Práctica 3

Archivos Binarios y ODBC

Tarea

> Implementar y manejar en C la estructura de tabla de la base de datos de manera que se puedan manejar algunos datos localmente.

> **Parte 1 [50% de la práctica]**: Implementar una serie de funciones que manejen la estructura de tabla, almacenada en un **fichero binario (1 tabla = 1 fichero binario)**. Podéis encontrar las cabeceras de estas funciones en los archivos **table.h.** Implementar una serie de funciones que permitan la creación de un indice. Podéis encontrar las cabeceras de estas funciones en los archivos **index.h**.

> **Parte 2 [50% de la práctica]**: Implementar una serie de programas que usen la conexión con la base de datos a través de las librerías de ODBC y las funciones implementadas en la parte 1 para integrar datos almacenados en local y datos que están almacenados en el servidor de la base de datos.

Parte 1.a

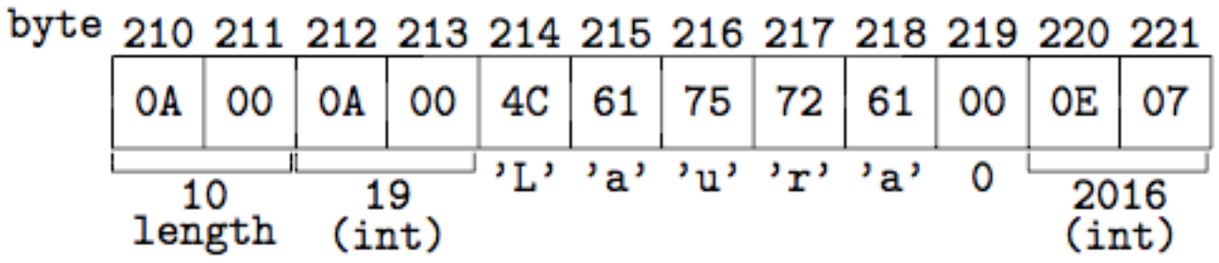
Archivos binarios

1 tabla = 1 fichero binario

Una tabla de la base de datos está compuesta por:

> **La cabecera de la tabla**: contiene el número de columnas que componen la tabla y los tipos de datos de cada una de las columnas. Los tipos de datos están definidos en el archivo **types.h** así como algunas funciones para comparar valores, imprimir valores o parsearlos dependiendo del tipo de dato de la columna. **Atención!, usad estas funciones en vuestras implementaciones.**

> **Una lista de registros, es decir, los datos**:



Parte 1.a

Archivos binarios

Cabecera de la tabla

- Supongamos, ya que estamos trabajando con ficheros binarios, que conocemos cuanto ocupan en memoria cada uno de los tipos de datos que vamos a utilizar:

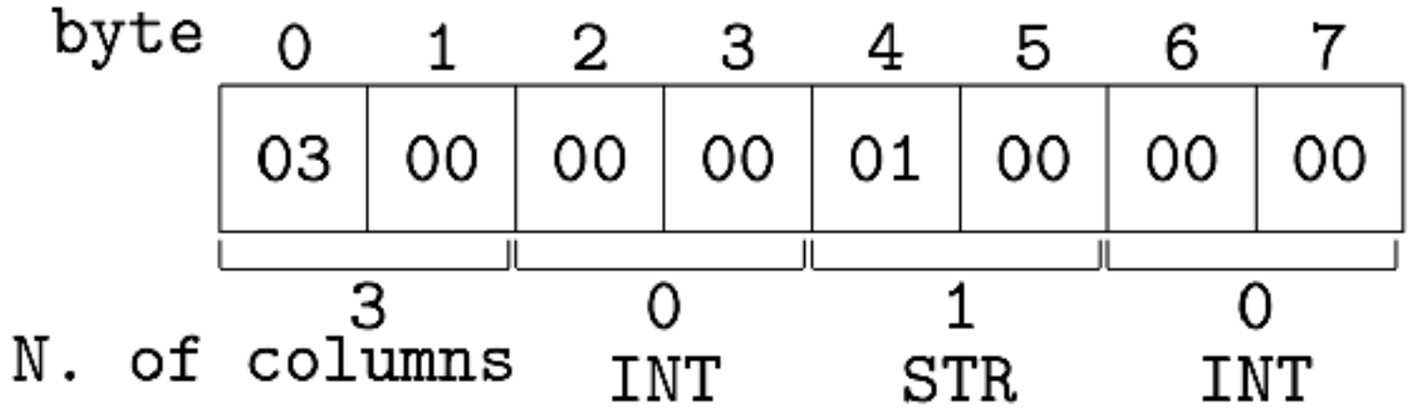
Tipos de datos	Descripción	Tamaño en memoria
int	Cantidad Entera	2 bytes
char	Carácter	1 byte
double	Real con doble precisión	8 bytes
long int	Entero largo	4 bytes
void	Valor genérico o nulo	-

