**Informe: Trabajo 1 Bases de datos 2 2021-1**

**Integrantes:**

Santiago Espinosa Arteaga cc 1037671102 Juan Pablo Carvajal cc 1152472428

**Punto 1:**

**Elementos necesarios**

Para la ejecución de este punto es necesario crear la tabla auxiliar **cerdoAux** definida en el archivo inserts.sql en la carpeta punto 1

**Desarrollo de la solución:**

**Algoritmo de apoyo**

Para llevar a cabo este ejercicio nos basamos en el algoritmo conocido como 0/1 Knapsack, el cual es explicado en los siguientes enlaces

<https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

<https://youtu.be/xCbYmUPvc2Q>

Este algoritmo es usado para poder escoger la combinación de cerdos disponibles donde la suma de sus pesos sea la máxima permitida ya sea por el peso del camión o por el pedido del cliente.

Para lograrlo nos apoyamos en la programación dinámica guardando en un arreglo temporal posibles soluciones y en base a ellas encontrar la mayor posible.

**Estructura del código**

**Variables**

En principio se definen las variables a usar, donde podemos resaltar que la variable **kilosPedidos** representa la cantidad de kilos que hace el cliente, después de esto se definen tipos a usar dentro del algoritmo como arrays de camiones, cerdos y números. También se hace uso de múltiples variables auxiliares que nos permiten guardar datos temporales, realizar los ciclos y guardar los textos de salidas. Posteriormente tenemos un conjunto de variables de control del algoritmo de selección. Y ya por último se declara una variable que realiza el trabajo de acumulador de los kilos despachados

**Controles iniciales**

En principio se llenan en arrays todos los camiones y todos los cerdos para poder iterar con ellos, vale aclarar que los cerdos se llenan en un array de tipo cerdoAux, el cual tiene una columna de más de tipo NUMBER(1) que nos permitirá saber si un cerdo ya fue seleccionado o no por el algoritmo de selección.

Después de esto se consideran unos condicionales para descartar casos extremos como que no haya camiones, que no haya cerdos, que los cerdos sean más pesados que la cantidad de kilos pedidos y que los cerdos sean más pesados que los camiones.

**Algoritmo de selección**

Una vez pasados estos casos se entra en un bucle de camiones desde el más pesado al menos pesado. Allí primero se filtra la cantidad de kilos límite que el algoritmo recibirá, esta puede ser el peso total del camión o un valor menor que sale de restar los kilos pedidos del cliente con la cantidad de kilos despachados usados por los camiones al momento. Después de esto se llena un array de cerdos posibles con aquellos cerdos que no han sido seleccionados, una vez teniendo esto se le pasa estos 2 parámetros al algoritmo de selección, el cual como comenté antes se apoya en un arreglo dinámico para guardar la combinación más grande posible cumpliendo la regla del arreglo de cerdos posibles.

Una vez se tiene este valor más grande posible. Se aplica un proceso de retroceso en el arreglo dinámico de solución para poder obtener la combinación de elementos cuya suma da como resultado el valor óptimo. Si durante este proceso se encuentra que el arreglo óptimo es vacío significa que no quedan camiones que puedan llevar a los cerdos restantes por lo cual se termina el proceso y se pasa al informe final

**Final del proceso**

Una vez se tiene el arreglo óptimo, se itera sobre él para recolectar la información a mostrar sobre los cerdos usados y después sobre la capacidad usada del camión.

Al final del ciclo se actualiza el arreglo principal de cerdos poniendo 1 el atributo seleccionado de aquellos cerdos que fueron usados para el arreglo óptimo.

Después se resta del peso restante posible toda la capacidad del camión. Si este proceso llega a dar como resultado un número negativo significa que la suma de los pesos de los camiones enviados al momento es mayor a la capacidad pedida por lo que el ciclo de camiones se detiene y se pasa al informe final. En caso contrario se continúa con el ciclo con el siguiente camión más pesado por despachar. Se repite todo esto hasta que el contador de kilos sea negativo o hasta que ya no queden camiones disponibles y por ende se termine el ciclo.

**Punto 2**

**Triggers A y B**

En un solo trigger BEFORE INSERT, primero se setea el número de hijos del dato ingresado en 0.

Después de esto se verifica que el atributo padre no sea nulo, dado el caso se investiga de la tabla individuo el número de hijos del padre. Se le aumenta una unidad y por último se hace un UPDATE para aumentar el número de hijos del padre.

Al final se tiene un control de excepción NOT DATA FOUND el cual se dispara si se intenta ingresar un individuo con un padre no válido, este manejo de excepciones no hace nada, pero es necesario para que el trigger se ejecute correctamente y ya que esta excepción la controle el SGBD como tal.

**Triggers C y D**

Para ejecutar este trigger es necesario crear esta tabla auxiliar **individuoAux** especificada en el archivo inserts.sql de la carpeta punto 2

Como en este proceso actualiza tabla de individuo mientras se ejecuta es susceptible al problema de la tabla mutante, por esto se usa un COMPOUND TRIGGER usando las partes de BEFORE STATEMENT, BEFORE EACH ROW y AFTER STATEMENT

En la parte de BEFORE STATEMENT se guarda todos los registros de la tabla individuo en la tabla auxiliar individuoAux. Posteriormente se hace NULL el atributo padre en la tabla de individuo.

