PROTOTIPO DE MANEJADOR DE BASE DE DATOS

Se desea implementar un sistema manejador de base de datos que permita crear y mantener tablas, almacenar datos en ellas y realizar ciertas operaciones sobre los datos.

Una **base de datos** es un conjunto de tablas que poseen nombres únicos que permiten identificarlas. Una **tabla** es, desde el punto de vista lógico, un conjunto de **filas** y **columnas** (**campos**) donde se distribuye la información.

Ejemplo:

Tabla Personas

CI	**Apellido**	**Nombre**
0000001	Castaña	María
2256698	Pérez	Juan
3333444	Filth	Daniel
2123328	Pérez	Laura

Cada columna tiene un nombre que debe ser único dentro de las columnas de la tabla y contiene datos de una misma naturaleza.

Si tomamos como ejemplo la tabla de personas, cada columna representa un atributo de las personas. Luego las filas contienen los datos de cada persona.

Una **tupla** es la colección de datos presentes en una fila de la tabla, considerando el orden en que se presentan los datos. Esto nos lleva a que dos tablas con todas sus columnas iguales pero dispuestas en distinto orden se consideren tablas distintas. En otras palabras, una tabla es un conjunto de tuplas y una tupla es un conjunto de datos, debiendo cumplirse que para una misma tabla todas las tuplas tienen la misma cantidad de datos de los mismos tipos y en el mismo orden. No se permite que en una tabla existan dos tuplas idénticas.

Existen tres formas de **calificar** a una columna de una tabla: NOT EMPTY, PRIMARY KEY y ANY. NOT EMPTY significa que la columna no admite campos vacíos. Es decir no admite campos con valor EMPTY. El calificador PRIMARY KEY puede aplicarse a 0 o 1 columnas de una tabla. PRIMARY KEY indica que los valores de la columna no pueden ser vacíos y son únicos. Entonces, se puede identificar unívocamente a un tupla por el valor del campo correspondiente a la columna PRIMARY KEY. ANY se considera un calificador neutro, es decir, no restringe los valores de la columna. El valor EMPTY se acepta para cualquier columna, independientemente de su tipo de datos, siempre que la columna no esté calificada como NOT EMPTY ni como PRIMARY KEY. Notar que en el ejemplo previo de la tabla de personas, Apellido no podría ser un campo PRIMARY KEY, pero sí el campo CI (de acuerdo a las tuplas del ejemplo).

Por motivos de simplicidad se considerarán sólo dos posibles tipos de datos almacenables en una tabla: strings e integers. Un string es una secuencia de caracteres donde se excluyen los siguientes caracteres: el mayor (>), el menor (<), el igual (=) y el dos puntos (:).

En resumen, el sistema deberá poder almacenar y administrar tablas que cumplan el siguiente esquema:

- Un nombre que identifica a la tabla de manera única
- Las columnas, de las que se conoce su:
  - Nombre (que la identifica dentro de la tabla)
  - Tipo de datos (string, integer)

- Calificador que se aplica a esa columna

Consideraciones Generales

El sistema "**deberá**" implementar de forma eficiente las operaciones sobre la base de datos. En particular, teniendo en cuenta la posible existencia en las tablas de una clave primaria (columna PRIMARY KEY).

En caso de que alguna operación, de las que se describen más adelante, genere una tabla con tuplas repetidas, se deberán eliminar las tuplas repetidas, dejando tan sólo una de éstas. Note que de no hacerlo se estaría violando la definición de tabla.

Si la aplicación de alguna operación, de las que se describen más adelante, deja a las tablas de la base de datos en un estado inconsistente o la operación no está permitida por su especificación o por las reglas antes mencionadas se establecerá una situación de error en la cual:

- Permanecerá invariante el estado de la base de datos
- Se mostrará un mensaje de error adecuado y clarificador

Tipo de datos a manejar:

TipoRet	`enum retorno {OK, ERROR, NO_IMPLEMENTADA};` `typedef enum retorno TipoRet;`

Pueden definirse tipos de datos auxiliares.

