Clase AgenteRemoto

Esta clase es el proxy del agente. A traves de la misma el servidor obtendrá entradas de actualización y enviara respuestas. Es, a su vez, producer que alimenta la cola de consultas del servidor. Se maneja en un hilo aparte, dado que va a haber un AgenteRemoto por agente conectado al servidor.

Atributos:

- **Mutex m**: es un mutex que se utiliza para manejar correctamente el envio de respuestas a traves del socket al agente

- **Socket agente**: es el socket que se encuentra conectado al agente. El AgenteRemoto se debe encargar de liberarlo y cerrarlo debidamente.

- **ResolvedorEntradas& blresolvedor**: la interfaz resolvedor entrada nos permite desacoplarnos de lo que sea que haya abajo. El único requisito es que mantenga el invariante de que las entradas solo se resuelven de a una .

- **unsigned id**: un id de cliente remoto, que sirve para identificación

- **ConsultasAgentesServidor& cconsultas**: es una referencia a la cola de consultas de los agentes del servidor. Es la cola que contiene todos los pedidos de actualización que tiene el servidor. La misma debe ser "tread safe."

Métodos públicos :

- **void correr()** : Método que se ejecuta mientras este en ejecución el hilo. Es el encargado de recibir elementos del socket y encolarlos para que responda el servidor

- **void detener_agente()** : Detiene la ejecución del agente remoto. Cierra la conexión, detiene y sincroniza

- **void enviarRespuesta(Respuesta& r)** : Envía la respuesta obtenida del servidor al agente

- **AgenteRemoto(Socket agt, ResolvedorEntradas& rentr, ConsultasAgentesServidor& cons)** : constructor del agente remoto. Recibe el socket activo conectado con el servidor y un resolvedor de entradas

- **~AgenteRemoto()** : destructor de agente remoto. Si esta corriendo, lo detiene. Si esta conectado, lo desconecta

Clase ArchivoDeDatos

Esta clase es la encargada de almacenar todo tipo de registros guardándolos en disco, permitiendo acceder a estos a partir de su id de registro.

Atributos:

- **fstream _archivoPrincipa**l: stream que contiene los registros de tamaño variable en disco.

- **fstream _posRelativas**: stream que contiene las posiciones relativas de los registros del _archivoPrincipal.

- **Id_Registro _ultimoID**: variable numérica que representa la cantidad de registros que se encuentran almacenados.

- **std::string _rutaArchivoPrin**: string que guarda la ruta del archivo principal en disco.

- **std::string _rutaArchivoSec**: string que guarda la ruta del archivo de posiciones relativas.

- **Mutex _mutex**: utilizado para bloquear el acceso a disco para hacer la clase "Thread Safe".

Clase BaseDeDatos

Esta clase es la encargada de resolver las todos los tipos de consultas hechas por un cliente yo agente.

Atributos:

- **Indice <Dimension> _indDimensiones**: es vector dinamico que contiene los indices de ids de registro que se utilizaran para filtrar los datos en la resolucion de una consulta. El tipo Dimension es un std::string.

- **IndiceDeFechas _indFechas**: es un indice especial utilizado para guardar los id de registros y filtrarlos a partir de una gran cantidad de diferentes tipos de rangos.

- **ArchivoDeDatos _archDatos**: clase que se encarga de almacenar los registros en disco y permite acceder a estos a partir de su id cuando sea necesario

- **unsigned _cantDimensionesSimples**: se utiliza para tener la cantidad de dimensiones simples que poseen por parte del Modelo.

Clase ClienteRemoto

Esta clase es el proxy del cliente. A traves de la misma el servidor obtendrá consultas realizadas y enviara respuestas. Es, a su vez, producer que alimenta la cola de consultas del servidor. Se maneja en un hilo aparte, dado que va a haber un ClienteRemoto por cliente conectado al servidor.

Atributos:

- **Mutex m**: es un mutex que se utiliza para manejar correctamente el envío de respuestas a traves del socket al cliente.

- **Socket cliente**: es el socket que se encuentra conectado al agente. El ClienteRemoto se debe encargar de liberarlo y cerrarlo debidamente.

