De AR hacia SQL, y de SQL hacia AR

Estos ejercicios suelen aparecer en los recuperatorios de los parciales. No se explican en clase y se aprenden en general cuando uno le pregunta al que corrigió cómo se hacen. Quizá lo más objetable sea que estos ejercicios "califican" el parcial: hay una cuentita que genera la nota final, donde estos ejercicios de no estar hechos correctamente, condicionan completamente la aprobación.

Sea:

Relr = R(A,B,C,D)
Rels = R(A,B,X,S)
Relt = R(X,Y,T)

Resolver en álgebra relacional las siguientes sentencias en SQL:

```
1. Select * From Relt
        Where Exists (Select * From Rels
                Where Relt.X = Rels.X
                And Not Exists (Select * From Relr
                        Where Relr.A = Rels.A
                        And Relr.B = Rels.B ))

2. Select * From Relr
        Where B Not In (Select B From Rels
                Where Exists (Select * From Relt
                        Where Relt.X = Rels.X
                        And Relt.T = 11 ) )

3. Select * From Relt As T1
        Where T1.T < (Select Max(T2.T) from Relt As T2)
```

Obtener la expresión en SQL de las siguientes expresiones en álgebra relacional:

4. rels % π_X (relt)

5. $\pi_{A,B,C,X}$ (relr * rels) % π_X (relt)

6. $\pi_{X,Y,T}$ (relt) * ($\pi_{A,B,X}$ (rels) % $\pi_{A,B}$ (relr))

Metodología para la resolución

No la hay. Lo mejor es hacer muchos ejercicios de AR y SQL por separado. Teniendo frescos ambos temas, habrá que intentar resolver un término e intentar llegar a una expresión equivalente. De todos modos, hay un par de equivalencias que siempre conviene tener a mano, éstas son:

Algebra relacional	Predicado SQL
-	... not in (...)
%	... not exists (... not exists (...))
$\pi_A(R) - \pi_{R_1.A}(R_1 \underset{R_1.A<R_2.A}{\ltimes} R_2)$	max(*R.A*)
$\pi_A(R) - \pi_{R_1.A}(R_1 \underset{R_1.A>R_2.A}{\ltimes} R_2)$	min(*R.A*)

1. $(\pi_X$ (rels) - π_X (rels * relr)) * relt

   Este ejercicio no puede resolverse con la tabla de equivalencias. La idea es pensar que el Exists se puede tratar como una junta natural entre relt y rels, salvo que sobre rels opera el Not Exists. Lo que hace el Not Exists en definitiva es quitar la intersección entre rels y relr, dejando sólo la parte de rels que no "toca" relr. La última junta natural con relt está hecha para devolver todos los atributos.

2. $(\pi_B$ (relr) - π_B ($\sigma_{T=11}$ (rels * relt))) * relr

   La idea es aplicar la equivalencia de la diferencia (-) de AR, es decir, la Not In de SQL. El resto es sencillo: se extraen de relr todos los atributos B que cumplen la condición sobre la junta natural. Una vez hecho eso, como nos piden todos los atributos necesitamos otra junta natural para recuperarlos.

3. π_T ($relt_{T1} \underset{T_1.T<T_2.T}{|\times|} relt_{T2}$) * relt

   En este caso, se debe aplicar la equivalencia de *max(R.A)* pero hasta el punto de obtener la parte que es menor al máximo. Así obtenemos todo el subconjunto de T menor a su máximo, y luego aplicamos una junta natural para recuperar todos los demás atributos.

4. Select A,B,S From $Rels_{S1}$
   Where Not Exists (Select * From Relt
   Where Not Exists (Select * from $Rels_{S2}$
   Where S2.X = Relt.X
   And S2.A = S1.A
   And S2.B = S1.B
   And S2.S = S1.S))

   Nuevamente en la tabla tenemos la equivalencia para la división. Hay dos cosas importantes para tener en cuenta: dentro los atributos que se obtienen al dividir, nunca están presentes los que posee el divisor (en este caso X), y el divisor siempre queda en el medio de la sentencia.

5. Select A,B,C From Relr
   Where Not Exists (Select * from Relt
   Where Not Exists (Select * from Rels
   Where S.A = R.A
   And S.B = R.B
   And S.X = T.X))

   La idea es aplicar la misma equivalencia que en el punto anterior. Si lo vemos durante un buen rato, veremos que la expresión $\pi_{A,B,C,X}$ (relr * rels) % π_X (relt) es equivalente a $\pi_{A,B,C}$ (relr) * ($\pi_{A,B,X}$ (rels) % π_X (relt)); esto último nos ayudará a resolver varios ejercicios similares y surge de la observación que la división opera únicamente sobre la tabla que tiene el atributo del divisor. Para pasar a SQL, tomamos la última expresión y le aplicamos el ejemplo de la divisón (ejercicio anterior).

