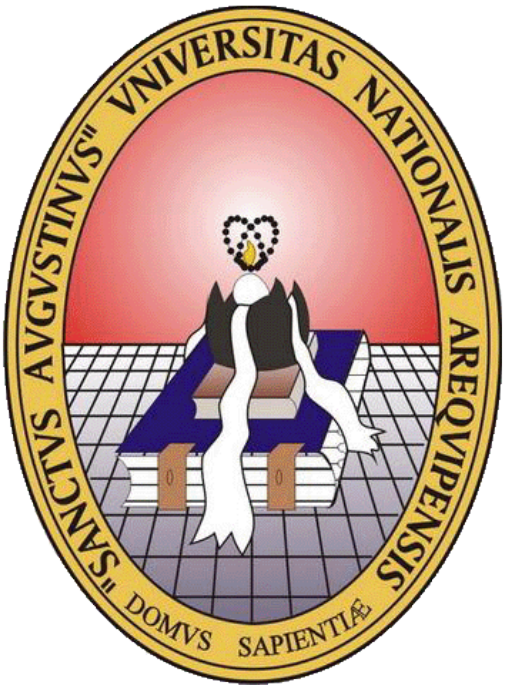
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE PRODUCCIÓN Y SERVICIOS

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN



SISTEMA GESTOR DE BASE DE DATOS

CURSO DE BASE DE DATOS II

ESTUDIANTE:

RUIZ MAMANI, EDUARDO GERMÁN

CUI: 20193061

LINK REPOSITORIO DE GITHUB:

https://github.com/EGRM23/BDII_2023

TURNO: A

AREQUIPA- PERÚ

2023

CONTENIDO

1. CONTEXTO

Crear un SGBD para organizar la asistencia de participantes a un evento de larga duración, parece simple, pero tiene algunas condiciones especiales, las cuales hacen necesario que se deba crear un SGBD especial para este caso. Su nombre es Hicsumito.

Este sistema de asistencia tiene un alto índice de actualizaciones, todos los días se toma asistencia y es necesario editar los datos vacíos, las consultas normalmente son por grupos o secciones, no son muy complejas.

Por otro lado, tiene un porcentaje relativamente alto de eliminaciones, porque el límite de faltas al evento es estricto, cada semana hay nuevas eliminaciones, y es necesario debido a que el sistema de asistencia está relacionado directamente con la base de datos, y es ineficiente que siga apareciendo el nombre de participantes que ya no forman parte del evento, siendo bastantes.

En contraparte, la creación de nuevos participantes solo es alta al comienzo del evento, disminuyendo considerablemente después de las primeras semanas, pero no volviéndose nulo, debido a que aún es posible ingresar hasta pasado el primer mes y en casos excepcionales hasta la mitad de las fechas.

Hicsumito está basado en un encargo que se me hizo a comienzo de año, y aunque la base de datos no era grande, se presentaron algunos problemas (explicados en la siguiente parte) que de haber aparecido en grandes volúmenes de datos hubieran llevado a un gran caos, esta es la razón por la que decidí expandirlo a grandes dimensiones y proponer algunas soluciones respecto a lo que he aprendido.

2. PROBLEMA

Hicsumito necesita procesar datos que van a cambiar constantemente en el tiempo (diariamente) debido a la naturaleza de sus registros, ya que existen un número de columnas igual a la cantidad de días que se tomará asistencia, sí o sí habrán campos vacíos que se llenarán aún hasta finalizar el evento, además no escapan las situaciones especiales donde quizás no haya sesión diaria o no se tome asistencia por alguna razón en especial (feriados no previstos o situaciones externas), y él peor caso donde ingresan nuevos participantes después de haber iniciado el evento, no se les puede poner falta ni presente en esos días que faltaron, solo deben estar "vacíos".

Hay varias ventajas y desventajas entre usar registros de longitud variable o fija. Desde luego que los registros variables serán mejores, sin embargo, creo que es necesario aclarar los inconvenientes presentados:

- Al haber una alta frecuencia de edición de datos diaria, es necesario tener un completo acceso a los registros sin sobrecargar el sistema, en registros variables será complejo, debido a su propio formato, además que al haber edición continua, también se requerirá reestructuración continua de los registros, por otro lado, con registros de longitud fija sería mucho más fácil el acceso y la edición no necesariamente implicaría reestructuración, sin embargo, aquí es donde entra en juego el espacio, ya que estamos hablando de grandes bases de datos, tener registros de longitud fija seria muy MUY **MUY** costoso, teniendo en cuenta que a comienzo del evento no todos los datos estarán llenos y habrá un porcentaje muy grande de vacíos, nuestro espacio disponible sería llenado muy fácilmente... con registros de longitud variable se podría administrar dinámicamente todo ese espacio, con el tiempo crecería de manera uniforme.
- Al haber un porcentaje alto de eliminaciones, es necesario una reestructuración constante de los registros para aprovechar el espacio no usado, desde luego con registros de longitud fija sería más fácil, pero tomando otra vez como referencia el espacio usado… quizás no sea la mejor opción, por el otro lado, los registros de longitud variable administran mejor las eliminaciones y el espacio disponible, incluso esta ventaja se podría usar para tratar de ubicar registros de un mismo lugar en el mismo bloque (aunque desconozco si es posible hacerlo en la vida real, después de todo solo es una opción).