- **ResolvedorConsultas& blresolvedor**: la interfaz resolvedor consultas nos permite desacoplarnos de lo que sea que haya abajo. El único requisito es que mantenga el invariante de que las consultas se deben ejecutar en paralelo.

- **unsigned id**: un id de cliente remoto, que sirve para identificación

- **ConsultasClientesServidor& cconsultas**: es una referencia a la cola de consultas de los clientes del servidor. Es la cola que contiene todos los pedidos de consulta que tiene el servidor. La misma debe ser “tread safe”.

Métodos públicos:

- **void correr()**
  Método que se ejecuta mientras este en ejecución el hilo. Es el encargado de recibir elementos del socket y encolarlos para que responda el servidor

- **void detener_cliente()**
  Detiene la ejecución del cliente remoto. Cierra la conexión, detiene y sincroniza

- **void enviarRespuesta(Respuesta& r)**
  Envía la respuesta obtenida del servidor al cliente

- **ClienteRemoto(Socket cl, ResolvedorConsultas& rcons, ConsultasClientesServidor& cons)**
  constructor del servidor remoto. Recibe el socket activo conectado con el cliente, un resolvedor de consultas y la cola.

- **~ClienteRemoto()**
  destructor de cliente remoto. Si esta corriendo, lo detiene. Si

esta conectado, lo desconecta

Clase ComparadorHechos

Esta clase es la encargada de decidir si un registro en particular debe o no ser filtrado a partir de filtros de Hechos. Construyéndose a partir un Consulta en particular.

Atributos:

- **bool _filtrarHechos**: booleano indicando si debe comprobar el registro a analizar o no, es decir, solo filtra si hay filtros o entradas de hechos.

- **std::vector <unsigned> _indHechos**: Contenedor utilizado para guardar los índices de campo (segun Organizacion) de los hechos que se tiene que filtrar.

- **std::vector <std::string> _valorHechos**: Contenedor utilizado para guardar los valores por los cual se tiene que filtrar los registros.

- **std::string _campoActual**: string utilizado para guardar el campo de registro temporalmente.

- **Utilitario u**: clase utilizada para hacer operaciones y cálculos auxiliares.

Clase ContenedorAgentes

Es una clase que hace de interfaz a un objeto que pueda contener a los agentes remotos. La misma sera aplicada por el controlador del servidor que los contendrá a todos. Sirve como para mitigar el acoplamiento.

Métodos:

- **virtual void agregarAgente(AgenteRemoto agt) = 0**
  Método virtual abstracto que debe implementar cada contenedor.

Clase ContenedorClientes

Es una clase que hace de interfaz a un objeto que pueda contener a los clientes remotos. La misma sera aplicada por el controlador del servidor que los contendrá a todos. Sirve como para mitigar el acoplamiento.

Metodos:

- **virtual void agregarCliente(ClienteRemoto cli) = 0**
  Método virtual abstracto que debe implementar cada contenedor.

Clase ControladorServidor

Esta clase es fundamental. Su rol es controlar el flujo correcto de resolución de consultas y actualizaciones. Se encarga de permitir que el servidor maneje múltiples consultas, como también de frenarlas cuando se requiera de procesar actualizaciones. Contiene un Pool de workers, cada uno con una dedicación a alguna de las tareas ya sean de actualización como de consultas. Implementa interfaces como ResolvedorConsultas, ResolvedorEntradas, ContenedorClientes y ContenedorAgentes, que hace que sus workers solo vean lo que necesiten ver de el.

Es el encargado también de leer la configuración de puertos que va a tener el servidor, de manejar los hilos que se encargan de recibir clientes y agentes. En definitiva, es el encargado de la coordinación de la concurrencia del servidor.

Atributos:

- **ConsultasAgentesServidor centradas**: es la cola que contendrá todos los pedidos de actualización recibidos de los agentes. Dicha cola es "thread safe" y sera compartida con todos los workers que estén en el pool de workers para agentes.

- **ConsultasClientesServidor cconsultas**: es la cola que contendrá todos los pedidos de consultas recibidos de los clientes. Dicha cola es thread safe y sera compartida con todos los workers que estén en el pool de workers para clientes.