Toda operación del sistema debe retornar un elemento de tipo **`TipoRet`**. Si la operación se realizó exitosamente, deberá retornar `OK`; si la operación no se pudo realizar de forma exitosa deberá retornar `ERROR` e imprimir ***un mensaje de error correspondiente al error producido***; y finalmente, si la operación no fue implementada, deberá retornar `NO_IMPLEMENTADA`. En cualquier caso que la ejecución de una operación no sea satisfactoria (retorne ERROR), el estado del sistema deberá permanecer inalterado.

Al comenzar la ejecución del sistema se tendrá **una** base de datos vacía, sin tablas.

El sistema debe permitir realizar las operaciones que detallamos a continuación directamente sobre la línea de comandos, no utilizando un menú.

A continuación se describen las operaciones del sistema.

Operaciones sobre la Base de Datos

CREAR TABLA –

createTable(nombreTabla)

Descripción: Crea una nueva tabla vacía (sin columnas ni tuplas) en la base de datos con nombre: nombreTabla, siempre que no exista ya una tabla con dicho nombre.

Ejemplo. Si se desea crear 2 tablas llamadas Personas y Productos:

createTable (Personas)

createTable (Productos)

TipoRet createTable (char *nombreTabla);

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* existe. • Si *nombreTabla* no se especifica
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

ELIMINAR TABLA –

dropTable(nombreTabla)

Descripción: Elimina la tabla de nombre nombreTabla de la base de datos, si éste existe, y las tuplas que la misma posee.

Ejemplo. Eliminar la tabla Productos:

dropTable (Productos)

TipoRet dropTable (char *nombreTabla);

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe. • Si *nombreTabla* no se especifica
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

OPERACIONES PARA MODIFICAR UNA TABLA:

addCol (nombreTabla, nombreCol, tipoCol, calificadorCol)

Descripción: Agrega a la tabla de nombre nombreTabla, si éste existe, una nueva columna al final de nombre nombreCol, si ésta no existe, tipo tipoCol y calificador calificadorCol. Si la tabla tiene tuplas, el nuevo campo tendrá el valor EMPTY en cada tupla. Por lo tanto, en el caso en que la tabla tenga tuplas no es válido que se agregue un calificador distinto de ANY. Tampoco es válido que calificadorCol sea PRIMARY KEY si existe ya una columna con dicho calificador en la tabla nombreTabla.

