Trabajo 5 de bases de datos 2 (20%)

En este trabajo el objetivo es simular la labor de los índices. Se debe aclarar que el objetivo **NO** es competir con los índices nativos que ofrece un sistema de gestión de bases de datos, ya que estos suelen ofrecer índices optimizados y apropiados para la gestión de los datos en disco duro (como los árboles B+ (y sus variantes), bitmap, hash, entre otros) de los datos.

Punto 1. Sean:

```
CREATE TABLE empleado(
codigoE NUMBER(10) PRIMARY KEY,
nombreE VARCHAR2(20) NOT NULL,
depE NUMBER(8) NOT NULL --depE NO será clave foránea
);
--La tabla se llenará con filas...

CREATE TABLE departamento
(
codigoD NUMBER(8) PRIMARY KEY,
nombreD VARCHAR2(20) NOT NULL,
direccionD VARCHAR2(20) NOT NULL
);
--La tabla se llenará con filas...
```

Construya (mediante **PL/SQL**) a partir de todos los datos de la tabla departamento una *skip list*.
Cada nodo de la skip list tiene la siguiente estructura:

- Numnodo: este es un entero para enumerar los nodos de la skip list. El primer nodo (la cabeza, *head*) de la skip list se le asigna el número 1, al siguiente el 2, y así sucesivamente. Ver figura más abajo.
- CodigoD: o sea, el código del departamento por el cual se crea la skip list ordenada ascendentemente.
- Grupo de punteros (característicos de un nodo en una skip list).
- nombreD y DireccionD.
- Ptrback: El nodo tendrá un puntero adicional llamado Ptrback (aparte del grupo de punteros mencionado). El propósito de este puntero adicional es apuntar al nodo inmediatamente anterior para ofrecer así un mecanismo de ordenamiento descendente. Es decir, mediante este puntero se construye una lista enlazada convencional.

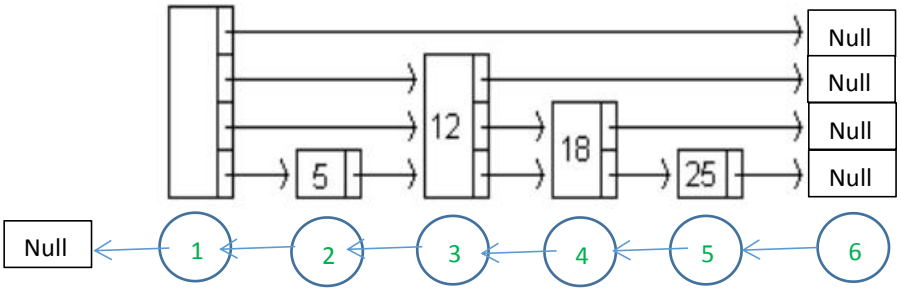
Para la construcción de la skip list, el programa debe recibir un parámetro maxPtrs que será una potencia de 2 (1, 2, 4, 8 o 16) siendo 16 el máximo posible. Este número determina el máximo número de punteros (aparte del Prtback) que tendrá un nodo en la skip list (si maxPtrs = 1, la skip list se degenera en una lista enlazada convencional).

Así, se deben tener los siguientes programas en PL/SQL (se recomienda ponerlos en un paquete):

(5%) CrearSL(maxPtrs NUMBER): crea la skip list a partir de todos los datos de la tabla departamento. La skip list debe quedar guardada en una tabla llamada **indexdepskip,** donde cada fila de dicha tabla corresponde a un nodo de la skip list.
Recuerde que el número de punteros de **cada nodo** (aparte del Prtback), entre 1 y maxPtrs, se debe generar en forma **aleatoria**.

(5%) ImprimirAscSL(): imprime la skip list. La lista se debe imprimir en orden ascendente (según CodigoD). No use en ningún momento ORDER BY. Supongamos que se tiene la skip list de la siguiente figura. En la figura **dentro de los nodos solo se muestra el codigoD**, pero **realmente allí también**

están los demás atributos (NombreD, DireccionD). El prtback se muestra con las flechas azules. Los números en los círculos debajo de cada nodo corresponden al Numnodo.



Para el ejemplo de la anterior figura, ImprimirAscSL debe imprimir lo siguiente:

Nodo 1
Datos dpto: No tiene
Número de punteros: 4
De arriba a abajo los punteros apuntan a los nodos: 6, 3, 3 y 2.
Nodo 2
Datos dpto: 5, Ventas, Cr 5 #8-45
Número de punteros: 1
De arriba a abajo los punteros apuntan a los nodos: 3.
Nodo 3
Datos dpto: 12, Recursos Inhumanos, Av 9 #9-76
Número de punteros: 3
De arriba a abajo los punteros apuntan a los nodos: 6, 4 y 4.

Y así sucesivamente...

(5%) ImprimirDescSL(): imprime la skip list en orden descendente (según CodigoD). El formato de impresión es idéntico a ImprimirAscSL, solo que descendente (por supuesto, acá hay que aprovechar el puntero ptrback...No puede usar ORDER BY).

