

OBLIGATORIO 2 EDA 2022

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

Nombre	Apellido	CI
María	Castaña	0.000.001
Juan	Pérez	2.256.698
Daniel	Fill	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 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	<pre>enum _retorno {OK, ERROR, NO_IMPLEMENTADA}; typedef enum _retorno TipoRet;</pre>
----------------	---

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.

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 (string nombreTabla);

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando.
ERROR	• Si <i>nombreTabla</i> existe. • Si <i>nombreTabla</i> 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 (string nombreTabla);

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla</i> no existe.• Si <i>nombreTabla</i> no se especifica
NO_IMPLEMENTADA	<ul style="list-style-type: none">• 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 éste 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 (string nombreTabla, string NombreCol, string tipoCol, string calificadorCol)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla</i> no existe o no se especifica.• Si <i>nombreCol</i> existe o no se especifica.• Si <i>tipoCol</i> no se especifica o no corresponde.• Si <i>calificadorCol</i> no se especifica o no corresponde.• Si la tabla <i>nombreTabla</i> tiene al menos una tupla y se agrega un calificador distinto a ANY.
NO_IMPLEMENTADA	<ul style="list-style-type: none">• 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.

Ejemplo. Eliminar la columna Apellido de la tabla Personas:

dropCol (Personas,Apellido)

TipoRet dropCol (string nombreTabla, string nombreCol)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla</i> no existe o no se especifica.• Si <i>nombreCol</i> no existe o no se especifica.• Si <i>nombreCol</i> es la PRIMARY KEY y la tabla tiene más columnas.
NO_IMPLEMENTADA	<ul style="list-style-type: none">• 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 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.

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

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

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla</i> no existe o no se especifica.• Si <i>nombreCol</i> no existe o no se especifica.• Si <i>nombreCol</i> es la PRIMARY KEY y la tabla tiene más columnas.• Si <i>tipoColNuevo</i> no se especifica o no corresponde.• Si <i>calificadorColNuevo</i> no se especifica o no corresponde.• Si <i>nombreCol</i> no se especifica
NO_IMPLEMENTADA	<ul style="list-style-type: none">• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones para la Edición de Datos

INSERTAR TUPLA – intertInto(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);`

TipoRet insertInto (string nombreTabla, string columnaTupla string valoresTupla)

Retornos posibles:	
OK	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none"> • Si <i>nombreTabla</i> no existe o no se especifica. • Si <i>columnaTupla</i> no existe. • Si la tabla <i>nombreTabla</i> no tiene columnas. • Si el valor de la PRIMARY KEY se repite en otra tupla.
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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 *condicionEliminar*. 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 *columna=EMPTY* resulta verdadera para una tupla si, y sólo si, el valor de la columna en dicha tupla es EMPTY; *columna<>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 deleteTupla (string nombreTabla, string condicionEliminar)

Retornos posibles:	
OK	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none"> • Si <i>nombreTabla</i> no existe o no se especifica. • Si la columna dentro de <i>condicionEliminar</i> no pertenece a la tabla <i>nombreTabla</i> o no se especifica. <p>No se considera error si ninguna tupla cumple la condición. Además se asume que la condición <i>condicionEliminar</i> respeta el formato establecido.</p>
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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 condicionModificar. 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 (string nombreTabla, string condicionModificar, string columnaModificar, string valorModificar)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla</i> no existe o no se especifica.• Si la columna dentro de <i>condicionModificar</i> no pertenece a la tabla <i>nombreTabla</i> o no se especifica.• Si la columna dentro de <i>columnaModificar</i> no pertenece a la tabla <i>nombreTabla</i> o no se especifica <p>No se considera error si ninguna tupla cumple la condición. Además se asume que la condición <i>condicionModificar</i> respeta el formato establecido.</p>
NO_IMPLEMENTADA	<ul style="list-style-type: none">• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones entre Tablas

SELECCIÓN – selectWhere (nombreTabla1, condicion, 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).

TipoRet selectWhere (string nombreTabla1, string condicion, string nombreTabla2)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla1</i> no existe o no se especifica.• Si <i>nombreTabla2</i> existe.• Si la columna dentro de <i>condicion</i> no pertenece a la tabla <i>nombreTabla1</i> o no se especifica. <p>No se considera error si ninguna tupla cumple la condición. Además se asume que la condición <i>condicion</i> respeta el formato establecido.</p>
NO_IMPLEMENTADA	<ul style="list-style-type: none">• 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 y nombreTabla2 no existe. 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 (:).

TipoRet select(string nombreTabla1, string columnas, string nombreTabla2)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla1</i> no existe o no se especifica.• Si <i>nombreTabla2</i> existe.• Si la/s columna/s no existe en nombreTabla1 o no se especifica.
NO_IMPLEMENTADA	<ul style="list-style-type: none">• 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 y tipo de datos, 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 excepto la que esté calificada como PRIMARY KEY (de modo de no repetirla), en ese orden. Asimismo, nombreTabla3 tendrá exclusivamente las tuplas que satisfagan la igualdad de las claves de ambas tablas.

