Informe - Trabajo 1 - Bases de Datos II

Integrantes:
Juan Esteban Cendales Sora
Juan Manuel Pajoy Lopez
Juan Pablo Ortega Medina

Punto 1

Para la solución del primer punto del trabajo se procedió primero a realizar un análisis de las posibles soluciones que se tenían: Se analizó una solución por fuerza bruta, sin embargo era complejo adaptar el código al principal problema propuesto en este punto que era *encontrar los cerdos que dieran la suma óptima a la capacidad del camión o a la cantidad de kilos restante para satisfacer el pedido.* Dentro de las opciones nos dimos cuenta que este problema era muy similar a un famoso problema de programación dinámica nombrado : **0-1 Knapsack Problem**. Donde se tiene una mochila con una capacidad de peso K , unos objetos cada uno con peso Wi y un valor Vi , de esta manera el algoritmo de programación dinámica resuelve el problema de llenar la mochila de la manera más óptima con un enfoque muy eficiente, en nuestro caso tendríamos una capacidad K correspondiente al menor entre : la capacidad del camión o la cantidad de peso restante a satisfacer , cada cerdo tendrá un peso Wi y el valor que tiene será igualmente su mismo peso es decir Wi = Vi.

La programación dinámica se basa en la solución de subproblemas para la solución del problema general , para nuestro caso comenzamos con nuestro peso objetivo *kg_input* que es ingresado por consola, un peso *kg_actual* el cual es el peso que vamos supliendo y comienza en 0 al inicio del programa , también tenemos Y camiones con capacidad MAXIMACAPACIDADKILOS cada uno almacenados en una lista de camiones *array_camiones* el cual se basa de la información de la tabla CAMION y X cerdos con peso PESOKILOS almacenados en una lista de cerdos *array_cerdos_global* el cual se basa en la información de la tabla CERDO. Comenzamos iterando por todos los camiones de *array_camiones* desde el de más capacidad al de menor capacidad y en cada iteración se resuelve un problema tipo Knapsack:

Por cada iteración tenemos una capacidad a maximizar *kg_maximo* que es el menor entre MAXIMACAPACIDADKILOS del camión en el cual se está iterando o el peso restante que sería *kg_input - kg_actual* esto último con el fin de asegurar no enviar más kg de los pedidos.
Una vez se define el *kg_maximo* para esta iteración se procede a seleccionar los cerdos cuyo PESOKILOS sea menor o igual al *kg_maximo* del *array_cerdos_global* , dichos cerdos que cumplan con la condición son almacenados en la lista *array_cerdos.*
Luego se va iterando por cada cerdo en orden desde el cerdo 1 al n siendo n la cantidad de cerdos en *array_cerdos.* Y se va resolviendo una serie de subproblemas que se describen de la siguiente manera.

Paso 1:

Para el primer cerdo en la iteración de *array_cerdos* se resuelve el subproblema de si solo se tuviese el primer cerdo y todo el *kg_maximo* disponible que sería mejor ¿ Tomar o no

tomar el primer cerdo?. También se resuelve el subproblema si solo tuviese el primer cerdo y disponible el *kg_maximo -1* que sería mejor¿Tomar o no tomar el primer cerdo?, así sucesivamente hasta llegar al problema trivial de si solo tuviese el primer cerdo y disponible el *kg_maximo - kg_maximo* que sería mejor¿Tomar o no tomar el primer cerdo? Una vez resuelto todos esos subproblemas pasamos al siguiente paso.

Paso 2:
Si tuviese el primer cerdo y el segundo y todo el *kg_maximo* disponible que sería mejor ¿Tomar o no tomar el segundo cerdo? Si lo toma caería en el subproblema de solo tener el primer cerdo y disponible el *kg_maximo - PESOKILOS* del segundo cerdo, y si no toma el segundo cerdo caería en el subproblema de tener solo el primer cerdo y disponible el *kg_maximo.* Ambos subproblemas ya fueron resueltos en el primer paso. Ahora también se soluciona el problema de si tuviese el primer cerdo y el segundo y disponible *kg_maximo -1* que sería mejor ¿Tomar o no tomar el segundo cerdo? el cual caería también en subproblemas anteriores o en problemas triviales donde ninguno de los dos cerdos cabe según el *kg_maximo* y así sucesivamente se soluciona los casos *kg_maximo-2, kg_maximo-3* hasta *kg_maximo - kg_maximo*.

Paso N:
Esto puede ser generalizado y hacerse si se tuviese el primero ,segundo y tercer cerdo . Una vez solucionado todos los subproblemas relacionados a ese caso se procede a el primero, segundo, tercero y cuarto , así sucesivamente hasta llegar al caso de los n cerdos que tiene la lista *array_cerdos.*

De esta manera lógica se puede hallar al final el peso óptimo a enviar junto con los cerdos que fueron enviados, esto último recorriendo los subproblemas desde la solución final hasta el primer subproblema mirando qué decisiones fueron tomadas y guardando cuales cerdos fueron escogidos en el camino.