(5%) ConsultaDpto(codigoD NUMBER): recibe como parámetro el código de un departamento y lo busca en la skip list. **Para ello se debe usar el algoritmo de búsqueda propio de una skip list (al fin y al cabo para eso se creó,** en muchos artículos explican como buscar un dato en una skip list, comenzando con el puntero superior de la cabeza...**)**. Para verificar que se lleva a cabo la búsqueda mediante la skip list, se debe imprimir el proceso de búsqueda llevado a cabo por el algoritmo de búsqueda, imprimiendo los nodos (solo imprima el atributo Numnodo y el codigoD) **que son visitados durante el proceso de búsqueda**. Al final, si el codigoD es encontrado, se debe imprimir su nombreD y direccionD. Si el codigoD no se encontró, se debe imprimir: "Dpto. No existe" (pero así el codigoD no se encuentre, se debe imprimir como transcurrió el proceso de búsqueda). No se va a solicitar un formato de la salida específico, lo importante es que cumpla con lo indicado.

***Triggers* sobre la tabla departamento.**

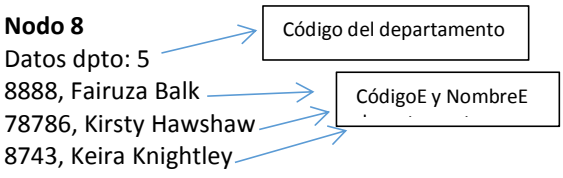
- **(5%)** Se debe construir un *trigger* sobre la tabla departamento, de tal forma que si se borran filas de dicha tabla (departamento), en la skip list también se produzca el borrado correspondiente. Se deben soportar borrados masivos.
- **(5%)** Se debe construir un *trigger* sobre la tabla departamento, de tal forma que si se actualizan datos de la tabla departamento, la actualización respectiva se propague a la skip list. Cualquiera de los tres atributos de la tabla departamento (o varios de ellos simultáneamente) pueden ser actualizados. Por simplicidad, cada update modificará máximo una fila de la tabla departamento (en otras palabras, no se probará con updates masivos).

- **(5%)** Se debe construir un *trigger* sobre la tabla departamento, de tal forma que si se inserta una fila en ella, la inserción respectiva se propague a la skip list. Por simplicidad, no se probará con inserts masivos.

(5%) Join(): Usando la skip list, al invocar este procedimiento se debe imprimir el *join* entre la tabla empleado y la tabla **indexdepskip.**
Para hacer el *join*, se debe ir tomando cada fila de la tabla empleado y se busca si su depE existe en la tabla indexdepskip (**usando el proceso de búsqueda típico de una skip list**), si lo encuentra, entonces se imprime la tupla: (codigoE, nombreE, depE, codigoD, nombreD, DireccionD) y se continúa el proceso con la siguiente fila de empleado. Note que acá se está simulando un *index nested loops*, tal y como se plantea en la diapositiva 11 de la presentación Pres21BDII. Para hacer el *join* **no puede usar la tabla departamento**. Debe hacer el *join* mediante el método *index join*.

De manera análoga, construya ahora una skip list usando la tabla empleado. Esta segunda skip list se ordena ascendentemente por el atributo depE de empleado. Note que en este caso habrá datos repetidos en el campo depE (ya que varios empleados pueden trabajar en el mismo departamento); por lo tanto, la skip list deberá acomodar estos datos repetidos así: todos los empleados de un mismo departamento deben quedar en el nodo de ese departamento. Esta skip list debe quedar guardada en una tabla llamada **indexempskip**.
Los programas para esta segunda skip list son análogos a la primera pero aclarando que:

(5%) CrearSL(maxPtrs NUMBER): análogo al de la primera skip list.
(3%) ImprimirAscSL(): análogo al de la primera skip list, pero al imprimir el campo Datos dpto, allí deben salir todos los empleados del nodo, ejemplo de un nodo:



(3%) ImprimirDescSL(): análogo al de la primera skip list.
(5%) ConsultaDpto(depE NUMBER): análogo al de la primera skip list, pero si el depE es encontrado, se imprimen sus empleados. Igualmente se debe imprimir el proceso de búsqueda.

(15%) ***Triggers*** **sobre la tabla empleado**: análogos a los de la primera skip list.

(5%) Join(): Usando las dos skip lists, al invocar este procedimiento se debe imprimir el *join* entre la tablas **indexdepskip** y la tabla **indexempskip.** Para hacer el *join*, se debe hacer el proceso de mezcla (*merge*) explicado en la diapositiva 24 de la presentación Pres21BDII (note que acá se está simulando un *sort merge join*, con la ventaja de que como las skip lists ya están ordenadas por el atributo de *join*, se evita el proceso de ordenamiento y solo se hace la mezcla (esta misma ventaja la ofrece un B+)). Para hacer el *join* **no puede usar las tablas empleado y departamento**, solo las tablas **indexdepskip** y la tabla **indexempskip** (al fin y al cabo para eso se construyeron).

Punto 2. En este punto[1] se va a ensayar otro tipo de estructura que nos servirá como índice para facilitar la respuesta de determinadas consultas. Se trata del *segment tree*.

