

“Debilidades del SQL en el Procesamiento de Bases de Datos”

Oswaldo Figueroa Domejean

Debilidades de SQL en el Procesamiento de Bases de Datos

"En los Sistemas de Información Computacionales la calidad de la información en el procesamiento de datos está en función directa del software en lenguaje **SQL**, es por este motivo que es imprescindible conocer las debilidades y características muy particulares de este lenguaje para no caer en interpretaciones erróneas."

Introducción

No todos los aspectos analizados en estas diapositivas hacen referencia a debilidades del SQL, también se analizan algunos aspectos que desde la perspectiva de la teoría matemática de la Lógica son válidos pero en las consultas SQL pueden acarrearnos interpretaciones que no condicen con la realidad en el contexto de Bases de Datos.

Todos los ejemplos mencionados en este trabajo han sido comprobados en el lenguaje SQL del Sistema de Administración de Base de Datos MySQL y DB2, la situación es la misma y en algunos casos aún peor en otros DBMS SQL.

Bases de Datos “Tarjetas de Débito”

CLIENTE			CUENTA					
CLI	NOMBRE	CIUDAD	CTA	CLI	BCO			
1	Pérez	Madrid	8123	2	BB			
2	Smith	Nueva York	4551	2	BA			
3	Bonner	Paris	3212	3	BC			
4	Chen	Hong Kong	3529	3	BC			
			4254	1	BB			
			4572	1	BA			
BANCO			DEPOSITO					
BCO	BANCO		DEP	CTA	FECHA			
BA	Banco A		d1	8123	12/05/2005			
BB	Banco B		d2	4551	17/05/2005			
BC	Banco C		d3	4254	21/05/2005			
ATM			ATM	CIUDAD	BCO	DEP	CTA	FECHA
ATM	CIUDAD	BCO	DEP	CTA	FECHA			
a1	Paris	BA	d4	3212	25/05/2005			
a2	Roma	BB	d5	3529	27/05/2005			
a3	Madrid	BA	d6	3212	12/06/2005			
a4	Rio	BB	d7	4254	15/06/2005			
a5	Paris	BC						
RETIRO			RETIRO					
RET	CTA	FECHA	RET	CTA	FECHA			
r1	8123	25/05/2005	r1	8123	25/05/2005			
r2	4254	25/05/2005	r2	4254	25/05/2005			
r3	4551	05/06/2005	r3	4551	05/06/2005			
r4	3529	10/06/2005	r4	3529	10/06/2005			
r5	8123	12/06/2005	r5	8123	12/06/2005			
r6	4254	12/06/2005	r6	4254	12/06/2005			
r7	3529	15/06/2005	r7	3529	15/06/2005			
r8	3212	17/06/2005	r8	3212	17/06/2005			
r9	4551	20/06/2005	r9	4551	20/06/2005			

La Interpretación de la marca *null*

La marca *null* en SQL utilizamos para representar el hecho de que en un momento dado no se cuenta con el dato para almacenarlo en un atributo de una Variable de Relación de una Base de Datos.

E. F. Codd fue quien planteó inicialmente la marca *null* como medio para representar la ausencia de información.

- “Información Faltante”, la marca *null* del SQL
- “Información Inaplicable”, no tiene una representación en SQL

El problema: En la **operación de Igualdad**, un *null* no es igual a otro, tiene sentido, pero en el caso de **eliminación de duplicados** un *null* es igual a otro.

La Interpretación de la marca *null*

La inclusión de la marca *null* como otro valor de verdad en una lógica de tres valores VERDADERO, FALSO y *unknown*.

$$x > y \text{ OR } x < y \text{ OR } x = y$$

Consecuencias en el Álgebra Relacional:

AA		AB	
a	b	a	b
a ₁	<i>null</i>	a ₁	<i>null</i>
<i>null</i>	b ₁		
AA INTERSECT BB			
a	b		
a ₁	<i>null</i>		

Con el lenguaje SQL no obtenemos la misma respuesta.

Alternativa: el operador MAYBE ???

La Interpretación de la marca *null*

Diversos significados de la marca *null*

Ejemplo:

RETIRO				
RET	CTA	FECHA	MONTO	ATM
r1	8123	25/05/2005	300	a1
r2	4254	25/05/2005	400	<i>null</i>
r3	4551	05/06/2005	300	<i>null</i>
r4	3529	10/06/2005	250	a2

Tres tipos de retiro de fondos:

- Retiro asociado a un ATM, ‘r1’.
- Compra de un producto o servicio, ‘r3’
- Transferencia de fondos, el código de retiro ‘r2’ y el depósito ‘d4’

La Interpretación de la marca *null*

Segunda y tercera no corresponden a “Información Faltante” sí las podríamos considerar “Información inaplicable”.

