TRABAJO OBLIGATORIO

SIMULADOR DE UN 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

Nombre	**Apellido**	**CI**
María	Castaña	0.000.001
Juan	Pérez	2.256.698
Daniel	Rodríguez	3.333.444
Laura	Pérez	2.123.328

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 si el campo CI (de acuerdo a las tuplas del ejemplo).

Por motivos de simplicidad se considerarán sólo dos tipos de datos almacenables en una tabla: string e integer. 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, implícito, 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

OPERACIONES PARA MODIFICAR 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 de la base de datos la tabla de nombre *nombreTabla*, y las tuplas que la misma posee, si *nombreTabla* existe.

```
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:

AGREGAR COLUMNA:

–addCol (nombreTabla, nombreCol, tipoCol, calificadorCol)

Descripción: Agrega a la tabla de nombre *nombreTabla*, si ésta existe, una nueva columna al final de nombre *nombreCol*, si ésta no existe. Si la tabla tiene tuplas el nuevo campo tendrá el valor EMPTY en cada tupla. Por lo tanto, en el caso de 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, Nombre, string, ANY)
```

Tabla Personas

CI *(int, PRIMARY_KEY)*	**Apellido** *(string, NOT_EMPTY)*	**Nombre** *(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 de ANY.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

ELIMINAR COLUMNA:

–dropCol(nombreTabla, nombreCol)

Descripción: Elimina de la tabla de nombre *nombreTabla*, si ésta existe, la columna de nombre *nombreCol*, si ésta existe. Si la tabla tiene tuplas, entonces se eliminará de éstas el campo correspondiente a la columna eliminada. Si la tabla posee una única columna de nombre *nombreCol* entonces quedará como tabla vacía.

Ejemplo. Eliminar la columna Apellido de la tabla Personas:

Tabla Personas

CI *(int, PRIMARY_KEY)*	**Apellido** *(string, NOT_EMPTY)*	**Nombre** *(string, ANY)*

dropCol (Personas,Apellido)

Tabla Personas

CI *(int, PRIMARY_KEY)*	**Nombre** *(string, ANY)*

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 PRIMARY_KEY y la tabla tiene más columnas.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

MODIFICAR COLUMNA:

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

Descripción: Modifica de la tabla de nombre *nombreTabla*, si ésta existe, la columna de nombre *nombreCol*, si ésta existe, quedando ésta 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.

Ejemplo. Cambiar el nombre de la columna Nombre por Name y calificador por NOT_EMPTY:

Tabla Personas

CI *(int, PRIMARY_KEY)*	**Apellido** *(string, NOT_EMPTY)*	**Nombre** *(string, ANY)*

alterCol (Personas, Nombre ,string, NOT_EMPTY, Nombre)

Tabla Personas

CI *(int, PRIMARY_KEY)*	**Apellido** *(string, NOT_EMPTY)*	**Name** *(string, NOT_EMPTY)*

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 PRIMARY_KEY y la tabla tiene más columnas. • Si t*ipoColNuevo* 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 ésta 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);
```

```
TipoRet insertInto (char *nombreTabla, chr *columnasTupla, 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

–deleteFrom (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:

deleteFrom (Personas,Nombre=Jose)
deleteFrom (Personas,Nombre=Maria)
```

```
TipoRet deleteFrom (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.

Operaciones entre Tablas

SELECCIÓN:

- selectWhere (nombreTabla1, condición, nombreTabla2)

Descripción: Dado un nombre de tabla *nombreTabla1* y una *condición*, genera una nueva tabla en la base de datos de nombre *nombreTabla2*, si *nombreTabla1* existe y *nombreTabla2* no existe, con las tuplas de la tabla *nombreTabla1* que cumplan la condición. En caso de que la condición sea “”, la operación tiene aplicación sobre todas las tuplas de la tabla. La condición respeta el formato descripto anteriormente para condiciones (ver operación delete).

```
Ejemplo: Copiar la tabla Personas a una nueva tabla llamada Personas2.


selectWhere (Personas2,“”,Personas)
```

`TipoRet selectWhere (char * nomTabla1, char * condicion, char * nomTabla2);`

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nomTabla1* no existe o no se especifica. • Si *nomTabla2* existe o no se especifica. • Si la columna dentro de *condicion* no pertenece a la tabla *nomTabla1* o no se especifica. No se considera error si ninguna tupla cumple la condición. Además se asume que la condición *condicion* respeta el formato establecido.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

