**UNIVERSIDAD AUTÓNOMA DE MADRID**



ESTRUCTURAS DE DATOS

(2018 - 2019)

**PRÁCTICA 3**

Alba Ramos

Miguel Díaz Martín

Grupo: 1211

Madrid, 14/12/2018

**TABLA DE CONTENIDOS**

[Introducción 3](#_30j0zll)

[Parte 1: tablas en ficheros binarios 3](#_1fob9te)

[Parte 2: programas con ODBC 4](#_3znysh7)

[Parte 3: índices en ficheros binarios 5](#_2et92p0)

[Conclusión 5](#_tyjcwt)

# Introducción

En esta práctica hemos tenido que trabajar con ficheros binarios para guardar los datos relativos a una base de datos. Además, hemos implementado dos programas usando ODBC para interactuar con la base de datos de PostgreSql y también con los datos que tenemos en los ficheros. Finalmente, hemos creado índices usando también ficheros binarios, y hemos editado los programas anteriores para que trabajen con ellos. El uso de índices permite optimizar el acceso a los datos de la base, por lo que se trata de una práctica muy importante.

En los siguientes apartados reflejaremos cómo hemos ido desarrollando cada punto del enunciado, los resultados obtenidos y los problemas que hemos ido encontrando en el camino.

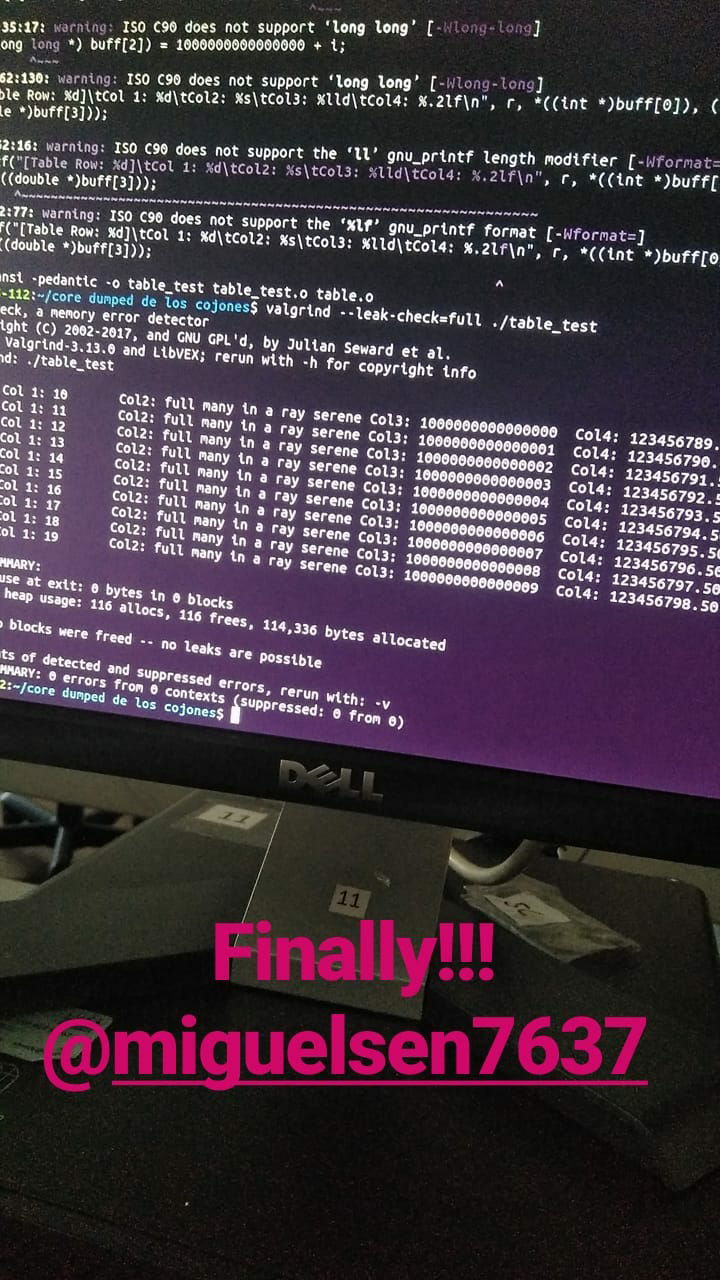
# Parte 1: tablas en ficheros binarios

Lo primero que hemos hecho ha sido añadir los tipos de datos LLNG y DBL, que se corresponden con el long long int y el double. Hemos modificado las funciones que teníamos disponibles para añadir estos tipos de datos, y a continuación, hemos creado un programa de prueba de estos tipos de datos. Este programa llama a estas funciones para ver que los tipos de datos se escriben correctamente en ficheros. Uno de los problemas que hemos encontrado ha sido que, si compilábamos con las banderas –ansi y –pedantic, nos daba warnings de que no se soporta el tipo de datos long long. Hay personas que han quitado estas banderas de compilación, pero nosotros las hemos dejado porque nos informan de otros errores que consideramos importantes. Para solucionarlo, hemos sustituido el long long por un long en este apartado, como se nos indicó en clase cuando comentamos este problema con los profesores.