Si ‘r4’ tuviera en el valor de ATM *null*, esto es “Información Faltante” se amplifica la ambigüedad.

Se observan al menos dos significados diferentes de la marca *null*

- “Información Faltante”
- “Información Inaplicable”
- y posiblemente otros . . .

La Interpretación de la marca *null*

Una interpretación errónea.

“Obtenga los códigos de cuentas con depósitos mayores a 1000 para el Banco con código ‘BB’”

CUENTA			DEPOSITO				
CTA	CLI	BCO	DEP	CTA	. . .	MONTO	. . .
8123	2	BB	d1	8123		2000	
4551	2	BA	d2	4551		3000	
4254	1	<i>null</i>	d3	4254		2500	
			d7	4254		150	

```
SELECT B.CTA, SUM(B.MONTO) AS TOTAL  
FROM CUENTA A, DEPOSITO B  
WHERE A.CTA = B.CTA  
      AND A.BCO = 'BB'  
GROUP BY B.CTA  
HAVING SUM(B.MONTO) > 1000
```

Resultado (8123, 2000)

La Interpretación de la marca *null*

Ahora si el código de Banco BCO para la cuenta ‘4254’ fuera ‘BC’, el resultado es FALSO. Una sola respuesta en dos situaciones diferentes:

“Se desconoce si existe alguna cuenta de cliente que tenga el total de sus depósitos en cantidades mayores a 1000 del Banco con código ‘BB’ ”

“Definitivamente no se tiene cuentas de clientes que tengan el total de sus depósitos mayores a 1000 del Banco con código ‘BB’ ”

La Interpretación de la marca *null*

La utilización de valores especiales

Alternativa a la utilización de la marca *null* ???

Lógica de dos valores VERDAD y FALSO, pero . . .

“Obtenga las cuentas y el monto de incentivo para los clientes que tienen depósitos mayores a 1000”

DEPOSITO					
DEP	CTA	. . .	MONTO	RET	INTERES
d3	4254		2500	<i>null</i>	0,2
d5	3529		1500	<i>null</i>	0

```
SELECT CTA, (MONTO * INTERES) AS TOTAL  
FROM DEPOSITO  
WHERE MONTO > 1000
```

Resultado:

CTA	TOTAL
4254	50
3529	0

Los Operadores del Álgebra Relacional

Operaciones definidas para el manejo de relaciones:

OPERACIONES	
Unión	Restricción
Intersección	Proyección
Diferencia	Reunión
Producto Cartesiano	División

Dos alternativas de elaboración de consultas:

- 1ra. Consultas en una forma tradicional.
- 2da. Utilizando los operadores del Álgebra Relacional en SQL.

Ejemplo:

RA		RB	
CLI	BCO	CLI	BCO
1	BB	3	BC
2	<i>null</i>	<i>null</i>	BA
<i>null</i>	BA	2	<i>null</i>
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
3	BC		

Los Operadores del Álgebra Relacional

Operador de Unión

La única posibilidad el operador UNION del SQL:

```
SELECT CLI, BCO FROM RA  
UNION  
SELECT CLI, BCO FROM RB
```

Resultado:

CLI	BCO
<i>null</i>	BA
1	BB
3	BC
2	<i>null</i>
<i>null</i>	<i>null</i>

Nótese que el SQL considera a la tupla $(2, \text{null})$ en ambas Variables de Relación como iguales!.
En el caso de valores especiales la historia sería la misma.

Los Operadores del Álgebra Relacional

Operador de Intersección

La operación `INTERSECT` del Álgebra Relacional en el SQL sí tiene su correspondiente operación en el SQL tradicional utilizando la reunión (i.e. `JOIN`)

```
SELECT CLI, BCO FROM RA  
INTERSECT  
SELECT CLI, BCO FROM RB
```

Resultado:

CLI	BCO
null	BA
3	BC
2	null
null	null

La intersección considera iguales a las tuplas $(2, \text{ null})$

Los Operadores del Álgebra Relacional

Utilizando la equivalencia con el SQL tradicional, esperaríamos obtener el mismo resultado:

```
SELECT RA.CLI, RB.BCO  
FROM RA, RB  
WHERE RA.CLI = RB.CLI  
AND RA.BCO = RB.BCO
```

Resultado:

CLI	BCO
3	BC

Nótese el hecho de que este resultado intersecta solamente lo que realmente “conoce” la Base de Datos, y no así la marca *null*, para la cual no se conoce en ese momento su valor.

En caso de usar valores especiales el resultado sería el mismo que en el caso de la utilización del operador `INTERSECT`.