Ejemplo. Crear 3 columnas en la tabla Personas llamadas CI, Apellido y Nombre:
addCol (Personas,CI,integer,PRIMARY KEY)
addCol (Personas,Apellido,string,NOT EMPTY)
addCol (Personas,Name,string,ANY)

```
TipoRet addCol (char *nombreTabla, char *NombreCol, char *tipoCol,
char *calificadorCol);
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si *nombreCol* existe o no se especifica.

	• Si *tipoCol* no se especifica o no corresponde. • Si *calificadorCol* no se especifica o no corresponde. • Si la tabla *nombreTabla* tiene al menos una tupla y se agrega un calificador distinto a ANY.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

dropCol (nombreTabla, nombreCol)

Descripción: Elimina de la tabla de nombre nombreTabla, si éste existe, la columna de nombre nombreCol, si éste existe. Si la tabla tiene tuplas, entonces se eliminará de éstas el campo correspondiente a la columna eliminada. Si la tabla tenía una única columna de nombre nombreCol entonces quedará como tabla vacía. Si la tabla tiene varias columnas no puedo eliminar la columna con calificador PRIMARY KEY.

Ejemplo. Eliminar la columna Apellido de la tabla Personas:

dropCol (Personas,Apellido)

TipoRet dropCol (char * nombreTabla, char * nombreCol)

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si *nombreCol* no existe o no se especifica. • Si *nombreCol* es la PRIMARY KEY y la tabla tiene más columnas.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

alterCol (nombreTabla, nombreCol, tipoColNuevo, calificadorColNuevo, nombreColNuevo)

Descripción: Modifica de la tabla de nombre nombreTabla, si éste existe, la columna de nombre nombreCol, si éste existe, quedando esta columna con el nuevo tipo de datos tipoColNuevo, calificador calificadorColNuevo y nombre nombreColNuevo, si éste último no es el nombre de otra columna de la tabla. Si la tabla tiene tuplas, los valores de la columna modificada deberán satisfacer las nuevas características (tipo de dato y calificador). El tipo de datos sólo puede cambiar de integer a string y en este caso se deberá realizar la conversión de tipo de la columna especificada en todas las tuplas de la tabla.

alterCol (Personas,Name,string,NOT EMPTY,Nombre)

TipoRet alterCol (char * nombreTabla, char * nombreCol, char *tipoColNuevo, char *calificadorColNuevo, char *nombreColNuevo)

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si *nombreCol* no existe o no se especifica. • Si *nombreCol* es la PRIMARY KEY y la tabla tiene más columnas.

	• Si *tipoColNuevo* no se especifica o no corresponde. • Si *calificadorColNuevo* no se especifica o no corresponde. • Si *nombreCol* no se especifica
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones para la Edición de Datos

INSERTAR TUPLA –

insertInto(nombreTabla, columnasTupla, valoresTupla)

Descripción: Inserta en la tabla de nombre nombreTabla, si éste existe, una tupla con los valores dados en valoresTupla para las columnas indicadas en columnasTupla, si los nombres de las columnas existen, los valores son del tipo adecuado y satisfacen los calificadores correspondientes a cada columna. Si no se indican todas las columnas se inserta EMPTY en las otras. Por lo tanto, la operación se permite sólo si las columnas que no se indican tienen el calificador ANY. Los nombres de las columnas en columnasTupla y los valores de valoresTupla se separan con el uso del caracter dos puntos (:) y deben corresponderse uno a uno. Esto es, el nombre de columna *i* en columnasTupla con el valor en la posición *i* de valoresTupla. Si la tupla a insertar pertenece a la tabla, la operación no tendrá efecto.

Ejemplo. Insertar tuplas en la tabla Personas:

insertInto (“Personas”, “Nombre:CI”, “Telma:3333111”);
insertInto (“Personas”, “Nombre:CI”, “Jose:2566499”);
insertInto (“Personas”, “Nombre:CI”, “Juan:4232323”);
insertInto (“Personas”, “CI:Nombre”, “1555000:Pepe”);
insertInto (“Personas”, “CI:Nombre”, “2565000:Maria”);

Tabla Personas

Nombre	**CI**
Telma	3333111
Jose	2566499
Juan	4232323
Pepe	1111111
Maria	2565000

```
TipoRet insertInto (char * nombreTabla, char *columnaTupla, char * valoresTupla)
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si columnaTupla no existe. • Si la tabla *nombreTabla* no tiene columnas. • Si el valor de la PRIMARY KEY se repite en otra tupla.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

ELIMINAR TUPLA –

delete(nombreTabla, condicionEliminar)

Descripción: Elimina de la tabla de nombre nombreTabla, si éste existe, todas las tuplas que cumplen la condición condiciónEliminar. En caso de que la condición sea “”, se eliminan todas las tuplas de la tabla. Si ninguna tupla cumple la condición, la operación no tendrá efecto.

El formato de las condiciones es: *columna operador valor* (sin espacios en blanco intermedios). Los operadores a utilizar son: = “igual”, <> “Distinto”, > “Mayor” y < “Menor”.

Por ejemplo, Sexo=Masculino, Edad<18, Código<>20, Apellido>Perez.

Para comparar strings con el operador > o < se utilizará el orden lexicográfico habitual.

El valor EMPTY puede usarse en una condición. La condición c*olumna*=EMPTY resulta verdadera para una tupla si, y sólo si, el valor de la columna en dicha tupla es EMPTY; c*olumna*<>EMPTY resulta verdadera para una tupla si, y sólo si, el valor de la columna en dicha tupla es distinto de EMPTY. En cualquier otro caso, una condición que involucre el valor EMPTY resulta ser falsa. Asimismo, toda condición que no involucre al valor EMPTY resultará falsa al ser instanciada por una tupla que tenga el valor EMPTY en la columna de la condición.