Lo siguiente que hemos hecho ha sido implementar las funciones de table.c. Hemos creado la estructura table que consta de un nombre que indica el fichero que se corresponde con la tabla, un array de punteros genéricos que se utiliza para leer registros del fichero e indicadores con las posiciones inicial, final y actual del fichero. El fichero podría abrirse y cerrarse una sola vez, al abrir y cerrar la tabla, pero lo hemos estado abriendo y cerrando en todas las funciones que lo utilizan para asegurar que en ningún momento arrastramos posibles errores de otras funciones.

A la hora de insertar tuplas en la tabla, nosotros hemos decidido que, cuando se inserte una cadena, se va a escribir delante su longitud, ya que esto nos facilitará la posterior lectura de la misma.

La implementación de este apartado nos resultó bastante complicada, ya que nunca habíamos trabajado con ficheros binarios y hasta que conseguimos saber interpretar lo que se estaba escribiendo tardamos bastante. También tuvimos que aprender a usar fwrite y fread, que al principio no sabíamos usarlo y también perdimos mucho tiempo. Otro problema que encontramos y que nos costó descubrir fue que, a la hora de escribir y leer cadenas, estábamos reservando memoria de menos con malloc, sin tener en cuenta el ‘\0’ del final de cadena, y al arreglar esto se solucionaron muchísimas cosas. Finalmente, probamos a ejecutar el test que se nos proporcionó y obtuvimos la salida esperada. Aunque más adelante vimos que se habían subido más test al Moodle, nosotros utilizamos este en todo momento.



# Parte 2: programas con ODBC

En este apartado hemos tenido que crear dos programas usando ODBC: uno que vaya escribiendo opiniones y puntuaciones sobre usuarios en un fichero, y otro que sugiera usuarios dada una puntuación.

Para el primero, score.c, se deben pasar por argumento un nombre de usuario, una puntuación y un comentario. En el fichero se escribe el id del usurio, el screenname, la puntuación y el comentario (LLNG, STR, INT y STR, 4 columnas). Para obtener el id hemos reutilizado una consulta que teníamos en odbc de la práctica anterior, y a continuación hemos hecho llamadas a las funciones de table.c. También hemos copiado parte del código del table\_test que nos daban para leer el fichero y comprobar que nos estaba escribiendo bien los datos. Además, hemos utilizado esta sentencia:

if( access("score.txt", F\_OK ) != -1 ) {

}

else {

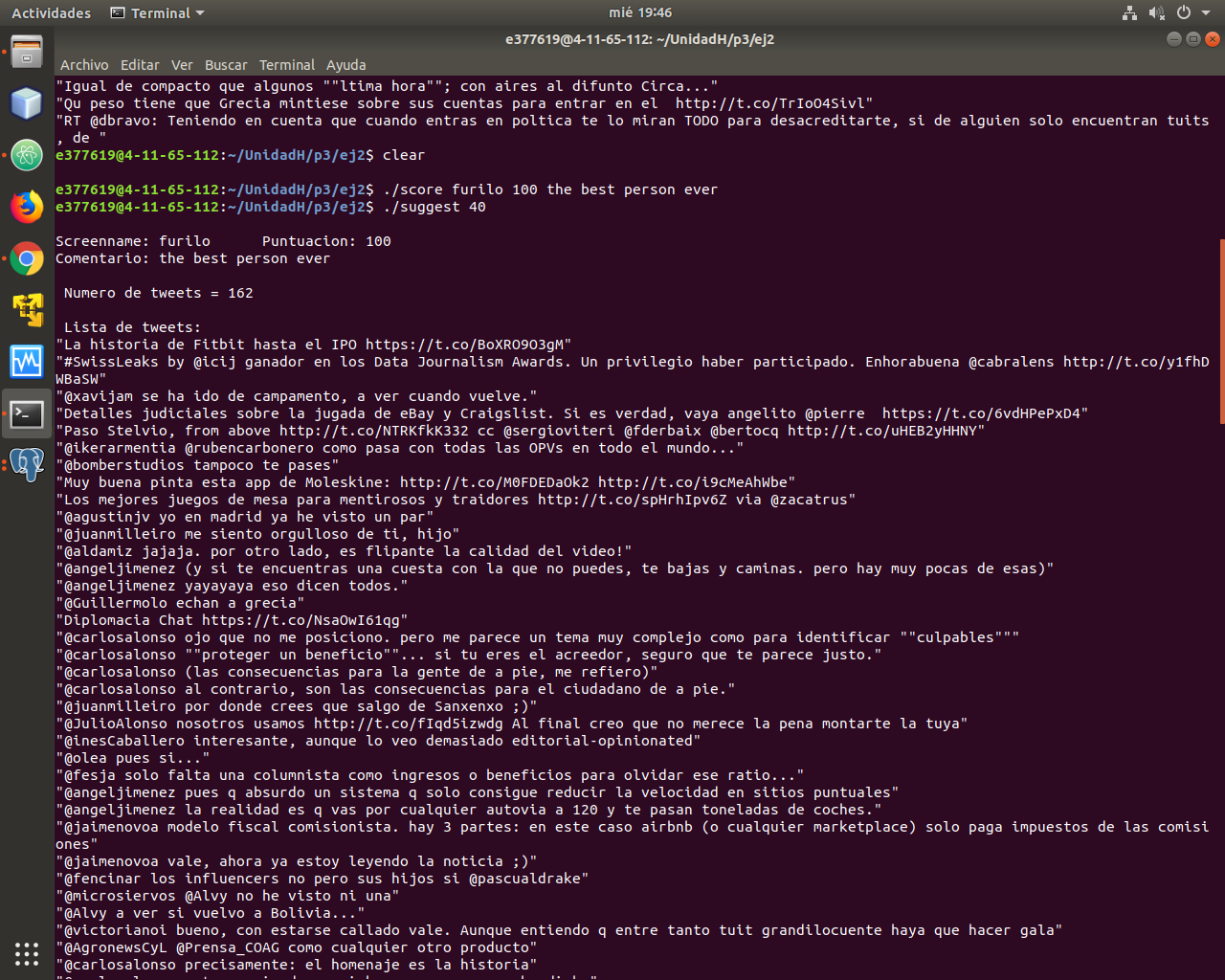
table\_create("score.txt", NCOLS, types);

}

Esto lo hemos hecho para comprobar si ya existe un fichero de puntuaciones. Si ya existe, entonces directamente se llama a table\_open, para seguir añadiendo datos en él, pero si no existiera, llamaríamos primero a table\_create para crear el fichero desde cero.

Después hemos realizado el segundo programa. En esta ocasión, se recibe una puntuación por línea de comandos y se debe ir recorriendo el fichero de puntuaciones, comparando la tercera columna con el valor recibido y, si es mayor, devolver el resto de datos que se nos especifican en el enunciado, utilizando para ello consultas de odbc que hemos reutilizado también de la práctica anterior.

Para probar los programas, hemos introducido primero una puntuación de 100 y un comentario para el usuario furilo, y después hemos llamado al programa de sugerir para una puntuación mayor de 40, obteniendo los resultados esperados. En el fichero salida.txt se adjunta la salida completa de la ejecución de la imagen.