Los Operadores del Álgebra Relacional

Operador de Diferencia

La operación `MINUS` del Álgebra Relacional en SQL sí tiene algunas equivalencias de planteamiento en el SQL tradicional, una de las alternativas es la utilización de la condición `EXISTS`.

```
SELECT CLI, BCO FROM RA  
MINUS  
SELECT CLI, BCO FROM RB
```

Resultado:

CLI	BCO
1	BB

Nótese que la operación de diferencia considera que la tupla $(2, \text{null})$ es la misma en las dos Variables de Relación, cosa que no necesariamente es correcta ya que la marca *null* puede tener un valor diferente en cada una de las tuplas en cuestión.

Los Operadores del Álgebra Relacional

Utilizando la equivalencia del operador de diferencia con la condición EXISTS del SQL tradicional, esperaríamos obtener el mismo resultado:

```
SELECT CLI, BCO
FROM RA
WHERE NOT EXISTS (SELECT * FROM RB
                    WHERE RB.CLI = RA.CLI
                    AND RB.BCO = RA.BCO)
```

Resultado:

CLI	BCO
1	BB
2	<i>null</i>
<i>null</i>	BA
<i>null</i>	<i>null</i>

El resultado considera diferentes a la tupla $(2, \text{null})$ lo cual es correcto, este resultado se enmarca en el contexto de funcionamiento del EXISTS.

Los Operadores del Álgebra Relacional

Operador de División

Esta operación del Álgebra Relacional no tiene un operador específico en el SQL tradicional, es por este motivo que la División en el SQL se plantea sobre la base de la siguiente equivalencia:

$$\text{FORALL } V \ (p(V)) \equiv \text{NOT}(\text{EXISTS } V \ (\text{NOT } p(V)))$$

con:

V: Variable

p(V): Función con valor de verdad

El Cuantificador Universal planteado como una doble negación utilizando el Cuantificador Existencial

Los Operadores del Álgebra Relacional

“Obtenga los códigos de clientes que tienen cuentas en todos los Bancos”

RG			RH
CLI	BCO		BCO
1	BA		BA
1	BB		BB
1	BC		BC
2	BB		null
2	BC		
2	BD		
3	BA		
3	BB		
1	null		

```
SELECT DISTINCT X.CLI
  FROM RG X
 WHERE NOT EXISTS (SELECT *
    FROM RH Y
 WHERE NOT EXISTS (SELECT *
      FROM RG
 WHERE RG.BCO = Y.BCO
 AND RG.CLI = X.CLI))
```

El resultado es vacío por las tuplas con la marca *null*, lo correcto sería “desconocido”.

Existen otras posibilidades en la respuesta dependiendo de las inclusión o no de la marca *null* y de los valores especiales.

Los Operadores del Álgebra Relacional

Equivalencia de la Intersección con la doble Diferencia

La operación de intersección puede ser expresada en forma equivalente como la doble diferencia de las Variables de Relación involucradas, es decir:

A INTERSECT B

es equivalente a las expresiones:

A MINUS (A MINUS B)
B MINUS (B MINUS A)

```
SELECT CLI, BCO FROM RA  
INTERSECT  
SELECT CLI, BCO FROM RB
```

Resultado:

CLI	BCO
null	BA
3	BC
2	null
null	null

Los Operadores del Álgebra Relacional

Iniciamos el análisis utilizando el operador MINUS – EXCEPT.
Creamos una vista VA de la diferencia entre RA y RB:

```
CREATE VIEW VA AS  
SELECT CLI, BCO FROM RA  
MINUS  
SELECT CLI, BCO FROM RB
```

Resultado:

CLI	BCO
1	BB

luego realizamos la resta correspondiente entre RA y la vista VA:

```
SELECT CLI, BCO FROM RA  
MINUS  
SELECT CLI, BCO FROM VA
```

El resultado es el mismo que utilizando el operador INTERSECT
Se comprueba también la segunda equivalencia.

Los Operadores del Álgebra Relacional

Equivalencia de la División con la Diferencia y el Producto Cartesiano

La operación de división puede ser expresada en forma equivalente utilizando otros operadores del Álgebra Relacional, la expresión:

$A[x, y] \text{ DIVIDED BY } B[y]$

es equivalente a:

$A[x] \text{ MINUS } ((A[x] \text{ TIMES } B[y]) \text{ MINUS } A[x, y]) [x]$

RG			RH
CLI	BCO		BCO
1	BA		BA
1	BB		BB
1	BC		BC
2	BB		
2	BC		
2	BD		
3	BA		
3	BB		

La expresión para este ejemplo:

$RG[CLI] \text{ MINUS } ((RG[CLI] \text{ TIMES } RH[BCO]) \text{ MINUS } RG[CLI, BCO]) [CLI]$