Sea:
CREATE TABLE tash_sultana(

[1] Este punto está dedicado a "*Notion*" de la multi-instrumentalista Tash Sultana.

consecutivo NUMBER(8) PRIMARY KEY,
valornotion NUMBER(8) NOT NULL
);

--La tabla se llenará con filas...Se garantizará que las filas son consecutivas (según el atributo consecutivo) y que el consecutivo inicia en cero.

Ejemplo:

consecutivo	**valornotion**
0	10
1	30
4	10
3	20
2	66

Por supuesto en una relación las filas no tienen orden, pero se garantiza que allí están los valores consecutivos, para este **ejemplo**, consecutivos del 0 al 4.

Construya un segment tree a partir de la tabla tash_sultana. La función para construir el segment tree será **máximo de valornotion** (de esta forma, el segment tree nos ayudará a responder consultas basadas en hallar el máximo valornotion en un intervalo dado...).
Defina la estructura de cada nodo (por supuesto deberá tener como mínimo estos datos: el intervalo, el valornotion máximo, puntero al nodo izquierdo y puntero al nodo derecho).

Así se deben tener los siguientes programas en PL/SQL (se recomienda ponerlos en un paquete):

(6%) CrearST(): crea el segment tree a partir de todos los datos que están en la tabla tash_sultana. El segment tree debe quedar guardado en una tabla llamada **indexsegtree,** donde cada fila de dicha tabla corresponde a un nodo del segment tree.

(6%) ImprimirST(sw number): Imprime el segment tree. Sw puede ser 0 o 1.

- Si sw = 0: El programa debe imprimir el segment tree por niveles, primero se imprimen los datos (intervalo y valornotion) del nodo raíz, luego se imprimen los datos de sus nodos hijos (hijo izquierdo e hijo derecho), luego los datos de los nietos y así sucesivamente. No se va a solicitar un formato de salida específico, lo importante es que cumpla con lo indicado.
- Si sw = 1: El programa debe imprimir el segment tree por niveles, pero comenzando desde las hojas de izquierda a derecha (nodos inferiores) y luego con los nodos ubicados en el siguiente nivel (padres de las hojas), hasta llegar al nodo raíz, el cual se imprimirá de último. No se va a solicitar un formato de salida específico, lo importante es que cumpla con lo indicado.

(6%) ConsultaMaxValNotion(inicio, fin): busca el valor máximo (valornotion) en el intervalo dado [inicio, fin]. Por supuesto, para encontrar el valor máximo en el intervalo dado se debe usar el segment tree creado (**indexsegtree**) y para verificar que el programa esté usando el segment tree se deben imprimir los pasos seguidos por el proceso de búsqueda. No se indicará acá como debe imprimir dichos pasos, lo importante es que en la impresión se muestre, en el formato que usted defina, cuales nodos fueron visitados durante el proceso de búsqueda y que valor fue retornando por cada nodo visitado tal y como se explica en el video. https://www.youtube.com/watch?v=1qpUfxVBGaY

(6%) ***Trigger***
Finalmente, haga un *trigger* tal que si se incrementa (incremento **positivo**) el valornotion de un consecutivo en la tabla tash_sultana, se actualice también el índice (es decir, la tabla indexsegtree). Solo se actualizará una fila en cada update (es decir, por simplicidad, no se manejarán updates masivos). Si el incremento es negativo, el *trigger* rechaza la actualización. Nota: debe actualizar el índice, es decir, NO borrarlo y volverlo a crear desde cero.
No se considerarán *triggers* de inserción ni de borrado.

Notas adicionales:

- Sus soluciones deben funcionar para cualquier cantidad de filas que tengan las tablas.
- Puede usar todas las estructuras de datos y todas las tablas auxiliares que desee. Sin embargo para almacenar los datos de los índices, solo puede usar las tablas indicadas.
- Para entregar por email a *fjmoreno@unal.edu.co*, el martes 24 de noviembre **hasta las 11 am**. Solo se califican trabajos enviados a ese correo. Por cada grupo de trabajo solo se recibe un trabajo.
- **No se reciben trabajos en hora posterior**. No se reciben versiones "mejoradas". No se califican trabajos enviados "por accidente" a otros correos.
- **Se debe incluir un informe donde se describa cómo se solucionó cada punto.** Este informe hace parte de la calificación del trabajo (máximo 6 hojas). **No enviar los datos de prueba que usted usó para probar sus códigos.**
- Grupos de **tres** personas.
- Los trabajos deben ser independientes entre los grupos. Trabajos copiados **así sea en un SOLO punto** se califican con 0 (cero) en su totalidad para todos los integrantes. Las soluciones presentadas deben ser originales. El trabajo debe ser desarrollado por los integrantes del grupo no por personas ajenas a él.
- El monitor les puede ayudar con aspectos técnicos pero su función **no** es hacerles la práctica **ni está autorizado** para **cambiar las condiciones del trabajo**.
- **Si trabaja con otras herramientas**, así su trabajo funcione y sea "espectacular", el trabajo **NO** será calificado.
- Cualquier duda consultarla con el profesor.