UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE SISTEMAS
FUNDAMENTOS DE BASES DE DATOS
EXAMEN PARCIAL

NOMBRE: Harold Armando Achicanoy Estrella

CÓDIGO: 1702943

1. **Suponga que usted tiene una máquina del tiempo y puede viajar a finales de los años 70 cuando la discusión entre Relacional vs No relacional estaba en pleno auge. Con el conocimiento que tiene sobre la evolución del Hardware, Software e Internet:**

   a) **Defienda el modelo Relacional**
   El auge de la era de internet permitirá tener una mayor conectividad por ende un mayor flujo de datos que pueden almacenarse en un modelo de bases de datos simple y robusto como el modelo relacional. Utilizando los potentes recursos que proporcionarán el software y hardware, el modelo relacional podrá optimizar las consultas de manera eficiente haciendo uso del expresiones equivalentes derivadas del algebra relacional, permitiendo reducir tiempos de consulta y recursos utilizados.

   b) **Defienda el modelo No Relacional**
   Con el creciente volumen de datos provenientes de diversas fuentes en formatos no estructurados, el modelo de datos no relacional en consideración – modelo de red o CODASYL – entraría a jugar un papel importante explotando la posibilidad de modelar relaciones con diversas dependencias en forma de red. La constante mejora en términos de hardware y software no sería un obstáculo para el procesamiento de un registro a la vez en lugar de bloques, sin embargo, habría que mejorar en el campo de la independencia de datos, tanto física como lógica. Proporcionando así una fuerte herramienta eficiente para el almacenamiento de relaciones complejas provenientes de internet.

2. **Sustente con sus propias palabras por qué el concepto de "dominio" del modelo relacional es equivalente a al concepto de "clases objeto" en el modelo orientado a objetos.**

   Examinando los conceptos para cada modelo, se tiene que en el modelo relacional el **dominio** hace referencia a los posibles valores que puede tomar un atributo de una relación en particular, con lo cual dicho atributo no puede adquirir otro tipo de valor fuera de ese conjunto de valores. Mientras en el modelo orientado a objetos, las **clases objeto** constituyen básicamente una plantilla para la creación de objetos que proporcionan sus valores iniciales o estructura en la definición. Por tanto, haciendo analogía entre los dos conceptos presentados podría decirse que son equivalentes en cuanto a su función de definir o asignar los correspondientes valores que puede tomar su unidad básica (atributo u objeto).

3. **Explique cada uno de los componentes de un optimizador de bases de datos y mencione cuales algoritmos usa cada componente.**

   La optimización de consultas en un sistema gestor de bases de datos (DBMS) constituye un punto importante al momento de retornar de manera eficiente los resultados de un query optimizando los recursos disponibles en función de transformaciones eficientes de consulta. En términos generales un optimizador de bases de datos, funciona como un vínculo eficiente que se da desde el momento en que se crea la consulta por parte del usuario en lenguaje SQL hasta la consulta física de los datos almacenados, para proporcionar la respuesta precisa a la consulta de manera óptima.
   El optimizador de bases de datos consta de los siguientes componentes:
   Query parser, es el encargado de interpretar la consulta escrita en el lenguaje SQL por el usuario, generando una descomposición de la consulta en términos semánticos y sintácticos para ser interpretados por el optimizador a través de un árbol de consulta.
   Query optimizer, el optimizador de consulta recibe el árbol generado por el query parser, el cual necesita ser evaluado mediante diferentes planes de consulta generados por el espacio de búsqueda, midiendo sus respectivos y escogiendo el plan que tenga el menor costo asociado. Los diferentes planes de consulta se generan a través de transformaciones haciendo uso del algebra relacional a través de propiedades entre operadores JOIN, reducción de consultas multi-bloque en consultas simples y uso del operador SEMIJOIN, mientras los costos se miden en términos de tiempo de CPU, memoria usada, tiempos Input/Output, entre otros. Finalmente es un algoritmo de enumeración el encargado de seleccionar el plan más eficiente para el query proporcionado a través del uso de algoritmos de programación dinámica u órdenes interesantes.
   Query execution engine, es el encargado de implementar el plan de consulta escogido por el query optimizer para obtener el resultado final del query para retornar al usuario.