- Nuestros tipos de datos INT, STR, LLNG, DBL están definidos dentro de un tipo enumerado en el fichero **type.h**. Un tipo enumerado es un conjunto de **constantes enteras con nombre**. Es decir, cuando nos referimos a INT en realidad nos estamos refiriendo al valor 0 y cuando nos referimos a STR nos estamos refiriendo al valor 1. Los tipos enumerados se utilizar por simplificación en expresiones de indización y como operando en operaciones aritméticas. **Mirad las funciones definidas en el archivo type.c para entender como trabajar con un tipo enumerado.**

Parte 1.a

Archivos binarios

Cabecera de la tabla



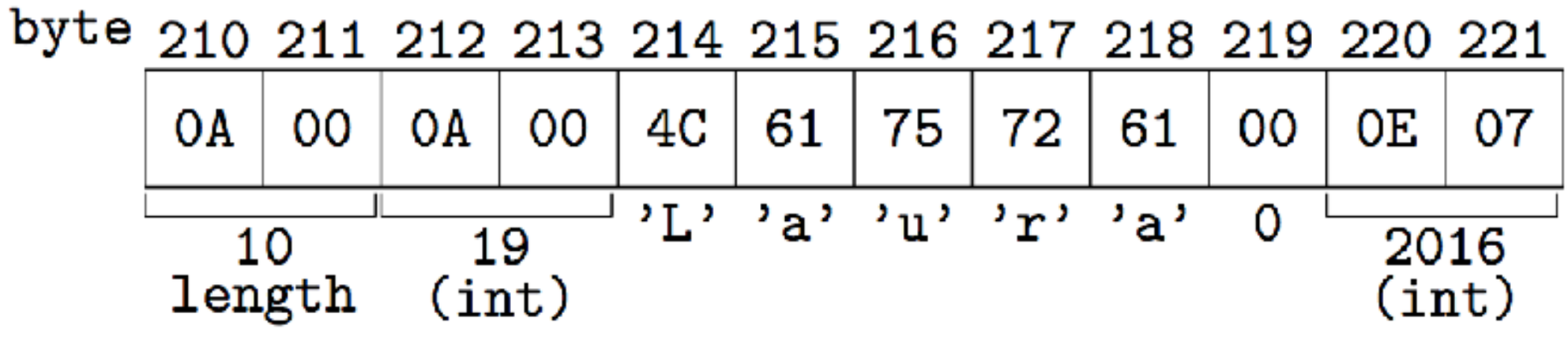
En la función de creación de la tabla (**table_create**) lo que haremos será crear el fichero binario y escribir en el fichero binario la cabecera de la tabla.

Cuando una tabla se abre (**table_open**) crearemos en memoria la estructura de tabla con la información que necesitemos para su manejo, como el número de columnas, el listado de tipos de datos y el puntero (path) al fichero que almacena la tabla. **Ojo!, la lista de registros con los datos no se almacenan en memoria dinámica. Los datos se encuentran en el fichero.**