6.
```
Select X,Y,T From Relt
    Where Not Exists (Select * From Relr
        Where Not Exits (Select * From Rels
            Where   R.A = S.A
            And     R.B = S.B
            And     T.X = S.X ))
```

Algo muy semejante al ejercicio precedente. La idea nuevamente es aplicar el ejemplo de la división. Fijando un rato la vista, $\pi_{X,Y,T}$ (relt) * ($\pi_{A,B,X}$ (rels) % $\pi_{A,B}$ (relr)) es equivalente a la expresión $\pi_{A,B,X,Y,T}$ (relt * rels) % $\pi_{A,B}$ (relr)) con lo cual tenemos de vuelta la mecánica del ejercicio anterior.

Expresiones de equivalencia en conjuntos

Una vez visto el tema anterior, el resto de parcial es dentro de todo sencillo salvo su arbitrario cálculo de correción: una parte del cálculo de la nota está formado por preguntas del tipo Verdadero o Falso con la dificultad que al responder mal se resta el valor del inciso. Lo mejor que se puede hacer para esta clase de ejercicios es no responder si no se está completamente seguro; esto se aplica en los casos que un inciso dependa de la circunstancia (no sea una aseveración tajante). La primer parte que veremos a continuación, surge de lo que ellos llaman equivalencias de conjuntos, que en realidad trata también con Algebra Relacional.

V 1. $(r_1 \times r_2) \,\%\, r_2 \equiv \sigma_{\neg P_1}(r_1) \cup \sigma_{P_1}(r_1)$

V 2. $\sigma_P(\pi_A(r_1 \cup r_2)) \equiv \pi_A(\sigma_P(r_1)) \cup \pi_A(\sigma_P(r_2))$ F

V 3. $r_1 * (\pi_A(r_2) \cap \pi_A(r_1)) \equiv r_1 \cap (r_1 * \pi_A(r_2))$ $\equiv (r_1 \cap r_2) \cup r_1$

V 4. $r_1 - (r_2 - r_1) \equiv (r_1 - r_2) \cup (r_2 \cap r_1)$ V

F 5. $(r_1 \times r_2) \,\%\, (r_2 - r_1) \equiv \sigma_{\neg P_1}(r_1) \cup \sigma_{P_1}(r_1)$ F se rompe uno y no el otro

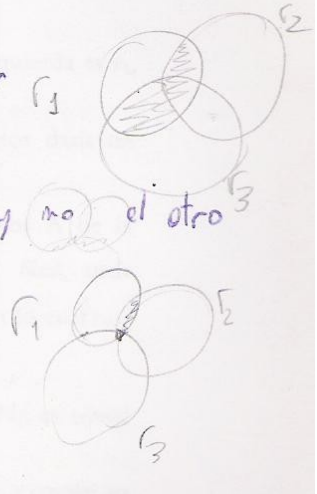
F 6. $r_1 \,\%\, (r_1 \,\%\, \pi_A(r_2)) \equiv \pi_A(r_1) \cap \pi_A(r_2)$

V 7. $r_1 \,\%\, (\pi_A(r_2) \cup \pi_A(r_3)) \equiv (r_1 \,\%\, \pi_A(r_2)) \cap (r_1 \,\%\, \pi_A(r_3))$

F 8. $(r_1 \cup r_2) - (r_2 - (r_2 - r_1)) \equiv (r_1 \cup r_2) \cap r_2$

V 9. $(r_1 \cap r_2) \cup (r_1 - r_2) \equiv r_1 - (r_2 - r_1)$

F 10. $r_1 * r_2 \equiv (\sigma_{\neg P_1}(r_1) * r_2) \cap (\sigma_{P_1}(r_1) * r_2)$



Metodología para la resolución

Debemos aplicar dos metodologías para su resolución:

- **teoría de conjuntos** para aquellos casos donde no tengamos operadores de AR: en general conviene siempre hacer el dibujo sombreando las partes y viendo si coinciden;
- **álgebra relacional** para todos los demás.