En la parte BEFORE EACH ROW se guardan en un arreglo los códigos de los registros que se van a eliminar

En la parte de AFTER STATEMENT se realiza lo siguiente

Primero se itera sobre todos los resultados guardados en la tabla auxiliar individuoAux, para cada uno de estos elementos se comprueba que el individuo al que el atributo padre referencia todavía exista, en caso positivo se setea el atributo padre para el registro en la tabla individuo. De esta manera se hacen NULL el atributo padre de los hijos cuyo padre fue eliminado.

En segundo lugar, itera sobre el arreglo de los elementos eliminados, allí primero se captura su atributo padre de la tabla individuoAux, posteriormente se comprueba que el padre siga existiendo mediante una consulta sobre la tabla individuo, en caso afirmativo se modifica su atributo nro\_hijos restándole una unidad, por último se actualiza el registro del padre.

**Trigger E**

De manera similar a los triggers anteriores se usa un compound trigger de UPDATE sobre la propiedad valor de la tabla individuo además de esto tiene la propiedad FOLLOWS del trigger **actualizaPadres** el cual es en este caso el trigger F, esto se hace con el propósito de evitar comportamientos inesperados si se hace un update que tenga valor y código. Se usan bloques como before statement, before each row y after statement, además debido a que durante este trigger se hace uso de inserts y estos hacen uso de la tabla **individuoAux**, es necesario crear una nueva tabla **indiviuoAuxUpdate** con columnas equivalentes a la tabla individuo y una extra llamada valorNuevo. ambas tablas están definidas en el archivo inserts.sql de la carpeta Punto 2

En la sección before statement se insertan en la tabla indiviuoAuxUpdate todas las filas presentes en la tabla individuo añadiendo el valor 0 en la columna valorNuevo, en la sección before each row se usa el :old.valor y :new.valor para verificar que la actualización realizada cumpla con las condiciones del enunciado, en caso contrario se indica que el incremento debe ser mayor o igual a 5.

En el after statement se actualizan todas las filas de la tabla individuoAuxUpdate en la columna valorViejo con el valor actual de cada fila de la tabla individuo una vez se realizó la actualización, luego de esto se verifica en cada fila de la tabla individuoAuxUpdate si la actualización se trató de un incremento, en caso de que no el trigger no hace nada más, en caso de que sí se elimina de la tabla individuo la fila cuyo código sea igual al código del loop actual, después de esto se añade a la tabla individuo una fila igual a la que se acaba de borrar con excepción de la columna valor, en la cual se añade el valor que tenía antes de la actualización incrementado en 2, luego de esto se verifica si la fila actual del loop tiene 1 o más hijos, en caso de que sí, se hace una búsqueda que devuelve un hijo, este es borrado de la tabla individuo y es insertado de nuevo, las variables código y nombre se dejan igual a como están presentes en la tabla auxiliar, el valor insertado es el valor que tenía antes de la actualización incrementado a la diferencia entre el valor nuevo y el viejo de su padre y restado en 2, su valor padre es dejado en nulo y nro\_hijos en 0, esto se solucionará en el paso posterior para recuperar las conexiones, se iterará sobre toda la tabla individuoAuxUpdate y se actualizarán los valores padre y número de hijos de todas las filas de la tabla individuo para que coincidan con las que tenían antes de la actualización, finalmente se borran todas las filas de individuoAuxUpdate.

**Trigger F**

Para este trigger se necesitará crear una nueva tabla **codigoAux** la cual tendrá una columna codigoViejo y una codigoNuevo. Especificada en el archivo inserts.sql de la carpeta punto 2.

Se hace uso de compound trigger de UPDATE sobre la propiedad código de la tabla individuo. Se definen variables auxiliares, un contador de actualizaciones y un valor i para iterar sobre un arreglo. Se tiene un bloque BEFORE STATEMENT similar al del Trigger C Y D, primero se agregan todos los registros de la tabla individuo en la tabla individuoAux, después de esto se hacen NULL todos los padres de la tabla individuo, esto se hace con propósito de evitar conflictos de clave foránea.

En el bloque BEFORE EACH ROW se agrega a un arreglo de tipo codigoAux el código viejo y el código nuevo del registro afectado.

En el bloque AFTER STATEMENT se sigue un proceso similar al trigger C y D, Iterando sobre todos los registros de la tabla individuoAux.

Primero se verifica que para cada registro el código de su padre aún exista, si es el caso se emite un update a la tabla individuo para devolver el código del padre al registro original, en caso de que el padre no exista se itera sobre los registros afectados; Si se comprueba que el codigoViejo es igual al código del padre del registro viejo, entonces se emite un update en la tabla individuo haciendo el padre igual al codigoNuevo.

Después de hacer esta comprobación se hace una más, si el registro con del codigoViejo en tabla individuoAux tenía un padre no nulo, entonces se verifica que en la tabla de individuo ese padre todavía exista, si es así entonces se emite un update en tabla individuo seteando el padre igual al padre encontrado desde la tabla individuoAux para el registro que posea el código nuevo.

Al final del proceso se eliminan todos los registros de la tabla individuoAux y el trigger estaría concluido.