Los Operadores del Álgebra Relacional

Tomamos la primera expresión como el Producto Cartesiano entre RG y RH, para lo cual creamos la vista VA :

```
CREATE VIEW VA AS  
SELECT RG.CLI, RH.BCO  
FROM RG, RH
```

Luego efectuamos la diferencia entre la vista VA y la Variable de Relación RG con todos sus atributos, para lo cual creamos la vista resultado VB.

```
CREATE VIEW VB AS  
SELECT CLI, BCO  
FROM VA  
WHERE NOT EXISTS (SELECT *  
                   FROM RG  
                   WHERE RG.CLI = VA.CLI  
                   AND RG.BCO = VA.BCO)
```

Los Operadores del Álgebra Relacional

Por último realizamos la diferencia entre RG y la vista VB , por lo tanto tenemos:

```
SELECT DISTINCT CLI  
FROM RG  
WHERE NOT EXISTS (SELECT *  
                   FROM VB  
                   WHERE VB.CLI = RG.CLI)
```

El resultado de esta consulta es el código de cliente ‘1’ lo cual es correcto. Nótese que en las Variables de Relación utilizadas no tenemos la marca *null*.

Las Condiciones EXISTS, IN, ANY y ALL

Los Cuantificadores Existencial y Universal en el contexto Relacional

Dada una Variable V con elementos v_1, v_2, \dots, v_n , y una función con valor de verdad $p(V)$:

Para EXISTS tenemos:

$$\text{EXISTS } (V p(V)) \equiv \text{FALSO OR } p(v_1) \text{ OR } p(v_2) \text{ OR } \dots \text{ OR } p(v_n)$$

Para FORALL tenemos:

$$\text{FORALL } (V p(V)) \equiv \text{VERDAD AND } p(v_1) \text{ AND } p(v_2) \text{ AND } \dots \text{ AND } p(v_n)$$

Si V vacía, el resultado es FALSO y VERDAD respectivamente

Las Condiciones EXISTS, IN, ANY y ALL

Correspondencia entre los Cuantificadores Existencial y Universal en SQL

$$\text{EXISTS } V \ (p(V)) \equiv \text{NOT } (\text{FORALL } V \ (\text{NOT } (p(V)))) \quad (1)$$

$$\text{FORALL } V \ (p(V)) \equiv \text{NOT } (\text{EXISTS } V \ (\text{NOT } (p(V)))) \quad (2)$$

Sobre la base de estas correspondencias se analizan las condiciones EXISTS, IN, ANY y ALL del SQL.
En el caso de la condición IN tenemos:

$V \text{ IN } (\text{conjunto})$

Esta expresión la podemos considerar una forma particular del Cuantificador Existencial

Las Condiciones EXISTS, IN, ANY y ALL

En la parte derecha de (1) tenemos una doble negación que incluye a la condición FORALL, observamos que $p(V)$ corresponde a la expresión en cuestión, reemplazando:

$$\text{NOT } (\text{FORALL } V \text{ (NOT } (V \text{ IN } (\text{conjunto}))))$$

La expresión dentro del paréntesis externo tiene la siguiente correspondencia con la condición ALL:

$$\text{FORALL } V \text{ (NOT } (V \text{ IN } (\text{conjunto}))) \equiv V \neq \text{ALL } (\text{conjunto})$$

Luego aplicamos la negación correspondiente con lo cual tenemos la correspondencia completa:

$$V \text{ IN } (\text{conjunto}) \equiv \text{NOT } (V \neq \text{ALL } (\text{conjunto}))$$

Las Condiciones EXISTS, IN, ANY y ALL

Claro está que en sentido contrario también se cumple la correspondencia la condición ALL en términos de la condición ANY:

$v > \text{ALL } (\text{conjunto})$

En la parte derecha de la expresión (2) tenemos una doble negación que incluye a la condición EXISTS, observamos que $p(v)$ corresponde a la expresión en cuestión, reemplazando:

$\text{NOT } (\text{EXISTS } v \text{ (NOT } (v > \text{ALL } (\text{conjunto}))))$

La expresión dentro del paréntesis externo tiene la siguiente correspondencia con la condición ANY:

$\text{EXISTS } v \text{ (NOT } (v > \text{ALL } (\text{conjunto}))) \equiv v \leq \text{ANY } (\text{conjunto})$

Las Condiciones EXISTS, IN, ANY y ALL

Luego aplicamos la negación respectiva con lo cual tenemos la correspondencia completa:

$$v > \text{ALL } (\text{conjunto}) \equiv \text{NOT } (v \leq \text{ANY } (\text{conjunto}))$$

Este aspecto es de gran relevancia en la elaboración de consultas PERO debemos tener cuidado en la utilización de las condiciones cuando tenemos la marca *null*, el conjunto vacío o los valores especiales de por medio.