Parte 1.a

Archivos binarios

Lista de registros = Datos



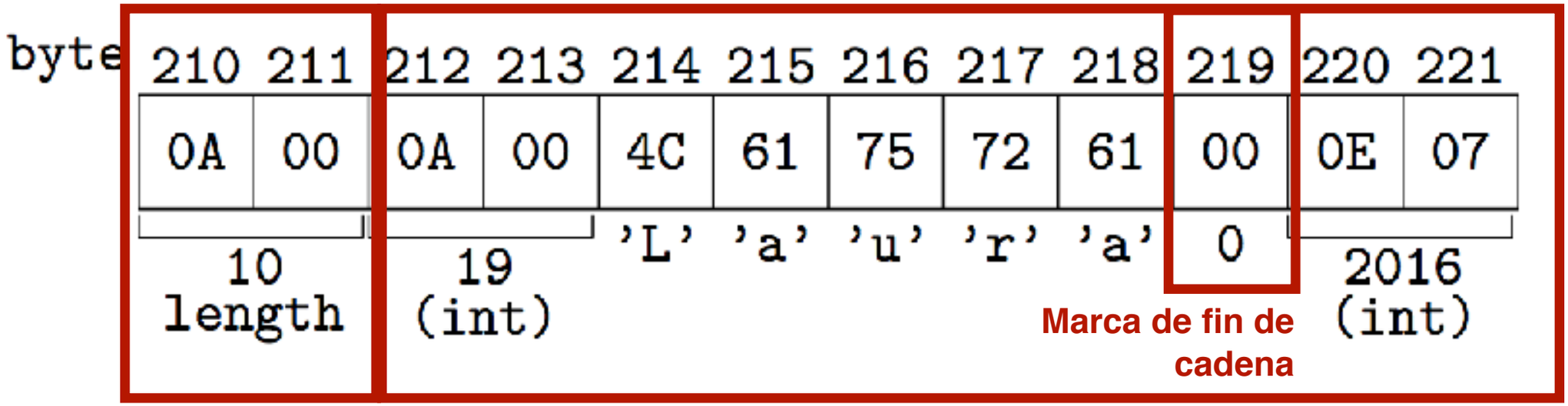
table_insert_record: Inserta un registro en la última posición de la tabla. Los valores están representados por void**. Recordad que **void*** es un puntero a un tipo genérico de datos. Por lo tanto, podemos convertir un puntero a void en un puntero a otro tipo de datos sin hacer un casting de manera explicita.

table_read_record: Lee un registro de la tabla empezando por la posición marcada.

Parte 1.a

Archivos binarios

Lista de registros = Datos



Tamaño del registro a leer

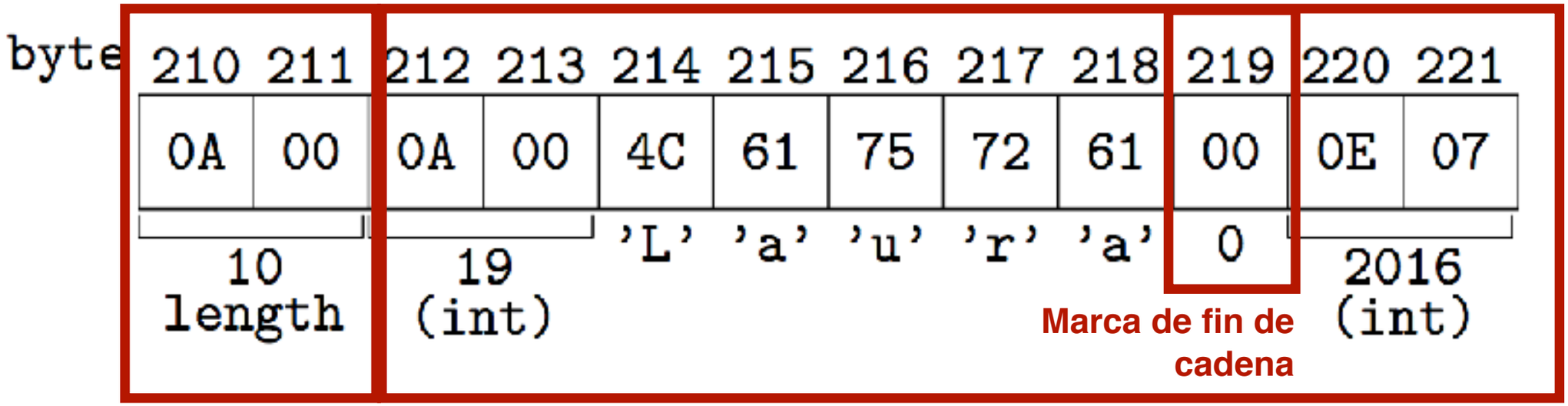
Registro de 10 bytes

Como los registros y las cadenas de caracteres son de **tamaño variable** al insertar un registro deberemos insertar **algunas marcas** que nos ayuden a controlar en su lectura el tamaño del registro a leer y el tamaño de la cadena de caracteres que compone un campo del conjunto de datos.

Parte 1.a

Archivos binarios

Lista de registros = Datos



Tamaño del registro a leer

Registro de 10 bytes

Lectura del tamaño del registro

```
int len;
fseek(fp, pt, SEEK_SET);
fread(&len, sizeof(int), 1, fp);
```

Lectura del registro y almacenamiento en un buffer

```
char *buf;
buf = (char *) malloc(len);
fread(buf, sizeof(char), len, fp);
```

Parte 1.a

Archivos binarios

Tareas

- Implementar las funciones del archivo **table.c**. Recordad que las cabeceras de las funciones no se pueden modificar ya que luego usaremos estas funciones como librería a la hora de manejar nuestros datos de manera local.

