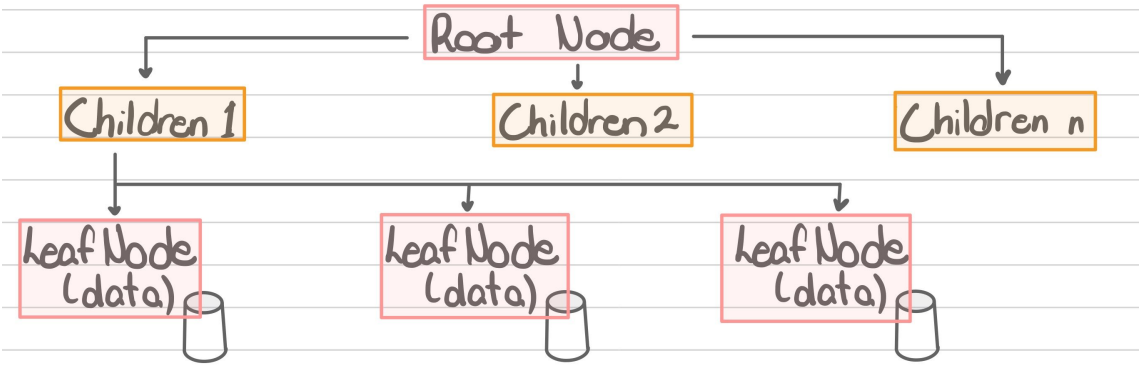
Prueba Corta 5 y 6

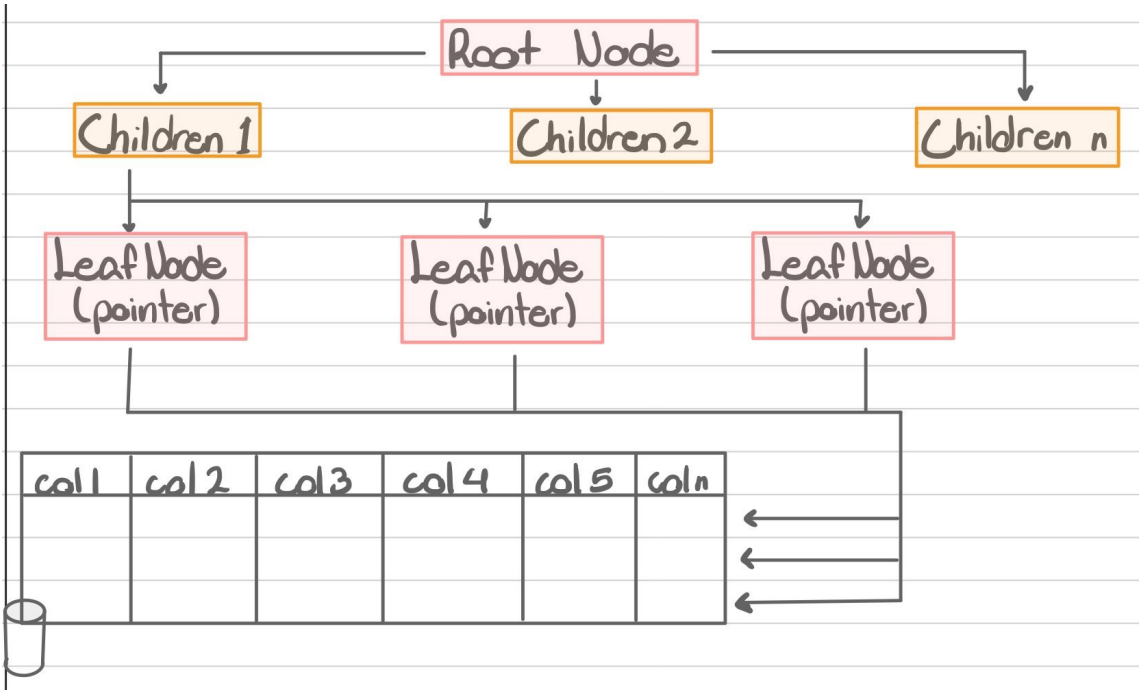
Gabriela Gutiérrez Valverde - 2019024089

Clustered index y la diferencia entre este y un índice non-clustered que utiliza INCLUDE

Un **clustered index** es un tipo de índice que define el orden físico en disco en el cual se ordenan en forma de árbol registros de una tabla dentro de una base de datos. Se puede definir si las columnas utilizadas son poco modificadas. Mientras que el **non clustered index** es un tipo de índice que crea una estructura aparte de los datos de la tabla donde se re-ordenan las columnas. Cuando utiliza el include significa que los punterps a los datos se almacenan en las hojas a más bajo nivel del árbol, lo cual permite que el índice sea más pequeño ya que no es parte del árbol principal. Por lo tanto, la diferencia es que el Clustered Index almacena los datos en las hojas del árbol y el Non Clustered con Include almacena el puntero a los datos. Imagenes de Ilustración para definir mejor la diferencia: **Clustered index:**



Non Clustered Index:



Memory footprint y cómo afecta este la creación de índices.

El **memory footprint** es la cantidad de memoria a la que un programa hace referencia mientras se ejecuta. Es decir, cuando se crea un programa se reserva espacio de memoria que el sistema cree que se va a utilizar, aunque a veces no se esté utilizando está reservado, lo cuál es una afectación, pero esto se podría intentar evitar a la hora de programación. Entre otras afectaciones está el hecho de que si hay índices muy grandes el memory footprint puede crecer mucho. Es normal que el memory footprint utilice mucha memoria (como un 90%), lo que no es normal pero podría suceder es que exista un uso de CPU muy alto.

Mejor forma de organizar los datos y el tipo de base de datos que usaría.

Datos: 40 billones de registros Registros: | Nombre | Datos | | ----- | ----- | | Country | Código de país | | City | Ciudad en específico | | Date | Fecha en que se agregó | | Payload | Documento Json con eventos |

Objetivo: optimizar las búsquedas sobre las columnas country, city y date.

La mejor forma de organizar los datos para incrementar la velocidad de búsqueda sería implementar un estructura de datos B+ Tree ya que los datos se almacenarían en las hojas del árbol y como la búsqueda depende de la profundidad del árbol, con este se puede garantizar que sea de poca profundidad. Además se podría utilizar tanto el código de ciudad como la fecha para ordenar los rangos sobre los que se va a realizar las búsquedas. Para el tipo de base de datos en el caso de que lo que se quiera sea solamente lectura lo mejor sería utilizar una base de datos NoSQL como Mongo o

Firebase, ya que estas se encuentran muy optimizadas para este tipo de lectura. Además el manejo de eventos con los documentos Json suele resultar más sencillo con este tipo de bases ya que la indexan nativamente. Sin embargo, esto es solamente en el caso de lectura ya que este tipo de bases de datos generalmente no tienen una alta consistencia de datos en caso de que se deban hacer otro tipo de transacciones. En el caso de que se requiera hacer muchas operaciones sobre todo el dataset y se requiere tenerlo cargado, si sería mejor que se utilice una base de datos de tipo relacional (SQL), ya que también como en el caso de *Microsoft SQL Server* almacena los datos en un BTree, que resulta muy optimizado para búsquedas pero además permite la alta consistencia.