Las Condiciones EXISTS, IN, ANY y ALL

Condiciones NOT IN (null), NOT (ANY (null))
y NOT EXISTS (Φ)

“Obtenga los nombres de Bancos que no tienen ATMs en la ciudad de Madrid”

BANCO		ATM		
BCO	BANCO	ATM	CIUDAD	BCO
BA	Banco A	a1	París	BA
BB	Banco B	a2	Roma	BB
BC	Banco C	a3	Madrid	null
		a4	null	BB
		a5	París	BC

Las Condiciones EXISTS, IN, ANY y ALL

Tenemos estas dos consultas que son equivalentes:

```
SELECT A.BANCO  
FROM BANCO A  
WHERE A.BCO NOT IN (SELECT B.BCO  
                     FROM ATM B  
                     WHERE CIUDAD = 'Madrid')
```

```
SELECT A.BANCO  
FROM BANCO A  
WHERE NOT (A.BCO = ANY (SELECT B.BCO  
                         FROM ATM B  
                         WHERE CIUDAD = 'Madrid'))
```

El resultado de estas dos consultas es vacío. Una consulta equivalente a las dos anteriores utilizando la condición EXISTS:

```
SELECT A.BANCO  
FROM BANCO A  
WHERE NOT EXISTS (SELECT *  
                  FROM ATM B  
                  WHERE CIUDAD = 'Madrid'  
                  AND A.BCO = B.BCO)
```

El resultado de esta consulta es una lista con todos los nombres de Bancos

Las Condiciones EXISTS, IN, ANY y ALL

Condición EXISTS (*null*)

“Obtenga los códigos de clientes del Banco ‘BB’, los cuales han hecho retiros en ATMs de montos mayores a 300”

CUENTA			RETIRO				
CTA	CLI	BCO	RET	CTA	. . .	MONTO	ATM
8123	2	BB	r1	8123		300	a1
3212	3	BC	r5	8123		450	<i>null</i>
			r8	3212		200	a3

```
SELECT A.CLI  
FROM CUENTA A  
WHERE A.BCO = 'BB'  
AND EXISTS (SELECT B.ATM  
            FROM RETIRO B  
            WHERE A.CTA = B.CTA  
            AND B.MONTO > 300)
```

Resultado:

CLI
2

Las Condiciones EXISTS, IN, ANY y ALL

Condición ALL ($e_1, e_2, \dots, null, \dots, e_n$) y ALL (Φ)

ALL ($e_1, e_2, \dots, null, \dots, e_n$) genera FALSO

Incorrecto debería ser desconocido

“Obtenga los códigos de cuentas en las cuales se realizan depósitos mayores a todos los depósitos realizados en cuentas del Banco ‘BC’”

CUENTA			DEPOSITO				
CTA	CLI	BCO	DEP	CTA	. . .	MONTO	. . .
8123	2	BB	d1	8123		2000	
4551	2	BA	d2	4551		3000	
3212	3	BC	d3	4254		2500	
3529	3	BC	d4	3212		400	
4572	1	BA	d5	3529		null	
			d6	3212		450	

Las Condiciones EXISTS, IN, ANY y ALL

```
SELECT DISTINCT A.CTA
FROM DEPOSITO A
WHERE A.MONTO > ALL (SELECT B.MONTO
                      FROM DEPOSITO B, CUENTA C
                      WHERE C.CTA = B.CTA
                      AND C.BCO = 'BC')
```

El resultado de esta consulta es vacío, la subconsulta genera el conjunto (400, null, 450), para el cual no existe un solo monto de depósito que sea mayor a todos los elementos de este conjunto por lo tanto la subconsulta genera FALSO

Las Condiciones EXISTS, IN, ANY y ALL

Cuando el conjunto asociado a la condición ALL es vacío, el resultado de la subconsulta es precisamente VERDAD.

```
SELECT DISTINCT A.CTA
FROM DEPOSITO A
WHERE A.MONTO > ALL (SELECT B.MONTO
                      FROM DEPOSITO B, CUENTA C
                      WHERE C.CTA = B.CTA
                      AND C.BCO = 'XYZ')
```

El Banco con código ‘XYZ’ no existe, entonces el resultado de la subconsulta es vacío, por este hecho el resultado es VERDAD y el resultado total corresponde a las cuentas en las cuales se realizan depósitos mayores a todos los depósitos realizados en cuentas del Banco ‘XYZ’ !

Las Condiciones **EXISTS**, **IN**, **ANY** y **ALL**

La subconsulta puede ser inclusive una aseveración absurda, mientras genere vacío la expresión de la subconsulta siempre será VERDAD.

Este no es un error desde la perspectiva de la Lógica, es una consecuencia de la definición de la condición ALL.

