**Clase: StorageManager**

**Descripción:**

El propósito principal de esta clase es proveernos el método para crear un archivo que contendrá el TableSpace de la base de datos.

**Funciones:**

* void createTableSpace(const char\* nombreBD, const char\* version, const char\* clave, unsigned int cantBloques);

nombreBD = recibe el nombre de la base de datos ejemplo: MyOracleDB.

versión = recibe una cadena para la versión de la Base de Datos, ejemplo: 1.0.

clave = recibe una cadena para la clave o password de la Base de Datos.

cantBloques = recibe la cantidad de bloques que va contener el TableSpace, tomando en cuenta que cada bloque tiene un tamaño por default de 4096 bytes (4 kb).

\*Esta función produce un archivo ubicado en la carpeta donde está el proyecto, ejemplo: /home/mairen/Documentos/TeoriadeBasedeDatosII\_StorageManager/StorageManager

Cuyo nombre es “tablespace.dat”.

\*Al inicio la lista de libres del TableSpace está formada por todos los bloques.

* void createTable(const char\* nombreTabla, unsigned int cant\_campos, InfoMDC \*campos);

nombreTabla = corresponde al nombre de la tabla, con un máximo de 50 caracteres.

cant\_campos = corresponde al número de campos que tiene el arreglo que se incorporaran a la tabla.

campos = corresponde al arreglo de objetos de la clase InfoMDC, es decir los campos que pasaran a formar parte de la tabla.

**Clase: Block**

Header header

Block(unsigned int blockID, unsigned int ant, unsigned int sig, const char \*type)

virtual void escribir() = 0

virtual unsigned int getEspacioDisponible() = 0

**Descripción:**

Esta clase instancia a un bloque vacío con su Header, en el constructor recibe de parámetro los atributos de la Clase Header, por lo tanto la clase Block tiene un atributo header de la Clase Header.

**Funciones:**

* void escribir();

Esta función escribe el bloque en el archivo, es decir solo la información que tenga el objeto de la clase que herede de Block al momento de llamar a la misma función. Llama a la función actualizarPrimerLibre() de la clase System Block.

* unsigned int getEspacioDisponible();

Esta function devuelve el espacio disponible que contiene un bloque.

**Clase: Header**  
 unsigned int blockID  
 unsigned int sig  
 unsigned int ant  
 char type[4]

**Descripción:**

El blockID representa la ubicación del bloque en el disco como si el disco fuera un arreglo de bloques de 4096 bytes. Debido a lo anterior los atributos sig y ant nos permiten tratar este arreglo de bloques como una lista doblemente enlazada para una mejor manipulación.

El atributo type tendrá el siguiente dominio de datos:

SB=System Block

BB=Blank Block

MDB=Metadata Block

MDCB=Continuación de Metadata Block

DB=Data Block

VB=Varchar Block

\*Para cada uno de los atributos anteriores de la clase Header, la clase Block contiene los sets y gets de dichos atributos.

**Clase: SystemBlock hereda de la clase Block**

InfoSB info

SystemBlock()

**Descripción:**

Esta clase nos permite instanciar el bloque de sistema correspondiente al archivo del tablespace de la base de datos, como se explica en la sección de la clase StorageManager.

El constructor de esta clase hala del disco la información correspondiente al bloque de sistema en el atributo info así como el Header de este bloque en particular.

**Funciones:**

* unsigned int getPrimerLibre();

Esta función corresponde al get del atributo primer libre de la clase InfoSB para la cual la clase SystemBlock contiene un atributo de dicha clase.

* void actualizarPrimerLibre();

Después de haber utilizado la función getPrimerLibre es necesario llamar a la función actualizarPrimerLibre debido a que dicha función nos ayudara a actualizar la lista de libres por lo tanto esta función acomodara aquellos bloques que se vean afectados en el proceso de elegir el bloque libre que se pidió anteriormente.