Para una mayor comprension del problema recomendamos leer el siguiente articulo puesto que nuestra solucion se baso directamente en la logica implementada en la solucion de este otro problema: https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

Al final de la iteración se imprime la lista de cerdos que fueron enviados en el camión el cual se está iterando y también se eliminan los cerdos enviados de la lista *array_cerdos_global* para continuar con la siguiente iteración.

¿Cuándo parar las iteraciones en los camiones?
Puede ser que se itero satisfactoriamente por todos los camiones o cuando la solución por programación dinámica en una iteración cualquiera da como mejor resultado 0 kg. Esto puede indicar varias cosas:

1. El kg_maximo es igual a 0 puesto que se suple toda la demanda.
2. El kg_maximo es tan pequeño que ningún cerdo puede ya suplir completamente la demanda
3. Los camiones restantes son de tan poca capacidad que no pueden mandar ya ningún cerdo

En caso de que esto suceda con el primer camión quiere decir que *'El pedido no se puede satisfacer'* ya que se está iterando desde el camión de mayor capacidad al de menor

capacidad. En estos casos se deja de iterar por los camiones para dar paso a la impresión de la parte final del informe.

Punto 2

Para la solución del segundo punto se hizo uso de diferentes compound triggers para ejecutar cada punto.

Punto A : Se ve reflejado en el before each row del insert_trigger, y cambia el valor que se haya ingresado en cantidad de hijos por individuo por un valor de cero independientemente del valor que haya intentado ingresar.

Punto B: También se ve reflejado en el mismo insert_trigger, y lo que se hace es en el before each row se asigna la variable global el valor del padre de la fila que va a ser insertada y luego en el after statement se actualiza el número de hijos del padre, se hace de esta manera para evitar el error de que la tabla está mutando cuando se ejecuta el trigger.

Punto C y D: Se ve reflejado en el delete_trigger donde en primera instancia se crean variables para futuros procesamientos en los cuales está un array de individuos donde se guardaran todos los individuos existentes de la tabla INDIVIDUO. En el BEFORE STATEMENT se llena dicha lista y se procede a establecer NULL en el campo PADRE de la tabla INDIVIDUO para poder eliminar el individuo en caso que sea padre de otros individuos , en el BEFORE EACH ROW se guarda en la variable old_codigo el codigo del individuo que se está eliminando. Por último en el AFTER STATEMENT se itera por cada valor comprobando si:

1. **Se encuentra con un hijo del usuario eliminado:** dicho individuo se actualiza su valor PADRE en el array a NULL
2. **Se encuentra con el individuo eliminado:** Si tenía padre se le actualiza en la tabla INDIVIDUO el valor correspondiente de hijos.

Por último, en caso que no sea el individuo eliminado se actualiza en la tabla INDIVIDUO el valor correcto de PADRE correspondiente al valor guardado en el array

Punto E Y F: Este punto se ve reflejado en el trigger update_trigger, este trigger se encarga de ambos casos con el fin de poder permitir actualizaciones en el que se cambia el código y el valor a la vez y poder controlar ese flujo de operaciones de manera centralizada. Primero se definieron ciertas variables globales, la diferencia entre el valor nuevo y viejo si la hay, el código del padre del individuo a actualizar, un booleano que determina si se hizo un aumento o no en el cambio de valor, una lista general de individuos,una lista de hijos del individuo que se está actualizando, un objeto hijo seleccionado que será el hijo que vamos a seleccionar para aumentar el valor restante, una lista de nietos correspondiente a los hijos de dicho hijo al cual se le aumentará el valor restante y un contador. Durante el BEFORE STATEMENT se llenará la lista de individuos al igual que se hizo en el delete_trigger y si el código se está actualizando también se asignará el valor de NULL al campo PADRE de todos los registros de la tabla INDIVIDUO. Durante el BEFORE EACH ROW se asignarán los valores correspondientes a las variables declaradas y en caso que se esté actualizando el valor se validará que si se trata de un aumento este sea mayor o igual a 5 y si es un aumento valido se asignará ahi mismo como :NEW.VALOR el antiguo valor + 2. Por último

en el AFTER STATEMENT si se está actualizando el código entonces se recorrerá por todo el array de individuos y en cada iteración se comprobará si:

1. **Se encuentra con un hijo del usuario actualizado :** En este caso se actualizará el valor de PADRE de dicho individuo en el array al nuevo código asignado.
2. **Se encuentra con el individuo actualizado:** En este caso se actualiza en array el código para que sea el código nuevo para su futura actualización del campo PADRE.

Al final todos los valores de PADRE son reasignados con los cambios a la tabla INDIVIDUOS

Si se está actualizando el valor se verifica que sea un aumento, en caso de serlo se escoge un hijo del individuo actualizado y se busca los hijos de este hijo para guardarlo en el array de nietos. Se procede a eliminar el hijo seleccionado y a reinsertarse con el valor actualizado esto para evitar un llamado recursivo de update_trigger, sin embargo el haber hecho esto hace que los nietos ya no tengan el registro de su padre por lo que se procede a actualizar el hijo seleccionado con el valor correcto de hijos y los nietos el valor correspondiente de su padre.