4. **Explique el concepto de independencia lógica e independencia física de los datos, proponga ejemplos.**

   La independencia física corresponde a la capacidad de la base de datos de realizar cambios a nivel de almacenamiento físico (unidades de almacenamiento como: cintas magnéticas, discos duros, entre otros) sin alterar el funcionamiento del motor de consultas usado por el usuario. Por otro lado, la independencia lógica permite modificar el esquema conceptual de la base de datos sin afectar las aplicaciones que usan los usuarios para hacer las consultas. Un ejemplo de independencia física consiste en la reorganización de ficheros de un disco duro a otro, bajo el modelo de base de datos relacional, aquí aunque hubo una modificación a nivel físico de los datos almacenados, el usuario final puede hacer uso de ellos sin verse alterada la consulta. Finalmente, un ejemplo de independencia lógica puede darse al incluirse nuevas relaciones en un esquema relacional previamente definido, esto no altera de ninguna manera las consultas del esquema base.

5. **Para reducir el número de planes de consulta los optimizadores de bases de datos por lo general usan algunas heurísticas, mencione 2 y explíquelas.**

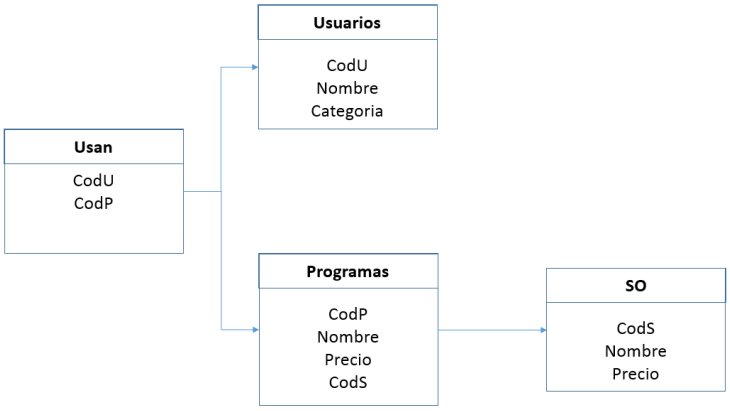
A continuación se presentan dos heurísticas de los algoritmos empleados por los optimizadores de bases de datos para reducir el número de planes de consulta basados en el optimizador System-R (Chaudhuri, 1998):

a) ***Programación dinámica***: donde se asume que para obtener un plan óptimo de una consulta SPJ (SELECT-PROJECT-JOIN) Q que se compone de k joins, se deben considerar solamente los planes óptimos con menor costo para subexpresiones de Q que consisten en k-1 joins y extender esos planes con un join adicional.
b) ***Ordenes interesantes***: al momento de aplicar un operador JOIN sobre más de dos relaciones que cuentan con llaves, el algoritmo de ordenes interesantes evalúa los costos de aplicar JOINs entre pares de relaciones aprovechando los ordenamientos que se pueden dar entre las llaves de una relación a otra para reducir el costo asociado a dichos planes, reduciendo el espacio de búsqueda y por tanto el número de planes a evaluar.

6. Dado el siguiente esquema relacional, escriba las consultas en álgebra relacional, cálculo relacional (tuplas o dominios) y SQL. Cuando escriba las consultas en álgebra relacional tenga la visión de un optimizador de consultas.