# Parte 3: índices en ficheros binarios

Esta última parte nos ha resultado especialmente complicada. Debíamos implementar un índice y guardarlo en fichero para su posterior utilización. A la hora de manejarlo, se debía cargar en memoria para optimizar estas operaciones.

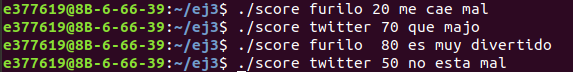
Hemos hecho la estructura con los siguientes campos: una cadena llamada path en la que se guarda el nombre del fichero en el que se guarda el índice, un entero llamado entries en el que se guarda el número de claves que tiene el índice, un entero llamado type en el que se guarda el tipo de dato que es la clave y una tabla de punteros a otra estructura llamada irecord. La estructura irecord tiene un entero llamado key en el que se guarda la clave, un entero llamado n\_regs donde se guarda el número de posiciones en las que ha sido escrita la clave y una tabla de enteros long en la que se guardan las posiciones en las que ha sido escrita la clave.

Hemos implementado otra función llamada index\_get\_edited de una forma parecida a index\_get, para adaptarla a nuestra implementación del programa. Hemos decidido que el puntero nposs almacena la posición del array en la que se encuentra la clave que buscamos, o -1 si no se encuentra. Para buscar la clave hemos implementado una función de búsqueda binaria. Esta implementación la hemos hecho así para facilitar la implementación de index\_put, de la que hablaremos a continuación.

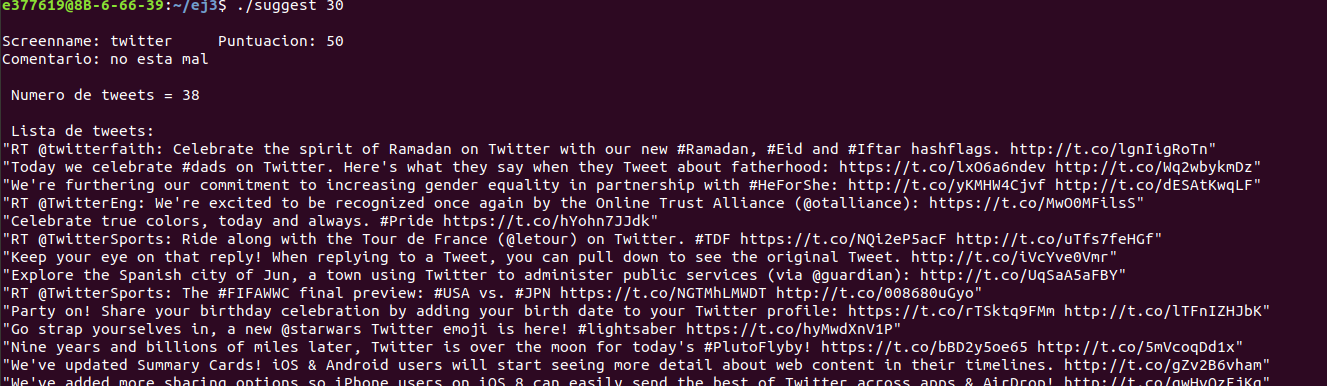
La función de inserción en el índice ha sido la más complicada. Para empezar, hacemos una búsqueda binaria en el índice con index\_get\_edited para ver si la clave que se quiere insertar ya existe. Si se inserta una nueva clave, hacemos un realloc de la estructura claves para que contenga una posición más.Ponemos la nueva clave en la última posición de la tabla y hacemos un malloc al campo de direcciones de la nueva posición con tamaño 1 para meter ahí el registro. Luego entramos en un bucle que va comparando la nueva clave con las anteriores y haciendo swap si es menor para que las claves estén ordenadas. También aumentamos el campo entries del índice, porque hay una clave más. Si ya existe la clave, solo tenemos que hacer un realloc del campo direcciones para que haya un espacio más, y metemos la posición en la última posición de la tabla. También aumentamos el campo n\_regs.