**Clase: InfoSB**

char nombreBD[32]

char version[8]

char clave[16]

unsigned int primerBMD

unsigned int ultimoBMD

unsigned int primerLibre

**Descripcion:**

nombreBD coincide con el parámetro nombreBD de la función createTableSpace de la clase StorageManager asi como versión y clave.

primerBMD corresponde al blockID del primer bloque de Metadata es decir la primera tabla en crearse

ultimoBMD corresponde al blockID del último bloque de Metadata.

primerLibre corresponde al blockID del primer boque libre de la lista de libres.

\*Para cada uno de los atributos anteriores de la clase InfoSB, la clase SystemBlock contiene los sets y gets de dichos atributos.

**Clase: Metadata hereda de la clase Block**

Metadata(unsigned int blockID, char\* nombreTabla, unsigned int cant\_campos):

Este constructor nos permite mantener un bloque de metadata en memoria para luego escribirse.

Metadata(unsigned int blockID):

Este constructor lee del archivo y carga en memoria los atributos correspondientes al bloque con dicho ID.

InfoMD info

**Funciones:**

* unsigned int getFreeSpace();

Esta función devuelve el blockID de un bloque de datos con espacio libre en caso que no existiese un bloque de datos libre retorna 0.

* InfoMDC readCampo(unsigned int index);

Devuelve el campo de una tabla correspondiente al index de parámetro, es decir una función específica para un determinado bloque de Metadata.

**Clase: InfoMDC**

unsigned short tipo\_campo

char nombre\_campo[30]

unsigned int escala

unsigned int precision

bool nulls

bool PK

char DEFAULT[30]

unsigned int inicio\_varchar (Ignorado por el QA)

unsigned int final\_varchar (Ignorado por el QA)

El tipo\_campo corresponde a una enumeración:

1. Int
2. Doble
3. Char
4. Varchar
5. Bool

* void adjuntarBloqueDatos(unsigned int blockID);

Esta función agrega el blockID, es decir el bloque de datos a la lista de bloque de datos que contiene un bloque de metadata.

**Clase: InfoMD**

char nombreTabla[50]

unsigned int cant\_campos

unsigned int cont\_MD

unsigned int inicio\_BD

unsigned int final\_BD

**Descripción:**

nombreTabla contiene el nombre de la tabla correspondiente al bloque de metadata.

cant\_campos es equivalente a la cantidad total de campos de una tabla, es decir que incluye la cantidad de campos que contienen los bloques de continuación de metadata si este es el caso.

cont\_MD corresponde al blockID del primer bloque de continuación de metadata si no existe es igual a 0.

inicio\_BD corresponde al blockID del primer bloque de datos de una tabla sino existe es igual a 0.

final\_BD corresponde al blockID del último bloque de datos de una tabla sino existe es igual a 0.

\*Para cada uno de los atributos anteriores de la clase InfoMD, la clase Metadata contiene los sets y gets de dichos atributos.

**Clase:** **MetadataContinuo hereda de la clase Block**

MetadataContinuo(unsigned int blockID):

Este constructor lee del archivo y carga en memoria los atributos correspondientes al bloque con dicho ID.

MetadataContinuo(unsigned int blockID, unsigned int blockIDMD):

Este constructor nos permite mantener un bloque de metadata en memoria para luego escribirse.

InfoMDC info

**Funciones:**

* InfoMDC readCampo(unsigned int index);

Devuelve el campo del bloque de MetadaContinuo correspondiente al index de parámetro, es decir una función específica para ese bloque o en otras palabras no considera los demás bloques de MetadataContinuo que puedan existir.

**Clase:** **InfoMDC**

unsigned int cant\_campos

unsigned int blockIDMD

**Descripcion:**

cant\_campos es equivalente a la cantidad de campos que contiene un bloque de MetadataContinuo.

blockIDMD es equivalente al blockID del bloque de Metadata al que corresponde un determinado bloque de MetadaContinuo

\*Para cada uno de los atributos anteriores de la clase InfoCMD, la clase MetadataContinuo contiene los sets y gets de dichos atributos.

**Clase: Data hereda de la clase Block**

Data(unsigned int blockID, unsigned int blockIDMD):

Este constructor nos permite mantener un bloque de data en memoria para luego escribirse.

Data(unsigned int blockID):

Este constructor lee del archivo y carga en memoria los atributos correspondientes al bloque con dicho ID.