- **unsigned int ncons**: es un indicador de la cantidad de consultas en proceso, que es distinto a la cantidad de consultas encoladas. Sirve para poder coordinar las consultas y las actualizaciones.

- **unsigned int nact**: es un indicador de la cantidad de entradas en proceso, que es distinto a la cantidad de entradas encoladas. Sirve para coordinar el proceso de actualización con el de consulta. A diferencia de ncons, nact debe ser o 1 o 0.

- **lclientes clientes**: es la lista que contiene todos los proxy de los clientes conectados. Se utiliza para poder detenerlos y eliminarlos en caso de que se quiera cerrar el servidor.

- **Puerto puerto_clientes**: es el numero de puerto por el cual se conectan los clientes.

- **Puerto puerto_agentes**: es el numero de puerto por el cual se conectan los agentes.

- **lagentes agentes**: es la lista que contiene todos los proxy de los agentes conectados. Se utiliza para poder detenerlos y eliminarlos en caso de que se quiera cerrar el servidor.

- **ResolvedorConsultas& rcons**: contiene una referencia al objeto que se encargue de resolver las consultas.

- **ResolvedorEntradas& rentr**: contiene una referencia al objeto que se

encargue de resolver las entradas.

- **ThreadEntradaAgentes tagentes**: este hilo se encargara de recibir todas las conexiones ingresantes de los agentes.

- **ThreadEntradaClientes tclientes**: este hilo se encargara de recibir todas las conexiones ingresantes de los clientes.

- **PoolClientes poolclientes**: este es el pool de workers que minara de la cola de consultas de clientes.

- **PoolAgentes poolagentes**: este es el pool de workers que minara de la cola de consultas de agentes.

- **Mutex m**: este mutex es el que se utilizara para coordinar las actualizaciones y las consultas.

Metodos publicos :

- **Respuesta resolverEntrada(Consulta& entrada)**

Este método es el encargado de resolver las actualizaciones. Si hay consultas resolviéndose en el momento de la llamada, este método quedara bloqueado hasta que pueda realizar la actualización.

- **Respuesta resolver(Consulta& entrada)**

Este método es el encargado de resolver las consultas. Si hay entradas resolviéndose en el momento de la llamada, este método quedara bloqueado hasta que pueda realizar la consulta. Sin embargo este método no se bloqueara si otras consultas están en proceso, permitiendo la resolución de consultas de forma concurrente.

- **ControladorServidor(ResolvedorConsultas& cons, ResolvedorEntradas& rent, Puerto pclientes, Puerto pagentes)**

Este es el constructor de controladorServidor. Debe recibir a los resolvedores correspondientes y los puertos que serán utilizados por los agentes.

- **bool activo();**

Método utilizado para saber si las entradas siguen activas.

- **void agregarCliente(ClienteRemoto rem)**

El método que permite agregar un cliente remoto a su lista. Es el heredado de la interfaz ContenedorClientes.

- **void agregarAgente(AgenteRemoto rem)**

El metodo que permite agregar un agente remoto a su lista. Es el heredado de la interfaz ContenedorAgentes.

- **void comenzar()**

Se encarga de inicar los hilos correspondientes a los que escuchan conexiones y a los pools de workers.

- **void detener()**

Se encarga de detener todos los hilos, tanto los workers como los que escuchan conexiones ingresantes, cierra las colas de consultas y realiza tareas varias previas a la destrucción del mismo.

- **~ControladorServidor()**

El destructor del controlador. Si esta en ejecución, libera toda la memoria que corresponda, cerrando conexiones y deteniendo hilos.

Clase HiloResponderConsulta

Es la clase encargada de la resolución de consultas del cliente. Es una clase heredera de hilo que hace las de consumer de la cola de consultas del servidor. A su vez, se encarga de enviar la consulta a traves del socket del cliente requerido.

Atributos:

- **ConsultasClientesServidor& cconsultas**: es una referencia a la cola de consultas del servidor. De acá obtendrá las consultas a resolver. Notar que esta cola debe ser bloqueante en el método pop2.

- **ResolvedorConsultas& resolvedor**: es una referencia al objeto que se encargue de resolver las consultas.