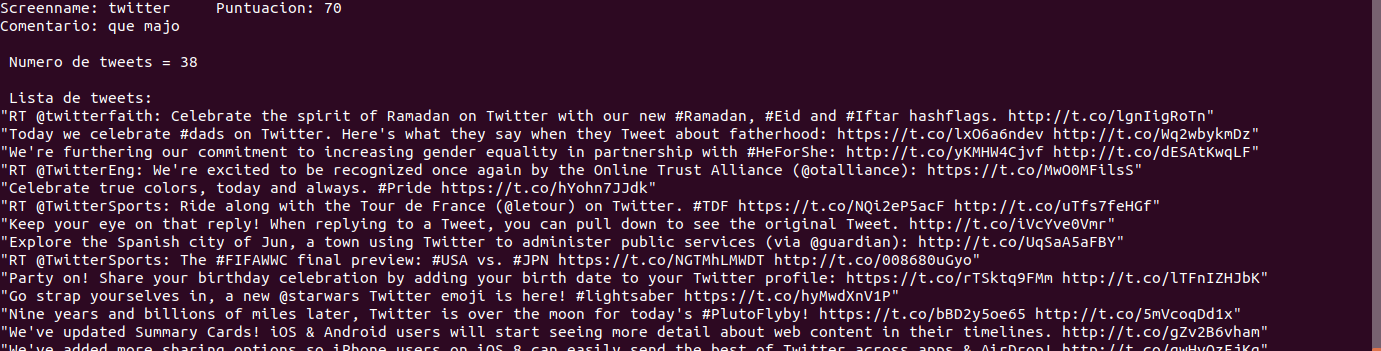
Finalmente hemos modificado los programas del apartado anterior, score y suggest, para que trabajen con índices.

Para comprobar que funciona, metemos cuatro puntuaciones a diferentes usuarios con el programa score:



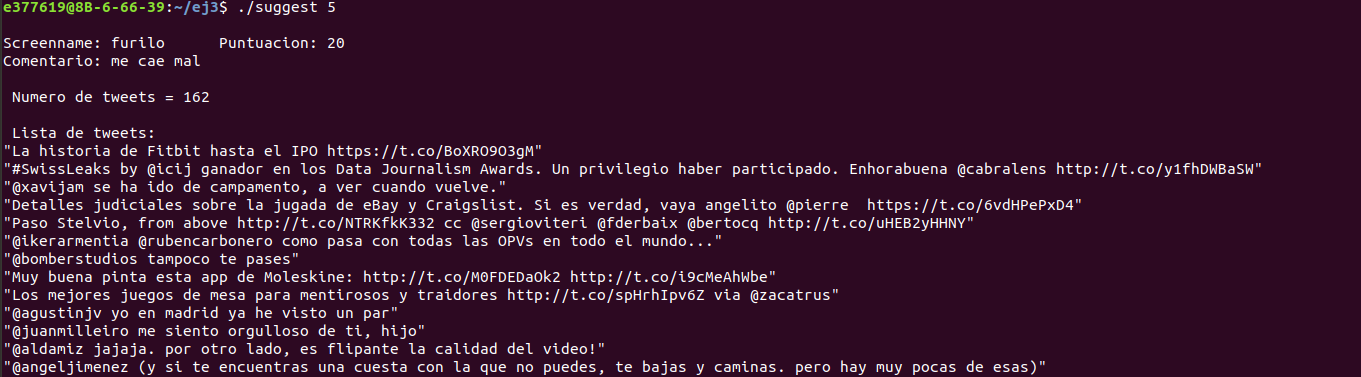
Y comprobamos cómo sale en el programa suggest pidiéndole que nos muestre las puntuaciones por encima de 30:







Como vemos, suggest sólo nos ha mostrado las puntuaciones que estaban por encima de 30. Si le pedimos que nos enseñe las puntuaciones por encima de 5 también nos enseña la otra que habíamos insertado:



También las imprime ordenadas, por lo que estaban bien almacenadas en el índice y los programas score y suggest funcionan correctamente.

# Conclusión

Como conclusión, creemos que ha sido una práctica útil para consolidar los conocimientos adquiridos en teoría y finalizar el curso con un alto nivel de conocimientos sobre bases de datos y su persistencia en ficheros binarios. Además, hemos aprendido bastante sobre índices, un tema que en teoría se ha visto bastante por encima, y cuando uno se enfrenta a ello en la práctica es bastante más complicado. Nos ha parecido una práctica interesante y con la que hemos mejorado bastante.