Ejemplo. Borrar tuplas de la tabla Personas:

delete (Personas,Nombre=”Jose”)
delete (Personas,Nombre=”Maria”)

```
TipoRet delete (char * nombreTabla, char * condicionEliminar)
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si la columna dentro de *condicionEliminar* no pertenece a la tabla *nombreTabla o no se especifica*. No se considera error si ninguna tupla cumple la condición. Además se asume que la condición *condicionEliminar* respeta el formato establecido.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

MODIFICAR TUPLA

update(nombreTabla, condicionModificar, columnaModificar, valorModificar)

Descripción: Modifica en la tabla de nombre nombreTabla, si éste existe, el valor de las tuplas en la columna de nombre columnaModificar, si éste existe, que cumplen la condición condiciónModificar. En la columna especificada de las tuplas que cumplen la condición se asigna el valor valorModificar, siempre que este valor sea del tipo adecuado y satisfaga el calificador de la columna especificada. En caso de que la condición sea “”, la operación tiene aplicación sobre todas las tuplas de la tabla.

Ejemplo: Modificar la CI de Pepe.

```
update (Personas,Nombre=”Pepe”,CI,1555000);
```

```
TipoRet update (char * nombreTabla, char * condicionModificar, char
* columnaModificar, char * valorModificar)
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. • Si la columna dentro de *condicionModificar* no pertenece a la tabla *nombreTabla* o no se especifica. • Si la columna dentro de *columnaModificar* no pertenece a la tabla *nombreTabla* o no se especifica No se considera error si ninguna tupla cumple la condición. Además se asume que la condición *condicionModificar* respeta el formato establecido.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

LISTAR ESQUEMA –

printMetadata(nombreTabla)

Descripción: Imprime el esquema de la tabla de nombre nombreTabla, si éste existe. Es decir, imprime el nombre de la Tabla, los nombres de sus columnas en el orden correspondiente, indicando para cada columna su tipo de datos y calificador si lo tuviera.

Ejemplo:

printMetadata (Personas)

Nombre - STRING - NOT_EMPTY
CI - INTEGER - PRIMARY KEY

```
TipoRet printMetadata(char *nombreTabla)
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si nombreTabla no existe o no se especifica.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

LISTAR TABLA –

printDataTable (nombreTabla, ordenadaPor)

Descripción: Imprime las tuplas de la tabla de nombre nombreTabla, si éste existe, ordenados de acuerdo a las columnas especificadas en el parámetro ordenadaPor. Los nombres de las columnas se expresan en el formato $columna_1$:$columna_2$: ... :$columna_n$. Las tuplas se muestran ordenadas por $columna_1$ (de menor a mayor) y si dos tuplas coinciden en el valor de la columna $columna_1$, entonces éstas se ordenan (de menor a mayor) por el campo $columna_2$, y así sucesivamente según los campos especificados, de izquierda a derecha, en ordenadaPor. Si el parámetro ordenadaPor es "", imprime las tuplas en cualquier orden, salvo que la tabla tenga un campo PRIMARY KEY, en cuyo caso las tuplas se imprimen ordenadas (de menor a mayor) según dicho campo.

Ejemplo. Si queremos ver las tuplas de la tabla Personas:

printDataTable (Personas,Nombre)