PROYECCIÓN :

- select (nombreTabla1, columnas, nombreTabla2)

Descripción: Dado un nombre de tabla *nombreTabla1* y uno o más nombres de columnas de dicha tabla especificados en el parámetro columnas, genera una nueva tabla en la base de datos de nombre *nombreTabla2*, si *nombreTabla1* existe, *nombreTabla2* no existe y si todas las columnas especificadas pertenecen a la tabla de nombre nombreTabla1. La tabla *nombreTabla2* tendrá las tuplas de la tabla *nombreTabla1* sólo con las columnas especificadas, manteniendo el orden establecido en el parámetro *columnas*. Los nombres de las columnas se separan con el uso del caracter dos puntos (:).

```
Ejemplo: Copiar la columa CI de la tabla Personas a una nueva tabla
llamada Cedulas.


 select (Cedulas, CI, Personas);
```

`TipoRet select (char * nomTabla1, char * nomColumnas, char * nombTabla2);`

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si *nomTabla1* no existe o no se especifica. • Si *nomTabla2* existe o no se especifica. • Si la columna dentro de *nomColumna* no pertenece a la tabla *nomTabla1* o no se especifica. No se considera error si ninguna tupla cumple la condición. Además se asume que el listado de columnas en n*omColumnas* respeta el formato establecido.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

JOIN NATURAL:

– join (nombreTabla1, nombreTabla2, nombreTabla3)

Descripción: Dadas dos tablas de nombres *nombreTabla1* y *nombreTabla2*, tales que ambas tengan *exactamente una única* columna en común con el mismo nombre, calificada como PRIMARY KEY, genera una nueva tabla en la base de datos de nombre *nombreTabla3*, si *nombreTabla1* y *nombreTabla2* existen y *nombreTabla3* no existe. La nueva tabla *nombreTabla3* tendrá todas las columnas de la tabla *nombreTabla1* y las columnas de *nombreTabla2*, en ese orden, excepto la que esté calificada como PRIMARY KEY en *nombreTabla2* (de modo de no repetirla). Asimismo, la tabla *nombreTabla3* tendrá exclusivamente las tuplas que satisfagan la igualdad de las claves de ambas tablas.

Ejemplo: El join natural de las tablas Personas y Profesiones (asumimos que esta última fue creada) genera la tabla PersonasProfesiones.

```
join (Personas, Profesiones, PersonasProfesiones)
```

Tabla Personas

CI *(PK)*	**Nombre**
3333111	Telma
4232323	Juan
1555000	Pepe

Tabla Profesiones

CI *(PK)*	**Cargo**
3333111	Dentista
7777777	Escribano
4232323	Ingeniero

Tabla PersonasProfesiones

CI *(PK)*	**Nombre**	**Cargo**
3333111	Telma	Dentista
4232323	Juan	Ingeniero

```
TipoRet join (char * nomTabla1, char * nomTabla2, char * nomTabla3);
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si nombreTabla1 no existe o no se especifica. • Si nombreTabla2 no existe o no se especifica. • Si nombreTabla3 existe. • Si no existe una columna en común entre los nombreTabla1 y nombreTabla2. • Si existen más de una columna en común entre nombreTabla1 y nombreTabla2 No se considera error si ninguna tupla cumple la condición. Además se asume que la condición *condicion* respeta el formato establecido.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

UNIÓN:

— union (nombreTabla1, nombreTabla2, nombreTabla3)

Descripción: Dadas dos tablas de nombres *nombreTabla1* y *nombreTabla2* con igual esquema (nombres de columnas, tipos de datos y calificadores), genera una nueva tabla en la base de datos de nombre *nombreTabla3*, si *nombreTabla1* y *nombreTabla2* existen y *nombreTabla3* no existe. La nueva tabla *nombreTabla3* tendrá el mismo esquema que las otras tablas y las tuplas de ambas.

