server Command Line Tool
Version 9.00.3042.00 NT INTEL X86
Copyright (c) Microsoft Corporation. All rights reserved.

Note: osql does not support all features of SQL Server 2005.
Use sqlcmd instead. See SQL Server Books Online for details.

usage: osql [-U login id] [-P password]
           [-S server] [-H hostname] [-E trusted connection]
           [-d use database name] [-l login timeout] [-t query timeout]
           [-h headers] [-s colseparator] [-w columnwidth]
           [-a packet size] [-e echo input] [-I Enable Quoted Identifiers]
           [-L list servers] [-c cmdend] [-D ODBC DSN name]
           [-q "cmdline query"] [-Q "cmdline query" and exit]
           [-n remove numbering] [-n errorlevel]
           [-r msgs to stderr] [-U severitylevel]
           [-i inputfile] [-o outputfile]
           [-p print statistics] [-b On error batch abort]
           [-X[1] disable commands [and exit with warning]]
           [-O use Old ISQL behavior disables the following]
               <EOF> batch processing
               Auto console width scaling
               Wide messages
               default errorlevel is -1 vs 1
           [-? show syntax summary]
```

# Creando Bases de Datos & Archivos

Cuando planificamos una nueva base, considerar:

- ✓ El número de transacciones / unidad de tiempo
- ✓ Crecimiento potencial del almacenamiento físico
- ✓ Hardware (RAM, CPU, redundancia de discos, etc.)

## Archivos físicos de una BD

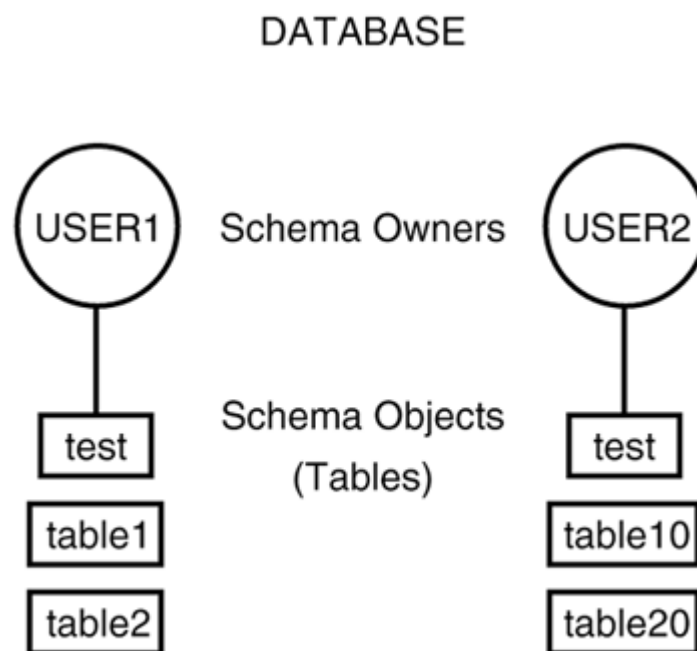
Colecciones de archivos

Tres tipos:

- ✓ Archivos de datos principales (MDF)
- ✓ Archivos de datos secundarios (NDF)
- ✓ Archivos de registro (Log/LDF)

# Schemas

- ✓ Contienen a los objetos creados en una base de datos
- ✓ Frontera/límite para el espacio de nombres para objetos
- ✓ Formato: base.schema.objeto
- ✓ Hasta 2012, espacio determinado por el nombre de usuario que lo creo (owner)



# Schema DBO & Creación

- ✓ Contendrá todo usuario que no tenga explícitamente definido un schema predeterminado
- ✓ Para creación:

```
= USE NOMBREBASE  
L
```

```
GO
```

```
CREATE SCHEMA VENTAS
```

```
GO
```

```
ALTER USER JUANLOPEZ WITH DEFAULT_SCHEMA = VENTAS
```

# Schema DBO & Creación

- ✓ Contendrá todo usuario que no tenga explícitamente definido un schema predeterminado
- ✓ Para creación:

```
USE NOMBREBASE
```

```
L
```

```
GO
```

```
CREATE SCHEMA VENTAS
```

```
GO
```

```
ALTER USER JUANLOPEZ WITH DEFAULT_SCHEMA = VENTAS
```

# Creación de Bases de Datos (transac-SQL)

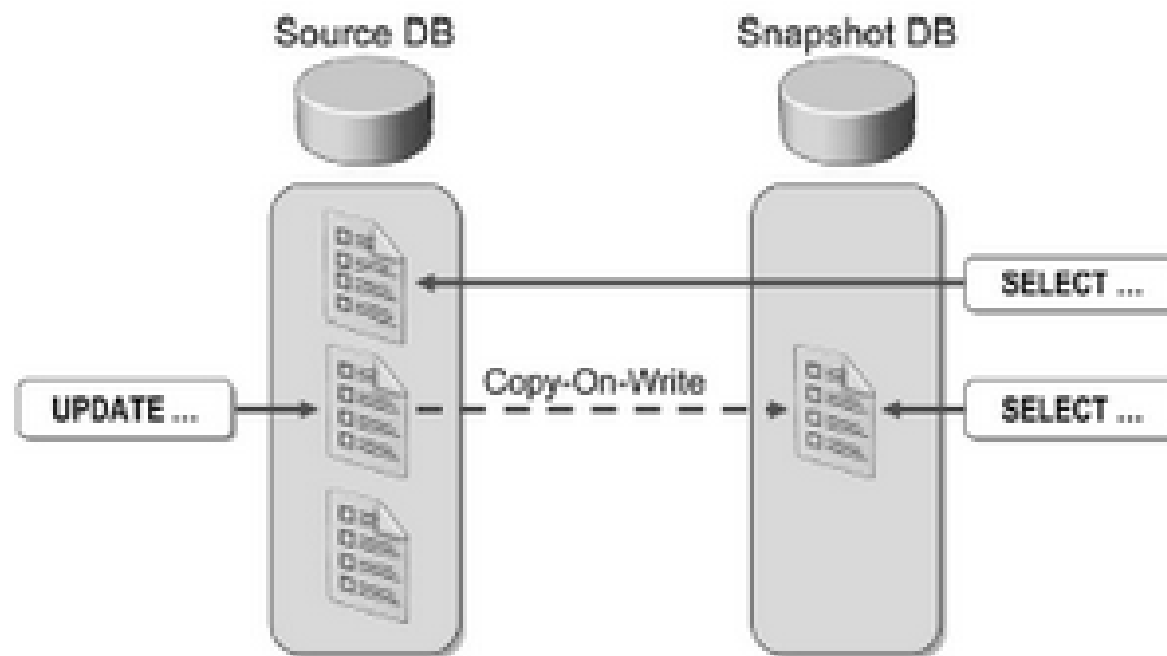
```
CREATE DATABASE TestDB
ON PRIMARY ( NAME = N'TestDB ', FILENAME = N'D:\DATA\TestDB.mdf',
SIZE = 20MB , MAXSIZE = UNLIMITED, FILEGROWTH = 10 MB )

LOG ON ( NAME = N'TestDB_log', FILENAME = N'D:\DATA\TestDB_Log.ldf',
SIZE = 5MB , MAXSIZE = 10MB , FILEGROWTH = 10MB )
GO
```

- ✓ SP\_HELPDB
- ✓ SP\_HELPFILE
- ✓ DROP DATABASE (eliminación en instancia y archivos físicos)
- ✓ COLLATION

# SnapShots

- ✓ Vistas inmediatas de sólo lectura de una base de datos
- ✓ Vista estática congelada en el tiempo
- ✓ Ahorro de tiempo y storage (vs crear una copia completa de la base)
- ✓ En un archivo de crecimiento automático.
- ✓ Con cada modificación de la base original, el snapshot recibe copia de la page original que se modifica desde que se creó el SS
- ✓ Si no, se redirige a la base principal



# Datos del Sistema

## Base de Datos MASTER

- ✓ Registra toda la información del motor:
- ✓ Existencia de las demás bases, ubicación archivos + info de inicio de SQL Server
- ✓ Cuentas de inicio de sesión, servers vinculados, configuración del sistema
- ✓ ¡No se puede iniciar SQL si esta base no está disponible!



# Datos del Sistema

## Tablas del sistema

- ✓ SQL Server utilizar el mismo gestor de base de datos para administrarse a sí mismo
- ✓ Son las tablas subyacentes que almacenan los metadatos
- ✓ Metadatos: datos que describen otros datos (análogo al uso de índices)

```
SELECT SO.NAME, SC.NAME  
FROM sys.objects SO inner join sys.columns SC  
ON SO.object_id= SC.object_id  
WHERE SO.type='U'  
ORDER BY SO.name, SC.name
```