```
Personas
CI:Nombre
4232323:Juan
```

1555000:Pepe
3333111:Telma

TipoRet printdatatable (char * NombreTabla, char *ordenadaPor)

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nombreTabla* no existe o no se especifica. No es error si no hay columnas o tuplas en *nombreTabla*. En este caso deberá imprimir "no hay tuplas en *nombreTabla*".
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones Adicionales del Sistema

DESHACER – undo()

Descripción: Deshace el efecto de la última operación ejecutada que modifica el estado de la base de datos. En particular, las operaciones de impresión de datos resultan transparentes al sistema ya que no modifican el estado de la base de datos. Si no hay operaciones previas que modifiquen el estado de la base de datos, la operación undo no tiene efecto. Una operación undo no deshace el efecto de una operación undo inmediatamente anterior a ella, sino que refiere a la operación previa a la operación cuyo efecto se deshace con la ejecución del primer undo.

Ejemplo.
undo()

TipoRet undo()

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

REHACER – redo()

Descripción: Rehace el efecto de la operación deshecha inmediatamente antes. La operación redo no tiene efecto si no hay una operación undo ejecutada inmediatamente antes que provoque un cambio de estado en la base de datos. Al igual que para el comando undo, asumimos que en particular las operaciones de impresión de datos resultan transparentes al sistema (no deberían tenerse en cuenta en el proceso undo-redo).

El sistema será capaz de deshacer a lo sumo las últimas 20 operaciones hechas que modificaron el estado del sistema y de rehacer a lo sumo las últimas 20 operaciones deshechas.

Ejemplo.
redo()

TipoRet redo()

Retornos posibles:

OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

COMANDOS EJECUTADOS ERRÓNEAMENTE

Aunque lo siguiente fue expresado al comienzo de este documento, decidimos sintetizar aquí la política de manejo de errores frente a la ejecución de las operaciones del sistema. Cada operación tiene determinadas precondiciones para que funcione correctamente. En caso de que alguna de estas precondiciones no se cumpla la operación debería retornar ERROR (tal cual se señala para cada operación), imprimiendo además por pantalla un texto breve apropiado a cada caso para destacar el tipo de error ocurrido. En cualquier caso que la ejecución de una operación no sea satisfactoria (retorne ERROR), el estado del sistema deberá permanecer inalterado.
Se deberá respetar la modularidad tal como fue vista en clase.

SOBRE LOS CHEQUEOS

Se permite considerar que la sintaxis de la entrada que recibirá el programa es válida. Esto quiere decir que no se requiere la realización de chequeos como los siguientes:
- cadenas de nombres de columnas y valores de tuplas que no respeten el formato establecido (usando el (:) como separador).
- nombres de tablas y columnas que tengan caracteres no permitidos.
- condición en la operación deleteFrom que no respete el formato establecido.

Esto no quiere decir que no se deban chequear las condiciones establecidas para las operaciones en la letra del obligatorio. Por ejemplo, creación de una tabla ya existente, supresión de una columna inexistente, entre otros.

Entrega primera parte

createTable	dropTable	alterCol
addCol	dropCol	update
insertInto	delete	undo
printDataTable	printMetaData	redo

Obligatorio: sin estas operaciones el obligatorio NO SE CORRIGE.
Obligatorio.
Opcional: acumula para nota final.

SE PIDE:

Implementar el sistema utilizando el Software provisto en clase (ZinjaI **20191006**) y los TAD vistos en la materia EDA.

No se aceptará código que no compile correctamente.

Recuerde que el trabajo colaborativo no es aceptado y que es necesaria la aprobación del obligatorio para no reprobar la materia EDA. Se entiende por trabajo colaborativo el utilizar software de compañeros de otro equipo con o sin su consentimiento, dejar que otro grupo utilice su código con o sin su consentimiento/conocimiento, reutilizar código de grupos de años anteriores.

Las operaciones opcionales suman para la nota final siempre que se haya llegado al mínimo de exoneración y se haya aprobado el obligatorio. Sólo se considerarán las operaciones opcionales si las obligatorias no tienen ninguna observación.

La aprobación del obligatorio se logra luego de la defensa individual.

FORMAS Y PLAZOS DE ENTREGA:

La tarea deberá realizarse en grupos de **dos integrantes cómo máximo (y mínimo) con una excepción de UN grupo de tres** estudiantes. Hasta el 27 de setiembre inclusive, tienen tiempo para enviar un mail a trabajositspnuevo@gmail.com con el nombre, CI y mail **de cada uno** de los integrantes del equipo con copia al compañero. De no realizarse la comunicación en tiempo y forma, los grupos serán conformados según el orden de lista. No se admitirán trabajos colaborativos ni código de laboratorios anteriores. En caso de encontrarse, todos los involucrados perderán el obligatorio. La tarea deberá ser entregada en la plataforma EVA y deberá constar de TODOS los archivos utilizados en el proyecto, comprimidos en una carpeta, dicha carpeta tendrá los apellidos de los integrantes del equipo. El trabajo deberá ser realizado en Zinjal versión 20191006. **La fecha límite de entrega de la primera parte del obligatorio es el lunes 17 de octubre a las 19:59 horas**. No se admitirán trabajos entregados luego de esa fecha y hora salvo expresa y previa autorización de la docente.