SO (CodS, Nombre, Precio)
Programas (CodP, Nombre, Precio, CodS)
Usuarios (CodU, Nombre, Categoria)
Usan (CodU, CodP)

Esquema de la base de datos



- Mostrar el código y nombre de los usuarios que no usan programas del SO Linux

Algebra relacional

$$R_1 \leftarrow \Pi_{CodU,Nombre}(\rho_U(Usuarios))$$
$$R_2 \leftarrow \Pi_{Usuarios.CodU,Usuarios.Nombre}(Usuarios) \bowtie_{Usuarios.CodU=Usan.CodU} Usan \bowtie_{Usan.CodP=Programas.CodP} Programas \bowtie_{Programas.CodS=SO.CodS} \sigma_{SO.Nombre="Linux"}(SO)$$

$$R_3 \leftarrow R_1 - R_2$$

Cálculo relacional

$$\{< U.CodU, U.Nombre > | U \in Usuarios \wedge \neg \exists\, Q\ P\ R | Q \in Usan\ \wedge P \in Programas \wedge R \in OS \wedge R.Nombre = "Linux" \wedge (P.CodS = R.CodS) \wedge (U.CodU = Q.CodU)\}$$

SQL

```
SELECT Usuarios.CodU, Usuarios.Nombre FROM Usuarios
WHERE Usuarios.CodU NOT IN(
SELECT Usuarios.CodU FROM Usuarios
INNER JOIN Usan ON Usuarios.CodU = Usan.CodU
INNER JOIN Programas ON Usan.CodP = Programas.CodP
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE SO.nombre = 'Linux');
```

- Obtener el nombre de los programas y su respectivo SO que son utilizados por usuarios de categoría "Dummies" y "Senior".

Algebra relacional

$$R_1 \leftarrow \Pi_{Programas.Nombre,SO.Nombre}(Programas) \bowtie_{Programas.CodS=SO.CodS} SO \bowtie_{Programas.CodP=Usan.CodP} Usan \bowtie_{Usan.CodU=Usuarios.CodU} \sigma_{Usuarios.Categoria="Dummies"}(Usuarios)$$

$$R_2 \leftarrow \Pi_{Programas.Nombre,SO.Nombre}(Programas) \bowtie_{Programas.CodS=SO.CodS} SO \bowtie_{Programas.CodP=Usan.CodP} Usan \bowtie_{Usan.CodU=Usuarios.CodU} \sigma_{Usuarios.Categoria="Senior"}(Usuarios)$$

$$R_3 \leftarrow R_1 \cap R_2$$

Cálculo relacional

$$\{< P.Nombre, O.Nombre > | P \in Programas \wedge O \in OS \wedge \exists Q\ R | Q \in Usan \wedge U \in Usuarios \wedge U.Categoria = "Dummies"\} \cap \{< P.Nombre, O.Nombre > | P \in Programas \wedge O \in OS \wedge \exists Q\ R | Q \in Usan \wedge U \in Usuarios \wedge U.Categoria = "Senior"\}$$

SQL

```
SELECT DISTINCT Programas.Nombre AS Programa, SO.Nombre AS Sistema_Operativo
FROM Usan
INNER JOIN Programas ON Usan.CodP = Programas.CodP
INNER JOIN SO ON Programas.CodS =  SO.CodS
INNER JOIN Usuarios ON Usan.CodU = Usuarios.CodU
WHERE Usuarios.Categoria = 'Dummies'
AND Programas.Nombre IN (
SELECT Programas.Nombre FROM Usan
INNER JOIN Programas ON Usan.CodP = Programas.CodP
INNER JOIN SO ON Programas.CodS = SO.CodS
```

```
INNER JOIN Usuarios ON Usan.CodU = Usuarios.CodU
WHERE Usuarios.Categoria = 'Senior');
```

- Mostrar el nombre de los usuarios que usan todos los programas del sistema operativo Windows

Algebra relacional

$$R_1 \leftarrow \Pi_{Usuarios.Nombre}(Usuarios)$$
$$R_2 \leftarrow \Pi_{Usuarios.Nombre}(Usuarios) \bowtie_{Usuarios.CodU=Usan.CodU} Usan \bowtie_{Usan.CodP=Programas.CodU} Programas \bowtie_{Programas.CodS=SO.CodS} SO\left(\sigma_{SO.Nombre="Windows"}(SO)\right)$$
$$R_3 \leftarrow R_1 \div R_2$$