Los Operadores de Agregación

En el caso en que `SUM (conjunto)` tenga como único elemento al conjunto vacío, el resultado es la marca *null* y no así cero como podríamos esperar.

“Obtenga los códigos de depósitos para clientes que tienen un total de retiros menor a 300”

DEPOSITO					RETIRO				
DEP	CTA	. . .	MONTO	. . .	RET	CTA	. . .	MONTO	. . .
d2	4551		3000		r2	4254		100	
d3	4254		2500		r4	3529		250	
d5	3529		1500		r6	4254		50	
d7	4254		150		r7	3529		150	

Los Operadores de Agregación

```
SELECT DISTINCT A.DEP  
FROM DEPOSITO A  
WHERE (SELECT SUM (B.MONTO)  
       FROM RETIRO B  
      WHERE A.CTA = B.CTA) < 360
```

Resultado:

DEP
d4

El depósito de la cuenta ‘4551’ no tiene ningún retiro de fondos.

El operador SUM de la subconsulta tiene como elemento al conjunto vacío.

La respuesta del SUM es la marca *null*, y no cero como podríamos esperar.

Los Operadores de Agregación

El operador AVG()

“Obtenga el promedio de retiros por cada cuenta de cliente”

RETIRO				
RET	CTA	...	MONTO	...
r1	8123		300	
r2	4254		400	
r5	8123		450	
r6	4254		null	
r9	4551		500	
r10	4254		100	

```
SELECT CTA, AVG(MONTO) AS PROMEDIO  
FROM RETIRO  
GROUP BY CTA
```

Resultado:

CTA	PROMEDIO
8123	375
4254	250
4551	500

El promedio para ‘4254’ es 150, calculado sobre los valores (200, null, 100).

Si utilizamos cero como valor especial, el promedio para ‘4254’ Es 100 calculado sobre la base de (200, 0, 100)

Los Operadores de Agregación

El operador COUNT () frente al SUM ()

Cuando el operador COUNT (conjunto) tiene como único elemento al conjunto vacío genera como resultado cero, en cambio en los casos de SUM, AVG, MIN y MAX en estas mismas condiciones generan la marca *null*.

“Obtener las cuentas y el total de depósitos mayores a 1500 de clientes que tienen un total de retiros menores a 300”

DEPOSITO					RETIRO				
DEP	CTA	...	MONTO	...	RET	CTA	...	MONTO	...
d2	4551		3000		r2	4254		100	
d3	4254		2500		r4	3529		250	
d5	3529		1500		r6	4254		50	
d7	4254		150		r7	3529		150	

Los Operadores de Agregación

```
SELECT DISTINCT A.CTA, SUM(A.MONTO) AS DEPOSITO  
FROM DEPOSITO A  
WHERE (SELECT SUM(B.MONTO)  
       FROM RETIRO B  
      WHERE A.CTA = B.CTA) < 700  
GROUP BY A.CTA  
HAVING SUM(A.MONTO) > 1500
```

Resultado:

CTA	DEPOSITO
4254	2650

El depósito de la cuenta ‘4551’ no tiene ningún retiro asociado.

El operador SUM de la subconsulta tiene como elemento al conjunto vacío, por este motivo genera *null* y no se incluye en el resultado.

Los Operadores de Agregación

Ahora quisiéramos obtener las cuentas y el total de depósitos que tienen un número de retiros menor a tres.

Ajustamos la consulta cambiando el operador SUM por el COUNT y la condición.

```
SELECT DISTINCT A.CTA, SUM(A.MONTO) AS DEPOSITO  
FROM DEPOSITO A  
WHERE (SELECT COUNT(B.MONTO)  
      FROM RETIRO B  
     WHERE A.CTA = B.CTA) < 3  
GROUP BY A.CTA  
HAVING SUM(A.MONTO) > 1500
```

Resultado:

CTA	DEPOSITO
4254	2650
4551	3000

Aparece la cuenta ‘4551’ que solamente tenemos en DEPOSITO.

La Implicación

Las tres alternativas lógicamente equivalentes de la Implicación son:

$$P \Rightarrow Q \equiv \sim P \text{ OR } Q \equiv \text{SI } P \text{ ENTONCES } Q$$

TABLA DE VERDAD DE LA IMPLICACIÓN

Nro	P	Q	$\sim P$	Q	$\sim P \text{ OR } Q$
1	V	V	F	V	V
2	V	F	F	F	F
3	F	V	V	V	V
4	F	F	V	F	V

En las filas 1 y 2, cuando el antecedente es VERDAD el resultado de toda la expresión es el mismo que el consecuente.

En la fila tres partimos de una premisa falsa en el antecedente y llegamos a VERDAD en la expresión completa.