Tenemos también una serie de equivalencias con las cual atacar el problema sin entrar en las dos metodologías anteriores:

1	$r_1 \cap r_2$	$r_1 - (r_1 - r_2)$
2	$\sigma_p(\sigma_q(r))$	$\sigma_q(\sigma_p(r))$
3	$\sigma_q(\sigma_p(r))$	$\sigma_{p \cap q}(r)$
4	$\sigma_{p \cap q}(r)$	$\sigma_p(r) \cap \sigma_q(r)$
5	$\sigma_{p \cup q}(r)$	$\sigma_p(r) \cup \sigma_q(r)$
6	$(r_1 \times r_2) \,\%\, r_2$	r_1
7	$r * \pi_A(r)$	r
8	$\pi_{A,B}(r_1) \,\%\, \pi_B(r_2)$	$\pi_A(r_1) - \pi_A((\pi_A(r_1) \times \pi_B(r_2)) - \pi_{A,B}(r_1))$

1. Verdadero. Utilizamos de la tabla la sexta equivalencia para la expresión izquierda; para la derecha vemos que condicionar por la presencia o ausencia de un mismo atributo, al unirlo dá el conjunto completo.
2. Verdadero. Utilizamos de la tabla la quinta equivalencia. La proyección, al ser sobre un atributo distinto a de la selección, es transparente en las dos expresiones.
3. Verdadero. Pensemos que si se "distribuye" r_1 dentro del paréntesis de la primera expresión, nos queda $r_1 * \pi_A(r_2) \cap r_1$ (pues aplicamos la séptima equivalencia).
4. Verdadero. Tenemos que aplicar un poco de teoría de conjuntos: la expresión izquierda es r_1, mientras que la derecha es la unión de la intersección y la parte "aislada" de r_1.
5. Falso. La expresión derecha vimos que daba el conjunto completo, mientras nos daría un "cachito" más que r_1, pues el divisor es "casi" r_2.
6. Falso. La expresión derecha es $\pi_A(r_1 \cap r_2)$ es decir el conjunto de atributos A de la intersección, mientras que la expresión izquierda, la manera de verlo más fácil, sería "distruibuir" la operación externa de división; en el primer caso $r_1 \,\%\, r_1 = \phi$ con lo cual nos aseguramos que al operar con $\pi_A(r_2)$ no obtendremos la expresión derecha.
7. Verdadero. Como regla, la división al distribuirse altera la unión en intersección. No es trivial que se vea.
8. Falso. Al hacer los gráficos, la expresión izquierda dá la unión de los dos conjuntos excepto su intersección. Mientras que la segunda expresión, obtenemos el conjunto r_2.
9. Verdadero. Al hacer los gráficos, tanto la expresión derecha como la izquierda nos da r_1.
10. Falso. Es fácil de ver que si la segunda expresión fuese la unión, entonces nos encontraríamos ante expresiones equivalentes.

Afirmaciones y casos

El tema de las preguntas del tipo Verdadero o Falso se completa con una serie de afirmaciones sobre SQL embebido, creación de índices y optimizaciones sobre utilidades en los DBMS. La idea es nuevamente cuidarse en dar respuestas sólo a las que se esté completamente seguro, pues de no ser así se puede salir de esta parte del parcial con un valor negativo.

- Las columnas de un índice compuesto deben respetar el orden posicional en que aparecen en el esquema de relación de la tabla.
  Falso, la estructura de un índice es independiente de la estructura de la tabla. Además en la sentencia CREATE INDEX ... es donde se establece el orden posicional de los atributos.
- La ejecución de la sentencia DECLARE CURSOR en SQL inmerso, genera como resultado la creación de una tabla temporaria, integrada por todas las tuplas que cumplen las condiciones establecidas en la sentencia SELECT respectiva. FALSO
  Verdadero, un *cursor* no es otra cosa que una tabla temporaria. Cuando se ejecuta desde dentro de un lenguaje *host*, se crea una tabla con el resultado de la ejecución de la instrucción y se procesa con Open/Fetch/Close.
- El usuario creador de una tabla SQL tiene implícitamente otorgados todos los privilegios sobre la misma. Para que, a su vez, pueda otorgar esos privilegios a otros usuarios debe ejecutarse, hacia él, la respectiva sentencia GRANT con la cláusula WITH GRANT OPTION.
  Falso, los permisos de creación, selección, actualización o cualquier otro van por separado; si bien es cierta la frase anexada sobre la sentencia GRANT.
- Es recomendable la creación de *views* sobre las tablas de uso más frecuente, para mejorar el tiempo de respuesta de las consultas.
  Falso, una vista no mejora la velocidad de respuesta de la consulta. En todo caso, sólo evita que el usuario tipee una larga instrucción
- Una vez declarado un *cursor*, en términos de una consulta SELECT, la ejecución de la respectiva sentencia OPEN, examina el contenido de las variables *host*, si las hubiera, en la cláusulas WHERE de la sentencia SELECT asociada al *cursor*, para determinar el conjunto de tuplas que satisfacen esa consulta.
  Verdadero, traduciendo un poco la frase, quiere decir que al realizar un OPEN se obtiene el valor de las variables *host* (aquellas que comienzan con dos puntos, e.g. :var$_1$) y luego se realiza la consulta con dichos parámetros.
- La cláusula ORDER BY de una sentencia SELECT asociada a la declaración de un *cursor*, en el SQL inmerso, debe corresponder posicionalmente a columnas de algún índice definido.
  Falso, dicha claúsula es completamente independiente de la existencia o inexistencia de un índice, ya sea en SQL o SQL inmerso. En cualquier caso se ejecutará más lento, pero se ejecutará.
- En el SQL inmerso, no es posible disparar *triggers* a través de la ejecución de sentencias SELECT.
  Falso... *triggers* es un tema del final; la respuesta "real" es que depende del DBMS. Muchos motores sí lo permiten; ocurre que en la época que estudiaron los profesores, los motores no podían depender de suficiente memoria RAM y velocidad del procesador para ejecutar el lenguaje *host* y el *trigger* y podría llegar a crearse una *race condition*.