Cálculo relacional

$$\{< U.Nombre > | U \in Usuarios \wedge \forall\ Q\ P\ R | Q \in Usan\ \wedge P \in Programas \wedge R \in OS \wedge R.Nombre = "Windows" \wedge (P.CodS = R.CodS) \wedge (U.CodU = Q.CodU)\}$$

SQL

```
SELEC Usuarios.Nombre FROM Usuarios
INNER JOIN Usan ON Usuarios.CodU = Usan.CodU AND Usan.CodP in
(SELECT Programas.CodP FROM Programas
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE SO.Nombre = 'Windows') GROUP BY Usuarios.Nombre
HAVING count(*) = (SELECT count(*) FROM Programas
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE SO.Nombre = 'Windows');
```

- Mostrar el nombre de los usuarios de categoría "Senior", el nombre de los programas que usan y sus respectivos sistemas operativos

Algebra relacional

$$R \leftarrow \Pi_{Usuarios.Nombre,Programas.Nombre,SO.Nombre}\left(\sigma_{Categoria="Senior"}(Usuarios)\right) \bowtie_{Usuarios.CodU=Usan.CodU} Usan \bowtie_{Usan.CodP=Programas.CodP} Programas \bowtie_{Programas.CodS=SO.CodS} SO$$

Cálculo relacional

$$\{< U.Nombre, P.Nombre, R.Nombre > | U \in Usuarios \wedge P \in Programas \wedge R \in OS \wedge Q \in Usan \wedge (P.CodS = R.CodS) \wedge (U.CodU = Q.CodU) \wedge (U.Categoria = \text{"Senior"})\}$$

SQL

```
SELECT Usuarios.Nombre AS Nombre, Programas.Nombre AS Programa, SO.Nombre
AS Sistema_Operativo
FROM Usuarios
INNER JOIN Usan ON Usuarios.CodU = Usan.CodU
INNER JOIN Programas ON Usan.CodP = Programas.CodP
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE Usuarios.Categoria = 'Senior';
```

- Mostrar el nombre de los usuarios solo usan programas del SO Unix

Algebra relacional

$$R_1 \leftarrow \Pi_{Usuarios.Nombre}(Usuarios) \bowtie_{Usuarios.CodU=Usan.CodU} Usan \bowtie_{Usan.CodP=Programas.CodP} Programas \bowtie_{Programas.CodS=SO.CodS} \sigma_{SO.Nombre="Unix"}(SO)$$

$$R_2 \leftarrow \Pi_{Usuarios.Nombre}(Usuarios) \bowtie_{Usuarios.CodU=Usan.CodU} Usan \bowtie_{Usan.CodP=Programas.CodP} Programas \bowtie_{Programas.CodS=SO.CodS} \sigma_{SO.Nombre!="Unix"}(SO)$$

$$R_3 \leftarrow R_1 - R_2$$

Cálculo relacional

$$\{< U.Nombre > | U \in Usuarios \wedge \forall\ Q\ P\ R | Q \in Usan\ \wedge P \in Programas \wedge R \in OS \wedge (P.CodS = R.CodS) \wedge (U.CodU = Q.CodU) \wedge \neg \exists Q\ P\ R | Q \in Usan\ \wedge P \in Programas \wedge R \in OS \wedge (R.Nombre! = "Unix") \wedge (P.CodS = R.CodS) \wedge (U.CodU = Q.CodU)\}$$

SQL

```
SELECT Usuarios.Nombre FROM Usuarios
INNER JOIN Usan ON Usuarios.CodU = Usan.CodU
INNER JOIN Programas ON Usan.CodP = Programas.CodP
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE SO.Nombre = 'Unix' AND Usuarios.Nombre NOT IN
(SELECT Usuarios.Nombre FROM Usuarios
INNER JOIN Usan ON Usuarios.CodU = Usan.CodU
INNER JOIN Programas ON Programas.CodP = Usan.CodP
INNER JOIN SO ON Programas.CodS = SO.CodS
WHERE SO.Nombre != 'Unix');
```