Metodos publicos:

- **HiloResponderConsulta(ConsultasClientesServidor& cconsultas, ResolvedorConsultas& rcons)**

Constructor de HiloResponderconsulta. Recibe las referencias correspondientes a la cola de consultas y al resolvedor.

- **void correr()**

Es el método llamado por el callback del hilo. Es el encargado de sacar consultas de la cola, pedir la respuesta y enviarla a traves del ClienteRemoto correspondiente.

Clase HiloResponderEntrada

Es la clase encargada de la resolución de entradas del agente. Es una clase heredera de hilo que hace las de consumer de la cola de entradas del servidor. A su vez, se encarga de enviar la respuestas a traves del socket del agente pertinente.

Atributos:

- **ConsultasAgentesServidor& centradas**: es una referencia a la cola de entradas del servidor. De acá obtendrá las entradas a resolver. Notar que esta cola debe ser bloqueante en el método pop2.

- **ResolvedorEntradas& resolvedor**: es una referencia al objeto que se encargue de resolver las entradas.

Metodos publicos:

- **HiloResponderEntrada(ConsultasAgentesServidor& centr, ResolvedorEntadas& rentr)**

Constructor de HiloResponderEntrada. Recibe las referencias correspondientes a la cola de entradas y al resolvedor.

- **void correr()**

Es el método llamado por el callback del hilo. Es el encargado de sacar consultas de la cola, pedir la respuesta y enviarla a traves del AgenteRemoto correspondiente.

Clase IndiceDeFechas

Esta clase es la encargada de funcionar como un índice para las fechas, guardando los id de registros, permitiendo que los ids sean recuperados por distintos tipos de rangos para las fechas.

Atributos:

- **MapaDeFechas _fechas**: mapa que guarda conjuntos de id de registros los valores de fechas que haya.

- **M_Fechas m_fechas**: encargada de manejar y transformar formatos de fechas para que sea fácil de recuperar registros a partir de una fecha simple o compuesta.

Clase PoolAgentes

Es una clase que tiene una determinada cantidad de workers que consumen de la cola de agentes. No es mas que una forma de encapsular una lista que contiene N instancias de HiloResponderEntrada.

Sus únicos métodos, detener() e iniciar() se encargan de detener e iniciar los hilos que contenga en su lista.

Clase PoolClientes

Es una clase que tiene una determinada cantidad de workers que consumen de la cola de clientes. No es mas que una forma de encapsular una lista que contiene N instancias de HiloResponderConsulta.

Sus únicos métodos, detener() e iniciar() se encargan de detener e iniciar los hilos que contenga en su lista.

Clase Servidor

Esta clase es la que se maneja desde el programa principal. La clase contendrá a la base de datos, los verificadores y al controlador. En si, es la clase que nuclea las funcionalidades necesarias para el funcionamiento.

Su función es básicamente iniciar la base de datos, los verificadores y el controlador. A su vez, también transmite las consultas y actualizaciones a la base de datos, haciendo previa verificación de las mismas.

Métodos públicos :

- **Respuesta resolverEntrada(Consulta& entrada)**

Es el método que realiza las actualizaciones. Antes de pasarle la consulta a la base de datos, realiza una verificación de la entrada. Si es valida, pasara la entrada a la base de datos. En caso contrario, devuelve un mensaje de error.

- **Respuesta resolver(Consulta& entrada)**

Es el método que realiza las consultas. Antes de pasarle la consulta a la base de datos, realiza una verificación de la misma. Si es valida, pasara la consulta a la base de datos, obteniendo la respuesta a la misma. En caso contrario, devuelve un mensaje de error.

- **bool funcional()**

Evalúa si el servidor esta funcional, es decir si los receptores de conexión son validos

Clase Consulta

Esta clase es la encargada de guardar y administrar la consulta realizada por un Cliente o Agente, para que sea enviada al servidor y sea resuelta por este.

Atributos:

- **Id_Mensaje _id**: Id numérico para identificar unívocamente a un Consulta

- **Filtros _filtros**: contenedor que almacena los filtros y sus valores para una consulta

- **Entradas _entradas**: contenedor que almacena las entradas y sus valores para una consulta.