- Añadir los tipos de datos LLNG (long long integer) y DBL (double) a los archivos de definición de tipos de datos (**types.c** y **types.h**).

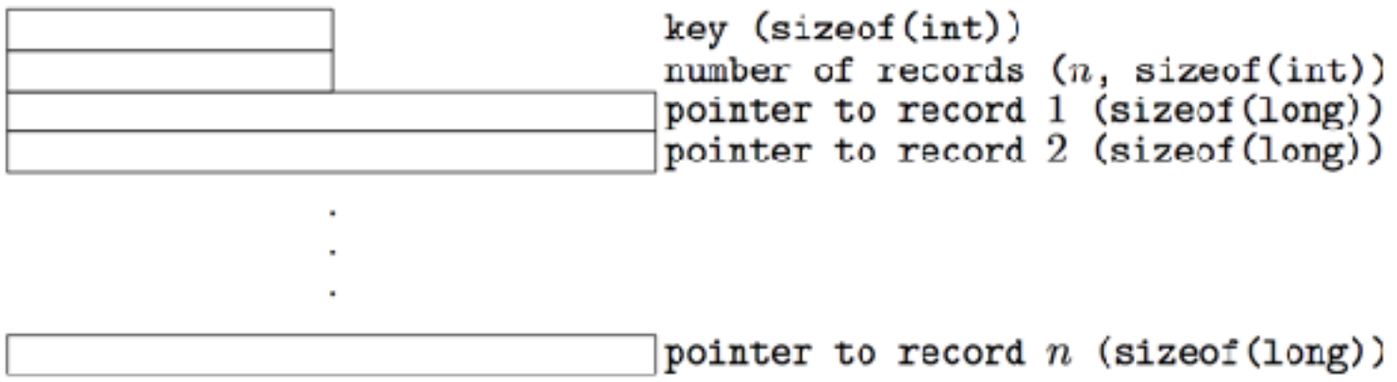
- Probar las dos tareas anteriores con un programa de prueba. Crear una tabla, insertar unos registros, cerrar la tabla, abrir la tabla, leerlos e imprimirlos.

Parte 1.b

Indices

1 indice = 1 fichero binario

Mantener ficheros de datos ordenados cuando el tamaño de estos aumenta es muy costoso computacionalmente hablando, ya que el tiempo de consulta aumenta cuando se lee un fichero de manera secuencial. Por lo tanto, una solución para disminuir el tiempo dedicado a consultar un fichero de datos es indexarlo por uno de los campos del registro de datos. De manera que crearemos un fichero binario para indexar todos los registros completos del fichero binario que compone la tabla.



Al leer el fichero binario de indice **crearemos la estructura de datos que convenga** para cargar toda la información almacenada en el indice y acelerar el tiempo de consulta de la información almacenada en el registro de datos de la tabla.

Parte 1.b

Indices

Tareas

Implementar las funciones del archivo **index.c**. Recordad que las cabeceras de las funciones no se pueden modificar ya que luego usaremos estas funciones como librería a la hora de manejar nuestros datos de manera local.

Parte 2

Datos en local y en remoto

Caso de estudio

Vamos a utilizar todo lo anterior para crear una tabla que nos permita dar puntuaciones a nuestros libros favoritos. Esa tabla es privada y se almacenará de manera local en un fichero binario. La estructura de la tabla a crear es la siguiente:

(ISBN : STR, titulo : STR, puntuación : INT, idLibro : INT)

Para la **inserción de datos** en la tabla crearemos un programa que se llame:

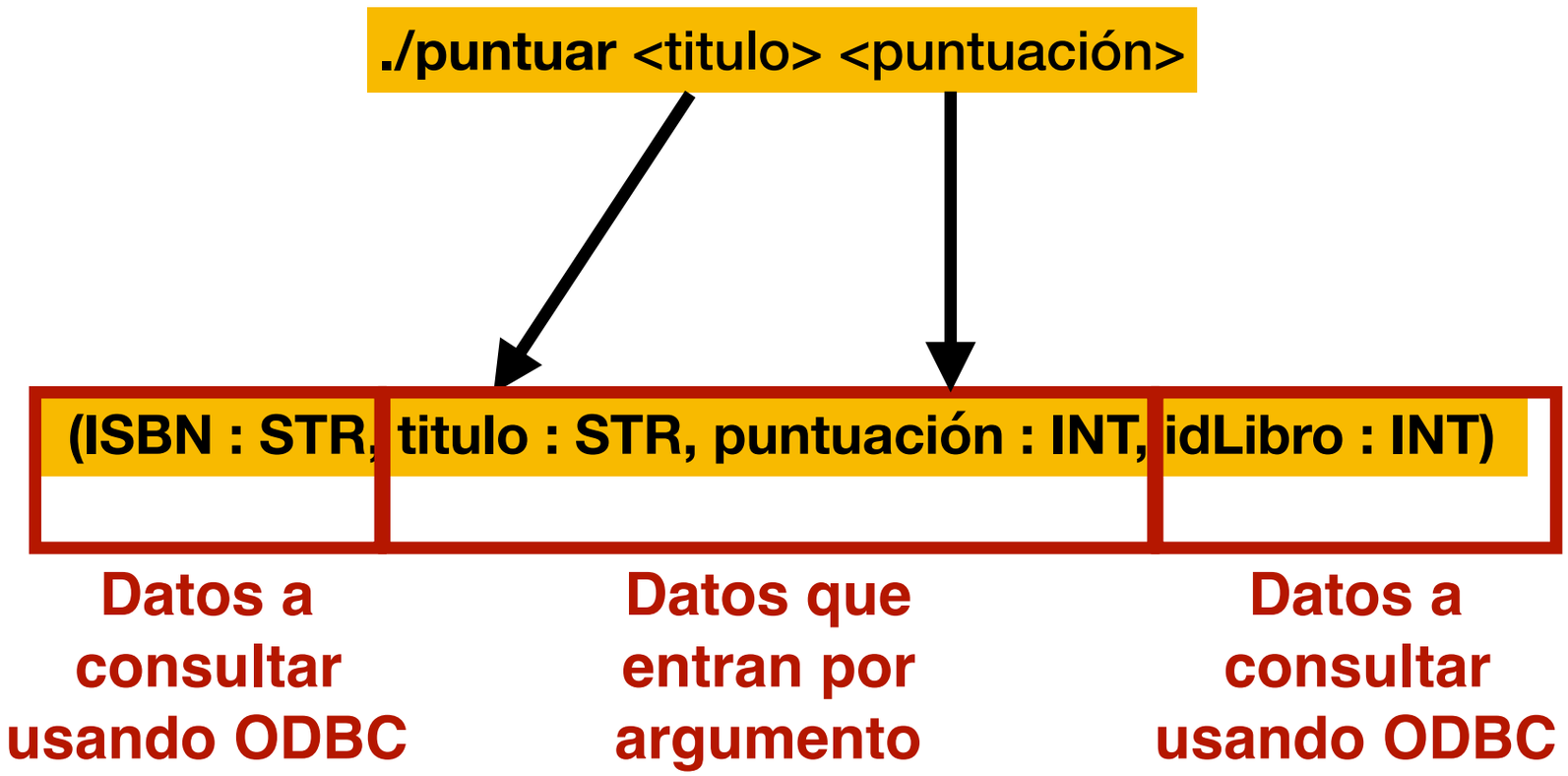
./puntuar <titulo> <puntuación>

Parte 2

Datos en local y en remoto

Caso de estudio

Como podéis comprobar para la inserción de un registro completo tendremos que combinar datos que **entran por argumento** y datos que estén almacenados en el servidor **postgres**.



Parte 2

Datos en local y en remoto

Caso de estudio

Para la **consulta de datos** en la tabla crearemos un programa que se llame:

./sugerir <puntuación>

Que imprima un listado de la forma:

<autor> <titulo>

Parte 2

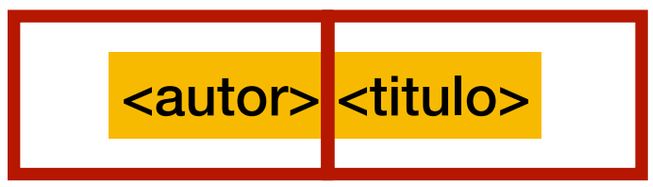
Datos en local y en remoto

Caso de estudio

Como podéis comprobar de nuevo para imprimir la lista de registros completos tendremos que combinar datos que **entran por argumento** y están **almacenados en la tabla en local** y datos que estén almacenados en el servidor **postgres**.

./sugerir <puntuación>

Datos a consultar usando ODBC



Datos almacenados en la tabla local

Parte 2

Datos en local y en remoto

Tareas

Implementar los dos programas anteriores sin utilizar indices.

Implementar los dos programas anteriores utilizando indices.