Es por esto que será de tipo esparza y con registros de longitud variable, añadiendo estrategia de optimización. Muchos SGBD no gestionan muy bien el almacenamiento cuando se tiene muchos campos en blanco y generan un consumo alto de memoria.

Algo adicional que aclarar es el tipo de consultas que se hará, normalmente será por rangos al tomar la asistencia por segmentos, pero en algunos casos puede ser personal para editar datos erróneos o comprobar. No todos los SGBD pueden encontrar de manera rápida un rango de datos.

3. OBJETIVOS

3.1. GENERAL

Proponer a Hicsumito como un SGBD capaz de gestionar eficientemente los datos de asistencia de un evento de larga duración, teniendo grandes volúmenes de datos y condiciones especiales.

3.2. ESPECÍFICOS

- Almacenar de manera óptima gran cantidad de datos de asistencia esparza con longitud variable, editables a lo largo del tiempo.
- Proponer un sistema de eliminación según el número de faltas de los asistentes (un parámetro elegido por la organización general)
- Organizar los registros mediante una estructura que pueda ubicar un rango de ellos por el índice y otros atributos como la sección y grupo, usando eficientemente la RAM sin hacer una búsqueda muy profunda.

4. ARQUITECTURA

4.1. GESTOR DE ALMACENAMIENTO

Todos los datos serán guardados dentro del disco, la estructura seguirá siendo la misma que en las primeras clases (platos, superficies, pistas, sectores) y obviamente los bloques.

Los registros se guardarán como tipos de longitud variable, debido a las grandes ventajas de ellos, además se toma en cuenta la posibilidad de reestructurar los datos, después de las eliminaciones, mediante alguna técnica de optimización (todo esto dentro de lo que es posible hacer a nivel físico en el disco).

Para poder encontrar los registros de manera eficiente, se propone el uso de una nueva estructura (basada en las estructuras mostradas en clase). Explicada en una de las fuentes de la bibliografía, la estructura TAG–Tree es una estructura que mezcla árboles B+ con tablas Hash ¿Por qué se decidió usar esta estructura? Debido al tipo de consultas que se tienen:

Si bien se harán consultas por rango, no todos los id's estarán seguidos o antecedidos de id's de su mismo grupo, el ingreso de los participantes no tiene un orden definido, por ejemplo, fácilmente se podría usar un B+ tree y seleccionar los id's del 1 al 10, pero nada asegura que sean todos del grupo A (debido al orden de ingreso), en los datos de cada registro el grupo es un atributo y también lo es su propio id dentro del grupo, que es diferente al id general.

También existe la necesidad de hacer consultas individuales, fácilmente se podría hacer por el id de cada participante, sin embargo, poniéndose en un caso real, es muy poco probable que cada uno conozca su propio id general, desde luego que no es imposible, pero la mayor clase de consultas individuales, tendrán 2 atributos como referencia, el grupo y su id por grupo (puede ser algún atributo personal dentro del grupo).

Evaluando estas 2 condiciones, se evaluó la capacidad de usar B+ tree y tablas Hash por separado, pero el consumo en recursos sería algo alto, se indagó y una estructura más eficiente sería el TAG – Tree, funcionando de la siguiente manera, primero se hace una

tabla Hash con los id's de todos los grupos existentes en la base de datos, luego en cada nodo de los grupos, se incluiría como atributo una estructura B+ tree que contendría los id's por grupo de los participantes. De esta manera las consultas serán más rápidas y eficientes, al comienzo se usan tablas Hash porque la búsqueda por grupos en su gran mayoría siempre será individual, no por grupos (las tablas Hash tienen ventaja en eso), en la segunda parte se usan árboles B+ tree porque ya que se está dentro de un grupo las consultas por rango serán más necesarias a este punto, además también se permite hacer la búsqueda individual en esta estructura.

De las fuentes donde se sacó, se hizo una comparación respecto al tiempo de respuesta e inserción en esta estructura respecto a otras, tomando en cuenta grandes cantidades de datos, y este fue el resultado en ms:

Table 2 Insert and retrieve performance comparison among four data interpolation index structure

Function	B+ Tree	Red-Black Tree	T- Tree
Batch insert	309.2	221.5	210.2
Cross section insert	160.0	120.1	100.6
query	119.6	91.8	49.7

5. DESCRIPCIÓN BASE DE DATOS DE EJEMPLO

No se encontró, o bueno no con las especificaciones necesarias

6. REFERENCIAS

Arreaga Salvatierra, J. H., & Garcia Sojos, W. A. (2022). *Análisis y propuesta tecnológica para el control de asistencia escolar en la Institución Educativa Sueños y Fantasías*.

Arulogun, O. T., Olatunbosun, A., Fakolujo, O. A., & Olaniyi, O. M. (2013). *RFID-based students attendance management system*. http://repository.futminna.edu.ng:8080/xmlui/handle/123456789/8782

Brenes Monge, R., & Rubí Artavia, P. (2000). *Árboles B+ para la administración de archivos indexados y registros de longitud variable*. https://repositorio.una.ac.cr/handle/11056/21393

Lim, T. S., Sim, S. C., & Mansor, M. M. (2009). RFID based attendance system. *2009 IEEE Symposium on Industrial Electronics & Applications*, *2*, 778–782.

Li, X., Ren, C., & Yue, M. (2011). A distributed real-time database index algorithm based on B+ tree and consistent hashing. *Procedia Engineering*, *24*, 171–176. https://doi.org/10.1016/j.proeng.2011.11.2621