En la fila cuatro partimos de un antecedente y un consecuente falsos para llegar al resultado VERDAD.

La Implicación

“Obtenga los códigos de clientes que si tienen transferencias de dinero del ATM ‘a3’ la cuenta del Banco tiene el código ‘BA’ ”

SI la transferencia de fondos tiene el código de ATM ‘a3’
ENTONCES la cuenta pertenece al Banco ‘BA’

Ahora, haciendo la transformación, tenemos:

```
NOT      (DEPOSITO.RET = RETIRO.RET AND RETIRO.ATM = 'a3')
OR       (CUENTA.CTA = RETIRO.CTA AND CUENTA.BCO = 'BA')
```

Como vemos el antecedente:

```
(DEPOSITO.RET = RETIRO.RET AND RETIRO.ATM = 'a3')
```

es falso. En cambio el consecuente:

La Implicación

(CUENTA.CTA = RETIRO.CTA AND CUENTA.BCO = 'BA')

es verdad, estamos en el caso de la fila tres de la Tabla. Con estos datos de ejemplo:

CUENTA			DEPOSITO					
CTA	CLI	BCO	DEP	CTA	. . .	CLI	MONTO	RET
8123	2	BB	d1	8123		2	2000	null
4551	2	BA	d2	4551		1	3000	null
3212	3	BC	d3	4254		1	2500	null
3529	3	BC	d4	3212		1	400	r2
4254	1	BB	d5	3529		4	1500	null
4572	1	BA	d6	3212		2	450	r5
			d7	4254		3	150	r7

RETIRO				
RET	CTA	. . .	MONTO	ATM
r1	8123		300	a1
r2	4254		400	null
r3	4551		300	null
r4	3529		250	a2
r5	8123		450	null
r6	4254		200	null
r7	3529		150	null
r8	3212		200	a3
r9	4551		500	a4

La Implicación

```
SELECT DISTINCT C.CLI  
FROM DEPOSITO A, RETIRO B, CUENTA C  
WHERE NOT (A.RET = B.RET AND B.ATM = 'a3')  
OR (C.CTA = B.CTA AND C.BCO = 'BA')
```

Resultado:

CLI
1
2
3

Este es un caso lógicamente válido que puede acarrear interpretaciones erróneas.

En un ejemplo relacionado a la fila cuatro tenemos una situación peor todavía ya que tanto el antecedente como el consecuente son falsos.

“Obtenga los códigos de clientes que si tienen transferencias de dinero del ATM ‘a3’ los montos de la misma son diferentes”

La Implicación

SI la transferencia de fondos tiene el código de ATM 'a3'
ENTONCES los montos de esta son diferentes

Ahora, haciendo la transformación, tenemos:

```
NOT      (DEPOSITO.RET = RETIRO.RET AND RETIRO.ATM = 'a3')
OR       (RETIRO.CTA = CUENTA.CTA AND DEPOSITO.MONTO <> RETIRO.MONTO)
```

Como vemos tanto el antecedente como el consecuente son falsos

```
SELECT DISTINCT C.CLI
FROM DEPOSITO A, RETIRO B, CUENTA C
WHERE NOT (A.RET = B.RET AND B.ATM = 'a3')
        OR (B.CTA = C.CTA AND A.MONTO <> B.MONTO)
```

El resultado que obtenemos es exactamente el mismo que en el caso anterior.

Los Operadores de Actualización

En este caso el problema es conflictivo ya que son operadores que cambian valores en Variables de Relación.

El operador **INSERT**

Incluimos dos Variables de Relación temporales TEMP1 y TEMP2, considerando que estas corresponden a tablas asociadas a una migración de datos hacia la Base de Datos de ejemplo.

“Insertar en la Variable de Relación ATM los ATMs tal que el Banco asociado no esté en la ciudad de ‘Montevideo’”

Los Operadores de Actualización

TEMP1			TEMP2	
ATM	CIUDAD	BCO	CIUDAD	BCO
a6	Quito	BB	Roma	BB
a7	Viena	BA	Montevideo	null

Tenemos dos consulta equivalentes:

```
INSERT INTO ATM (ATM, CIUDAD, BCO)
(SELECT A.ATM, A.CIUDAD, A.BCO
FROM TEMP1 A
WHERE A.BCO NOT IN (SELECT B.BCO
                     FROM TEMP2 B
                     WHERE B.CIUDAD = 'Montevideo'));
```

No inserta ninguna tupla en la Variable de Relación ATM.

```
INSERT INTO ATM (ATM, CIUDAD, BCO)
(SELECT A.ATM, A.CIUDAD, A.BCO
FROM TEMP1 A
WHERE NOT EXISTS (SELECT *
                   FROM TEMP2 B
                   WHERE A.BCO = B.BCO
                   AND B.CIUDAD = 'Montevideo'))
```

Sí inserta en la Variable de Relación ATM las dos tuplas de TEMP1.