Ejemplo: El join natural de las tabla1 y tabla2 genera la tabla3:

CI es clave (PRIMARY KEY) en las tres tablas.

Nombre	CI	CI	Cargo
Telma	3333111	3333111	Dentista
Juan	8232323	8232323	Ingeniero
Pepe	1555000	7777777	Escribano

Nombre	CI	Cargo
Telma	3333111	Dentista
Juan	8232323	Ingeniero

TipoRet join(string nombreTabla1, string nombreTabla2, string nombreTabla3)

Retornos posibles:	
OK	<ul style="list-style-type: none">• Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none">• Si <i>nombreTabla1</i> no existe o no se especifica.• Si <i>nombreTabla2</i> no existe o no se especifica.• Si <i>nombreTabla3</i> 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 <p>No se considera error si ninguna tupla cumple la condición. Además se asume que la condición <i>condicion</i> respeta el formato establecido.</p>
NO_IMPLEMENTADA	<ul style="list-style-type: none">• Cuando aún no se implementó. Es el tipo de retorno por defecto.

UNION – 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 unionTable(string nombreTabla1, string nombreTabla2, string nombreTabla3)

Retornos posibles:	
OK	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none"> • Si <i>nombreTabla1</i> no existe o no se especifica. • Si <i>nombreTabla2</i> no existe o no se especifica. • Si <i>nombreTabla3</i> existe. • Si los esquemas de <i>nombreTabla1</i> es distinta a <i>nombreTabla2</i>.
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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 intersect(string *nombreTabla1*, string *nombreTabla2*, string *nombreTabla3*)

Retornos posibles:	
OK	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none"> • Si <i>nombreTabla1</i> no existe o no se especifica. • Si <i>nombreTabla2</i> no existe o no se especifica. • Si <i>nombreTabla3</i> existe. • Si los esquemas de <i>nombreTabla1</i> es distinta a <i>nombreTabla2</i>.
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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(string *nombreTabla1*, string *nombreTabla2*, string *nombreTabla3*)

Retornos posibles:	
OK	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	<ul style="list-style-type: none"> • Si <i>nombreTabla1</i> no existe o no se especifica. • Si <i>nombreTabla2</i> no existe o no se especifica. • Si <i>nombreTabla3</i> existe. • Si los esquemas de <i>nombreTabla1</i> es distinta a <i>nombreTabla2</i>.
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • Cuando aún no se implementó. Es el tipo de retorno por defecto.

Operaciones para la Impresión de Información

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	• Si no hay tablas para imprimir.
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(string 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 *columna₁:columna₂: ... :columna_n*. Las tuplas se muestran ordenadas por *columna₁* (de menor a mayor) y si dos tuplas coinciden en el valor de la columna *columna₁*, entonces éstas se ordenan (de menor a mayor) por el campo *columna₂*, 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)
```

TipoRet printdatatable (string NombreTabla, string ordenadaPor)

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

	No es error si no hay columnas o tuplas en <i>nombreTabla</i> . En este caso deberá imprimir “no hay tuplas en <i>nombreTabla</i> ”.
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Si se pudo ejecutar exitosamente el comando.
ERROR	
NO_IMPLEMENTADA	<ul style="list-style-type: none"> • Cuando aún no se implementó. Es el tipo de retorno por defecto.

TRANSACCIONES:

En base de datos existe el concepto de “**transacción**” que consiste en la ejecución conjunta de una serie de operaciones. Entre otras propiedades, la transacción debe ejecutarse completamente o no debe ejecutarse ninguna de sus operaciones.

Para simular transacciones el sistema contará con las operaciones `beginTransaction` y `endTransaction`. Las operaciones que se encuentren dentro de un `beginTransaction` y `endTransaction` forman parte de una transacción. Por lo tanto, si alguna de las mismas no pudiera efectuarse debería detenerse la ejecución de la transacción y volver al estado previo al comienzo de la misma. Una transacción tendrá como máximo 20 operaciones del sistema.

Al finalizar exitosamente una transacción el sistema perderá la capacidad de deshacer el efecto de todas las operaciones previas del sistema, incluyendo las de la transacción.

Las operaciones undo y redo no deberían ocurrir dentro de una transacción. Pero si esto sucediera, se producirá el mismo efecto que el de una operación que no pudo efectuarse dentro de la transacción.

El sistema deberá permitir la ocurrencia de transacciones anidadas, esto es, transacciones que puedan tener como operaciones a otras transacciones. La falla de una operación dentro de una transacción no sólo causa la falla de dicha transacción sino también de toda transacción que la contenga.

COMENZAR TRANSACCIÓN – `beginTransaction()`

Descripción: Comienza una transacción.

Ejemplo.

```
beginTransaction()
```

TipoRet `beginTransaction()`

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.

FINALIZAR TRANSACCIÓN – `endTransaction()`

Descripción: Finaliza la transacción.

Ejemplo.

```
endTransaction()
```

TipoRet `endTransaction()`

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.

PLAZO DE ENTREGA

La tarea se entregará y defenderá el **martes anterior al comienzo de parciales**.