- Una sentencia FETCH del SQL inmerso no es válida si previamente no se activó el *cursor* correspondiente a través de una sentencia OPEN.
  Verdadero, necesitamos de la sentencia OPEN previa a la FETCH, la OPEN es la que ejecuta la sentencia SQL y crea la tabla temporal o *cursor.*
- La sentencia de creación de índice UNIQUE sobre un conjunto de atributos de una relación, denota la caracterización de esos atributos como clave candidata..
  Verdadero, recordemos que una clave candidata es aquella que es única pero quizá no es óptima o mínima. A su vez la opción UNIQUE le informa al DBMS que haga respetar la unicidad de la clave.
- Toda sentencia de DML del SQL inmerso en un programa, debiera estar seguida por un testeo del SQLCODE o, en su defecto, estar alcanzada por una cláusula WHENEVER ad-hoc.
  Verdadero, reescribamos lo que están afirmando: DML es Data Manipulation Language es decir Select, Update, etc.; SQLCODE es un elemento de la SQLCA (la estructura "puente" entre el lenguaje *host* y el DBMS) una cláusula WHENEVER perteneciente al lenguaje Cobol es una sentencia que verifica una condición cuando se altera una variable de entorno. En definitiva si no queremos verificar la bandera de estado SQLCODE cada vez que ejecutamos una sentencia del SQL, podemos escribir una claúsula WHENEVER que verificará cada vez que se altere la estructura SQLCA (y por lo tanto la SQLCODE), con lo cual nos ahorramos código Cobol y optimizamos su legibilidad.
- Después de la ejecución de cada sentencia del SQL inmerso, el DBMS retorna información de control al programa en la SQLCA. En este contexto, el valor cero en el SQLCODE denota ejecución éxitosa.
  Verdadero, véase la respuesta anterior. Por cierto, el valor nulo en la SQLCODE es el código de retorno exitoso por antonomasia en C/C++, cualquier otro valor indica código de error.
- Una clave foránea en un esquema de relación dado, no puede incluir un subconjunto propio de atributos que a su vez constituya otra clave foránea en ese mismo esquema de relación
  Verdadero, acordarse de que una clave foránea (de no ser nula) en la otra entidad será la clave primaria; a su vez, una clave primaria, es única y mínima; por lo tanto no puede incluir un subconjunto propio de atributos que sea clave. FALSO
- Una superclave que satisface la propiedad de unicidad constituye una clave candidata.
  Falso, una clave candidata requiere de dos propiedades: unicidad y minimalidad (ningún subconjunto de dicha clave tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes sin destruir la unicidad).
- La estructura SQLCA junto con el SQLCODE, dentro de SQL inmerso de lenguaje *host*, solamente se encuentra disponible al realizar una sentencia OPEN. Falso, viene con trampita... la SQLCA es una estructura que para el caso del lenguaje C/C++ ésta se incluye en la sentencia *#include*, para el caso de Cobol, es una sentencia que se anexa a preprocesador del compilador. Esto significa que la estructura está "disponible" más allá de que se haga un OPEN o no; según lo conversado con el profesor por "disponible", en este contexto, se denomina a la memoria ablocada por el compilador, más allá de que se utilice o no.
- Los argumentos que forman parte de las funciones de agregación o colectivas, son buenos candidatos para la creación de índices. Verdadero, viene con trampita... las funciones de agregación son cinco: AVG, SUM, COUNT, MAX y MIN. Si bien para las tres primeras es inútil la creación de un índice (y además crearía más labor de mantenimiento), para las dos últimas sería de utilidad puesto que podríamos acceder al máximo o mínimo con un único seek, mientras no exista ninguna condición extra en la claúsula WHERE.