```
TipoRet union(char * nombreTabla1, char * nombreTabla2, char *
nombreTabla3);
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si nombreTabla1 no existe o no se especifica. • Si nombreTabla2 no existe o no se especifica. • Si nombreTabla3 existe. • Si el esquema de *nombretabla1* es distinto a *nombreTabla2*
NO IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

INTERSECCIÓN:

– intersect (nombreTabla1, nombreTabla2, nombreTabla3)

Descripción: Idem a la descripción de la operación unión pero para la intersección. Es decir, la tabla resultante nombreTabla3 contiene sólo las tuplas comunes a *nombreTabla1* y *nombreTabla2*.

```
TipoRet intersec(char * nombreTabla1, char * nombreTabla2, char *
nombreTabla3);
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si nombreTabla1 no existe o no se especifica. • Si nombreTabla2 no existe o no se especifica. • Si nombreTabla3 existe. • Si el esquema de *nombretabla1* es distinto a *nombreTabla2*
NO IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

DIFERENCIA:

– minus (nombreTabla1, nombreTabla2, nombreTabla3)

Descripción: Idem a las operaciones anteriores pero para la diferencia de la tabla *nombreTabla1* con la tabla *nombreTabla2*. Es decir, la tabla resultante *nombreTabla3* tiene las tuplas que están en *nombreTabla1* pero no en *nombreTabla2*.

```
TipoRet minus(char * nombreTabla1, char * nombreTabla2, char *
nombreTabla3);
```

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si nombreTabla1 no existe o no se especifica. • Si nombreTabla2 no existe o no se especifica. • Si nombreTabla3 existe. • Si el esquema de *nombretabla1* es distinto a *nombreTabla2*
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones para la Impresión de Información

LISTAR TABLA

–printDataTable (nombreTabla)

Descripción: Imprime las tuplas de la tabla de nombre *nombreTabla*, si ésta existe. Los nombres y los valores de las columnas se expresan en el formato $columna_1:columna_2: \ldots :columna_n$. Las tuplas se muestran ordenadas ascendentemente por la PRIMARY KEY. Primero se imprime el nombre de la tabla, luego los nombres de las columnas, separados con (:), y por último las tuplas, cuyos campos se separan también con (:).

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

```
printDataTable (Personas)
```

```
Personas
CI:Nombre
1555000:Pepe
3333111:Telma
4232323:Juan
```

```
TipoRet printdatatable (char *NombreTabla)
```

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

LISTAR TABLAS:

– printTables()

Descripción: Imprime los nombres de las tablas de la base de datos del sistema, ordenados alfabéticamente de menor a mayor.

Ejemplo:

```
printTables()
```

```
TipoRet printTables();
```

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.

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)
```

```
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.

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).

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 cuál 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.

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.

PRIMER ENTREGA

Para esta entrega se considera que la base de datos tiene una sola tabla.

Operaciones obligatorias	**Operaciones Opcionales**
Operaciones sobre la Base de Datos Crear Tabla Eliminar Tabla	
Operaciones para Modificar Tablas Agregar Columna Eliminar Columna	Operaciones para Modificar Tablas Modificar Columna
Operaciones para la Edición de Datos Insertar Tupla Eliminar Tupla	Operaciones para la Edición de Datos Modificar Tupla
Operaciones para la Impresión de Información ListarTabla ListarEsquema	

SEGUNDA ENTREGA

Para esta entrega se deben poder manipular multiples tablas.

Operaciones obligatorias	**Operaciones Opcionales**
Operaciones sobre la Base de Datos Crear Tabla Eliminar Tabla	
Operaciones para Modificar Tablas Agregar Columna Eliminar Columna	Operaciones para Modificar Tablas Modificar Columna
Operaciones para la Edición de Datos Insertar Tupla Eliminar Tupla	Operaciones para la Edición de Datos Modificar Tupla
Operaciones entre Tablas Selección Proyección Unión Intersección	Operaciones entre Tablas Join Natural Diferencia
Operaciones para la Impresión de Información Listar Tabla Listar Tablas ListarEsquema	
	Operaciones adicionales del Sistema Deshacer Rehacer

FORMAS Y PLAZOS DE ENTREGA

La tarea deberá realizarse en grupos de dos o tres estudiantes cada uno. Los grupos deben inscribirse llenando un formulario web hasta el 10 de octubre inclusive, los estudiantes que se anoten de forma individual se les asignará un grupo de forma aleatoria.

El plazo de ***entrega para la primera parte será hasta el viernes 28 de octubre inclusive.*** vía el Campus Virtual.

El plazo de ***entrega para la primera parte será hasta el domingo 20 de noviembre inclusive*** via el Campus Virtual.

En ambos casos deberán además coordinar una defensa para evaluar el trabajo realizado.

Quienes realicen las partes opcionales se les podrá asignar hasta 10 puntos extra en el puntaje del curso.