Los Operadores de Actualización

El operador UPDATE

“Incrementar en un 2.5 % los montos individuales de depósitos para los cuales el monto depositado es mayor a todos los montos del cliente con código de cuenta ‘8123’”.

DEPOSITO				
DEP	CTA	. . .	MONTO	. . .
d1	8123		2000	
d2	4551		3000	
d3	4254		2500	
d4	3212		400	
d5	3529		1500	
d8	8123		2200	

```
UPDATE DEPOSITO  
SET MONTO = MONTO * 1.025  
WHERE MONTO > ALL (SELECT A.MONTO  
                     FROM DEPOSITO A  
                     WHERE A.CTA = 8123)
```

Las tuplas con códigos de depósito ‘d2’ y ‘d3’ obtienen un incremento de 2.5% en sus montos, hasta aquí no tenemos problema.

Los Operadores de Actualización

Ahora que sucede si utilizamos en la condición de la subconsulta una cuenta inexistente como ‘1111’:

```
UPDATE DEPOSITO  
SET MONTO = MONTO * 1.025  
WHERE MONTO > ALL (SELECT A.MONTO  
                     FROM DEPOSITO A  
                     WHERE A.CTA = 1111)
```

La subconsulta genera vacío como resultado, lo cual genera un incremento de 2.5%, pero para todos! los montos de DEPÓSITO.

Los Operadores de Actualización

En la siguiente expresión que es equivalente a la anterior utilizando el operador MAX:

```
UPDATE DEPOSITO  
SET MONTO = MONTO * 1.025  
WHERE MONTO > (SELECT MAX(A.MONTO)  
                  FROM DEPOSITO A  
                  WHERE A.CTA = 1111);
```

La subconsulta resulta en la marca *null*, por ese motivo la expresión de cambio no realiza ningún incremento del 2.5% para ninguna de las tuplas de DEPOSITO.

Conclusiones

- Es muy relevante notar que los DBMS SQL actuales cuentan con soporte para el manejo de la marca *null* para representar el hecho de falta de información en un momento dado. Hemos visto que no solamente la falta de información es necesaria registrar en un valor de atributo de una Variable de Relación, existen otras posibilidades que en definitiva pueden causarnos una interpretación errónea en el procesamiento de datos.
- Ahora la utilización de valores especiales en vez de la marca *null* aminora los problemas pero tampoco son una solución por la falta de soporte completo que nos ofrecen los DBMS SQL actuales.
- En definitiva debemos estar conscientes de los problemas que nos puede acarrear estos tipos de información, que necesariamente se presentan en el registro de datos en una Base de Datos, para no caer en interpretaciones erróneas a respuestas de consultas que hagamos a una Base de Datos.

Conclusiones

- No existen las equivalencias que podríamos pensar entre el SQL tradicional y el SQL utilizando operadores del Álgebra Relacional, estos problemas principalmente surgen a causa de la utilización de la marca *null*, ahora, tampoco la utilización de valores especiales nos resuelve el problema, en ambos casos deberíamos considerarlos en las condiciones de las consultas para disminuir la probabilidad de interpretaciones erróneas en consultas a una Base de Datos.
- A pesar de que las condiciones EXISTS, IN y ANY representan de alguna forma a la Cuantificación Existencial de la Lógica hemos visto que surgen una serie de problemas cuando está de por medio la marca *null*, el conjunto vacío o los valores especiales, a causa de esto no existe la equivalencia completa entre la perspectiva del SQL y la Lógica.

Conclusiones

- El problema es mayor cuando se trata de la condición ALL, en este caso más allá de buscar una equivalencia con la Cuantificación Universal notamos que el comportamiento de esta condición desde la perspectiva de la Lógica es válido, sin embargo nos puede causar confusión en respuestas de consultas a Bases de Datos.
- Los cálculos numéricos no funcionan en forma correcta cuando de por medio tenemos a la marca *null*, el conjunto vacío o los valores especiales. Otro aspecto que notamos es la falta de simetría entre los operadores SUM, MIN, MAX y AVG, frente al operador COUNT cuando tenemos de por medio al conjunto vacío.
- En el caso de la utilización de valores especiales el problema, si no es igual es mayor al mencionado ya que los DBMS SQL no hacen ninguna diferencia con otros tipos de valores.
- En definitiva los operadores mencionados no se comportan como podríamos esperar en los casos mencionados, recibimos respuestas que definitivamente afectan la exactitud que requerimos, y debe existir, al tiempo de procesar valores numéricos, nuevamente debemos estar conscientes del problema para no caer en interpretaciones erróneas.