- **Resultados _resultados**: contenedor que almacena los resultados que se quieren obtener de la consulta.

- **EntradasTabla _xTabla**: contenedor utilizado para guardar el grupo X de dimensiones yo hechos para una consulta de Tabla Pivote.

- **EntradasTabla _yTabla**: contenedor utilizado para guardar el grupo Y de dimensiones yo hechos para una consulta de Tabla Pivote.

- **bool _consultaTablaPivote**: booleano que indica si la consulta es de tabla pivote o

es un a consulta normal.

- **Campos &_campos**: contenedor que guarda los campos para una consulta hecha por una

agente.

- **Agregaciones _agregaciones**: contenedor que guarda las agregaciones que tiene que hacerse para en los resultados.

- **bool _consultaDeCliente**: booleano indicando si la consulta es de un cliente, en caso de que sea false la consulta seria de un agente.

- **bool _consultaValida**: booleano indicando si la consulta es valida o correcta en su contenido.

- **static std::string s_nulo**: string utilizado como referencia a string vacía.

Clase Respuesta

Esta clase se encarga guardar todo lo relacionado a la resolución de una consulta ya sea desde un agente o un servidor.

Atributos:

- **Id_Mensaje _id**: Id de la Respuesta que lo identifica unívocamente.

- **size_t _columnas**: indica la cantidad de columnas que tiene tabla de la Respuesta

- **DatosDeRespuesta _datos**: contenedor que guarda todo el contenido de la Tabla

- **Fila _filaActual**: contenedor que guarda el contenido para una fila, solo se utiliza cuando se va generando la respuesta por parte del servidor.

- **std::string _msjInterno**: string que guarda un mensaje para la Respuesta

- **static Campo campo_nulo**: string que para ser conferenciado como string vacío

- **Utilitario u**: se encarga de hacer operaciones y cálculos auxiliares.

Clase Organizacion

Esta clase estática encargada de cargar, guardar el modelo del Archivo, pudiendo consultar sobre sus dimensiones, hechos, y cantidades que hay de estos, es decir, se encarga de resolver toda cuestión que implicase al modelo. El funcionamiento de esta clase es solo de consulta para cuestiones que involucren al modelo de datos.

Atributos:

- **static std::fstream _archModelo**: stream que guarda los datos del modelo en disco
- **static ConjuntoCampos _dimensiones**: contenedor encargado de guardar las dimensiones.
- **static ConjuntoCampos _hechos**: contenedor encargado de guardar los hechos.
- **static vectorCampos _campos**: contenedor encargado de guardar todos los campos del modelo(dimensiones y hechos).
- **static std::string nombreNulo**: string nulo para referenciarlo como un string vacío.
- **static bool _existeCampoEspecial**: booleano indicando si existe alguna Dimensión Especial.
- **static unsigned _indEspecial**: posición que indica en que campo esta la Dimensión Especial.
- **static Utilitario u**: encargado de hacer operaciones y cálculos auxiliares.

Clase Socket

Esta clase es la encarga de la comunicación entre las distintas partes como servidor, agente y cliente. Es la capa mas baja de comunicación que hay en este proyecto.

Atributos:

- **struct timeval _timeout**: estructura para manejar el "timeout" del socket.
- **int _fd**: indica el file descriptor que representa al socket.
- **struct sockaddr_in _direccion**: estructura que guarda la cual se conectara el socket.
- **Puerto _puerto**: indica el puerto por el cual se conectara el socket.
- **bool _conectado**: booleano que indica si el socket esta conectado.
- **bool _enlazado**: booleano que indica si el socket esta enlazado para escuchar conexiones entrantes.

Clase Utilitario

Esta clase es usada para realizar operaciones auxiliares como separar un string en varios, borrar caracteres de un string, convertir un string en entero y al revés.

Atributos:

- **std::vector<size_t> _posiciones**: almacena las posiciones del string donde se encuentra el caracter separador.

- **std::string s_anterior**: almacena el string anterior a una separación, para no tener que recalcular las posiciones del carácter separador en el string.

- **char sep_anterior**: almacena el carácter usado para separar en la separación anterior.