**Implementing and Testing the Role of Data Structures in optimizing large scale databases**

The effective management of large-scale databases is of paramount importance in today's digital landscape. One of the key considerations in this regard is the use of appropriate data structures to optimize the storage and retrieval of data. In this paper, we will explore the use of relational databases and the role of data structures in optimizing their performance.

To illustrate this point, let us consider a hypothetical scenario where an app is developed to provide music to users, and to connect users with similar musical interests. In this app, there is a table for users that contains their personal information and the songs they listen to. For the sake of simplicity, let us assume that the table contains only the name of the user and their songs.

The brute force method of searching for all users who listen to a particular song is to loop over every user and then search every user's record to see if they listen to said song. This method is of time complexity of O(SU), where 'S' is the number of songs present in the database and 'U' is the number of users present in the database. While this method may work well for small databases, it quickly becomes impractical as the size of the database increases.

To optimize the performance of this app, we can introduce a second table that contains information about all the songs. This second table includes the name of the song, and pointers to the records of users who listen to that song. This allows us to fetch all users who listen to a particular song in O(U) time, where 'U' is the number of users. This significantly reduces the time complexity of searching for users with similar musical interests.

Furthermore, the addition of the extra table allows us to easily find what other songs other users of similar interests are listening to, which opens up the possibility for a recommendation system based on similar interests of other users. This not only improves the user experience, but also increases the chances of users discovering new music that they may enjoy.

Provided below, is the code for programs that demonstrate the brute-force method and the optimal method.

(Digital copy of the programs are available at [github.com/DMN-sohan/DS\_in\_DBMS](https://github.com/DMN-sohan/DS_in_DBMS))

Program #1 : **Brute-Force Method**

#include <stdio.h>

#include <string.h>

#include <time.h>

#define MAX\_USERS 10

#define MAX\_SONGS 5

struct song {

char name[100];

};

struct user {

char name[100];

struct song\* songs[MAX\_SONGS];

};

// array of users

struct user users[MAX\_USERS];

void init\_users(){

for (int i = 0; i < MAX\_USERS; i++){

for (int j = 0; j < MAX\_SONGS; j++){

users[i].songs[j] = NULL;

}

}

}

void add\_user(char\* name, struct song\* songs[]){

static int user\_count = 0;

strcpy(users[user\_count].name, name);

for (int i = 0; i < MAX\_SONGS; i++) {

users[user\_count].songs[i] = songs[i];

}

user\_count++;

}

void search\_users(struct song\* song){

for (int i = 0; i < MAX\_USERS; i++){

for (int j = 0; j < MAX\_SONGS; j++) {

if (users[i].songs[j] && !strcmp(users[i].songs[j]->name,song->name)) {

printf("User %s listen to the same song %s\n", users[i].name,song->name);

break;

}

}

}

}

void main() {

//initialize the users array

init\_users();

//create songs

struct song song1 = {"song1"};

struct song song2 = {"song2"};

struct song song3 = {"song3"};

struct song song4 = {"song4"};

struct song song5 = {"song5"};

//create user1 and add songs

struct song\* user1\_songs[MAX\_SONGS] = {&song1, &song2};

add\_user("user1", user1\_songs);

//create user2 and add songs

struct song\* user2\_songs[MAX\_SONGS] = {&song1, &song3};

add\_user("user2", user2\_songs);

//create user3 and add songs

struct song\* user3\_songs[MAX\_SONGS] = {&song2, &song4,&song5};

add\_user("user3", user3\_songs);

//search for users who listen to the same song

clock\_t start = clock();

search\_users(&song1);

clock\_t end = clock();

double time\_elapsed = (double)(end-start)/CLOCKS\_PER\_SEC;

printf("Time Elapsed in seconds : %f\n",time\_elapsed);

}

Program #2 : **Optimal Method**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct song {

char name[100];

struct user \*listeners;

};

struct user {

char name[100];

struct song \*songs;

};

struct song\* searchSong(char\* name, struct song\*songList, int numSongs) {

for (int i = 0; i < numSongs; i++) {

if (strcmp(songList[i].name, name) == 0) {

return &songList[i];

}

}

return NULL;

}

void addUser(char\* name, struct song\* songList, int numSongs) {

struct user\* newUser = (struct user\*) malloc(sizeof(struct user));

strcpy(newUser->name, name);

newUser->songs = (struct song\*) malloc(sizeof(struct song)\*numSongs);

for(int i=0; i<numSongs; i++) {

newUser->songs[i] =songList[i];

}

songList[0].listeners[0] = \*newUser;

}

void findUsers(char\* songName, struct song\* songList, int numSongs) {

struct song\* result = searchSong(songName,songList,numSongs);

if (result == NULL) {

printf("Song not found.\n");

} else {

printf("Users listening to %s: \n", result->name);

for (int i = 0; i < numSongs; i++) {

if(result->listeners[i].name != NULL){

printf("%s\n", result->listeners[i].name);

}

}

}

}

void main() {

int numSongs = 5;

struct song songList[numSongs];

char\*songName[5]={"song1","song2","song3","song4","song5"};

for(int i = 0; i < numSongs; i++) {

strcpy(songList[i].name,songName[i]);

}

int numUsers = 5;

struct user userList[numUsers];

char\* userName[5]={"user1","user2","user3","user4","user5"};

for(int i=0; i<numUsers; i++){

strcpy(userList[i].name, userName[i]);

userList[i].songs = (struct song\*) malloc(sizeof(struct song)\*numSongs);

}

for(int i=0; i<numUsers; i++){

for(int j=0; j<numSongs; j++){

userList[i].songs[j]=songList[j];

}

}

clock\_t begin = clock();

findUsers("song1",songList,numUsers);

clock\_t end = clock();

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Time spent: %f\n", time\_spent);

}

As show evidently by the execution of both the programs, the brute-force method on average requires 0.130 milliseconds to execute while the optimal method on average only required 0.027 milliseconds, which makes the search query improve by approximately by a factor of **S**, where ‘**S**’is the number of songs present in the database. While offering an reduction in the search query, the optimal method also provides us with a way to traverse and fetch the records of all users listening to a song, in a very efficient manner.

(Number of songs used in the above programs are 5)

In conclusion, the use of appropriate data structures in relational databases can significantly improve the performance of large-scale databases and allows for the implementation of advanced features such as recommendation systems. As the data size and complexity continue to increase, the use of appropriate data structures will become increasingly important in optimizing the performance of databases. Therefore, it is crucial for developers to understand the importance of data structures and their role in optimizing the performance of relational databases.