InfoD info

**Funciones:**

* void insertRecord(InfoReg reg);

Inserta el parámetro reg como un registro en el bloque de data. La parte de la definición de un registro se encuentra al final de este documento.

* InfoReg selectRecord(unsigned int index);

Retorna el registro correspondiente al index en ese bloque de datos solo considera los registros activos.

* void updateRecord(InfoReg reg, unsigned int index);

Actualiza el registro correspondiente al index en ese bloque de datos.

* void deleteRecord(unsigned int index);

Elimina el registro correspondiente al index en ese bloque de datos.

**Clase: InfoD**

unsigned int blockIDMD

unsigned int cantRegFisicos

unsigned int cantRegActivos

**Descripción:**

blockIDMD es el blockID del bloque de Metadata al cual esta enlazado dicho bloque de data.

cantRegFisicos es la cantidad de registros que contiene un bloque de data contando los que están marcados como eliminados.

Cant RegActivos es la cantidad de registros que contiene un bloque de data sin contar los que están marcados como eliminados.

\*Para cada uno de los atributos anteriores de la clase InfoD, la clase Data contiene los sets y gets de dichos atributos.

**Clase: InfoReg**

unsigned short nulos

unsigned int tam

unsigned char tombstone

unsigned char\* contentReg;

**Descripción:**

nulos es un mapa de bits que indica que campos son nulos y cuáles no por lo que una tabla no puede guardar más de 16 campos.

tam es el tamaño del arreglo de “bytes” contentReg.

tombstone es un mapa de bits de un byte en el cual los dos bit menos significativos representan de izquierda a derecha un bit para indicar si el registro es parcial(0)/entero(1) y el ultimo bit para indicar si el registro esta activo(1)/no activo(0)

contentReg se recibe de parámetro en el siguiente formato, por ejemplo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cod | Nombre | | Edad | Direccion | |
| 1 | 5 | Maria | 27 | 57 | Los Castaños |
| Int | Varchar | | Int | Varchar | |

Con lo anterior se puede observar que a los campos de tipo varchar se les pre-concatena el size en un byte.

\*Para cada uno de los atributos anteriores de la clase InfoReg esta misma contiene los sets y gets de dichos atributos.

En el caso de los varchars se manera un nuevo tipo de bloque que contendrá varchars correspondientes a un campo de un registro de una tabla por lo que siguiendo con el ejemplo anterior este será transformado a:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | BIDV | Index | 27 | BIDV | Index |

Este proceso será previamente a insertar o actualizar el registro.

\*BIDV = El blockID del bloque de varchars donde se encuentra ese campo de ese registro.

\*Index = La posición en ese bloque de varchars donde se encuentra ese campo de ese registro.

**Funciones:**

* unsigned char\* readCampo(unsigned int index);

Devuelve un arreglo de “bytes” que corresponden a un determinado campo de un registro cuyo index es el de parámetro.

**Clase: Varchar hereda de la clase Block**

Varchar(unsigned int blockID, unsigned int blockIDMD):

Este constructor nos permite mantener un bloque de varchars en memoria para luego escribirse.

Varchar(unsigned int blockID):

Este constructor lee del archivo y carga en memoria los atributos correspondientes al bloque con dicho ID.

InfoV info

**Funciones:**

* unsigned int insertVarchar(unsigned char\* varchar);

Inserta el parámetro varchar en el bloque de varchars y devuelve la posición de ese varchar en el bloque.

* unsigned char\* selectVarchar(unsigned int index);

Retorna el varchar correspondiente al index en ese bloque de varchars.

* void updateVarchar(unsigned char\* varchar, unsigned int index);

Actualiza el varchar correspondiente al index en ese bloque de varchars.

**Clase: InfoV**

unsigned int blockIDMD

unsigned int cant\_varchars

**Descripción:**

blockIDMD es el blockID del bloque de Metadata al cual está vinculado dicho bloque de varchars.

cant\_varchars es la cantidad de varchars que contiene el bloque.

\*Para cada uno de los atributos anteriores de la clase InfoV, la clase Varchar contiene los sets y gets de dichos atributos.