

C++ & SQLite - המשך

שליפת רשומות

עד כה הפעלנו שאילתות המבצעות שינויים בבסיס הנתונים (הוספה, מחיקה וכו'..).
כעת נלמד איך לבצע שאילתות השולפות נתונים, ואיך לקרוא את הנתונים הללו מתוך קוד C++.
לשם כך נשתמש שוב בפונקציה `sqlite3_exec`, אולם הפעם נעשה שימוש גם בפרמטר השלישי (**חובה**) והרביעי של הפונקציה (רשות) שעד כה היו `null`.
הפרמטר השלישי יכיל מצביע לפונקציית `Callback`, הפרמטר הרביעי יכיל מידע שנרצה להעביר במידת הצורך לפונקציית ה-`callback`.

מצביע לפונקציה

מה מייצג שם של פונקציה ?

שם של פונקציה הנו למעשה מצביע (Pointer).
מצביע רגיל מכיל כתובת בזיכרון של משתנה או אובייקט,
שם של מערך הוא מצביע (קבוע) לכתובת של האיבר הראשון במערך,
שם של פונקציה הוא מצביע (קבוע) שמכיל את הכתובת של הפונקציה בזיכרון.
לדוגמה:
אם הפונקציה `Func` מתחילה בכתובת `0x0EF135` של ה-`Code Segment` (האזור בזכרון שאליו נטענת התוכנית), אזי ניתן לומר ש: `Func = 0x0EF135`.
כאשר קוראים בקוד לפונקציה, משתמשים בשמה. לדוגמה: `Func()`, המשמעות היא הוראה למחשב לקפוץ לכתובת ששמורה בשם הפונקציה.
כל קריאה לפונקציה, אם כך, הנה למעשה קפיצה בזיכרון המחשב.
דוגמה למצביע לפונקציה
נניח שיש לנו את הפונקציה הבאה:

```
void Sum(int num1, int num2)
{
    cout << num1 + num2 << endl;
}
```

ניתן להגדיר בקוד מצביע שיוכל להכיל את הכתובת של הפונקציה.
התחביר **להגדרת** מצביע שיוכל להצביע על הפונקציה הנ"ל יהיה כך:

```
void (*ptrFunc)(int, int);
```

ערך מוחזר

שם המצביע

רשימת
פרמטרים

ניתן לקרוא לפונקציה בשיטה שאתם כבר מכירים מצוין:

```
Sum(10, 20);
```

וכאלטרנטיבה, ניתן להפעיל אותה באמצעות המצביע לפונקציה:

```
void (*ptrFunc)(int, int);  
ptrFunc = Sum;  
ptrFunc (10, 20);
```

מצביע
לפונקציה

השמת הכתובת של
הפונקציה למצביע

הפעלת הפונקציה
באמצעות המצביע

נניח שבקוד יש לנו גם את הפונקציה הבאה:

```
void Sub(int num1, int num2)  
{  
    cout << num1 - num2 << endl;  
}
```

ניתן לקרוא לה בצורה הסטנדרטית:

```
Sub(10, 20);
```

או כאלטרנטיבה, ניתן להפעיל גם אותה באמצעות המצביע לפונקציה:

```
ptrFunc = Sub;  
ptrFunc (10, 20);
```

השמה של הכתובת של
הפונקציה למצביע

הפעלת הפונקציה
באמצעות המצביע

מגבלה

המצביע מהטיפוס `int*` יכול להצביע רק על משתנה מהטיפוס `int`, מצביע מהטיפוס `Person*` יכול להצביע רק אובייקט מהטיפוס `Person` ובאותו אופן מצביע לפונקציה יכול להצביע רק על פונקציה בעלת **חתימה** זהה לחתימה של המצביע לפונקציה, דהיינו, מצביע לפונקציה הוא תלוי חתימת הפונקציה.

לדוגמה:

אם בקוד קיימת הפונקציה הבאה:

```
void Sum(double num1, double num2)  
{  
    cout << num1 + num2 << endl;  
}
```

המצביע `ptrFunc` לא יוכל לקבל את הכתובת של הפונקציה הזו משום שהחתימה שלה שונה. חשבו – מה יהיה התחביר של המצביע לפונקציה אשר יוכל לקבל את הכתובת של המתודה השנייה?

מצביע לפונקציה כפרמטר

ניתן לשלוח מצביע לפונקציה כפרמטר לפונקציה אחרת, לדוגמה:

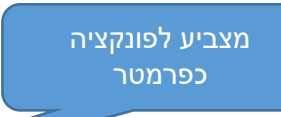
```
void Sum(int num1, int num2)
{
    cout << num1 + num2 << endl;
}

void Sub(int num1, int num2)
{
    cout << num1 - num2 << endl;
}

void MyFunc(void(*ptr)(int, int), int num1, int num2)
{
    cout<<"num1 = "<< num1 << " , num2 = " << num2 << endl;
    ptr(num1, num2);
}

int main()
{
    MyFunc(Sum, 10, 20);
    MyFunc(Sub, 10, 20);

    return 0;
}
```



באותו אופן, נשתמש במצביע לפונקציה על מנת לקרוא את הערכים החוזרים משאילתת SELECT.

Callback Functions (פונקציות משוב)

החתימה של פונקציית ה-callback:

```
int callback(void *data, int argc, char **argv, char **azColName)
```

void *data	המידע הכללי שעובר בפרמטר הרביעי של sqlite3_exec
int argc	מספר השדות שיש ברשומה
char **argv	מערך של מחרוזות שמכיל את המידע שנשלף
char **azColName	מערך של מחרוזות עם שמות השדות

תוצאת הרצת שאילתת SELECT על טבלה עשויה להחזיר מספר רשומות. כאשר נפעיל את הפונקציה sqlite3_exec, פונקציית המשוב תיקרא עבור כל רשומה בנפרד.

כלומר, אם תוצאת השאילתה היא 100 רשומות, פונקציית ה-callback תופעל 100 פעמים.

דוגמה ראשונה:

נריץ את sqlite3_exec עם שאילתת השליפה הבאה:

SELECT * FROM PERSONS;

בטבלה PERSONS ישנן 3 רשומות שישלפן, עבור כל רשומה תופעל פונקציית ה-callback בנפרד ובאמצעותה נדפיס את שם השדה ואת התוכן, כפי שניתן לראות בקטע קוד הבא:

```
int callback(void *data, int argc, char **argv, char **azColName)
{
    for (int i = 0; i < argc; i++)
    {
        cout << azColName[i] << " = " << argv[i] << " , ";
    }
    cout << endl;
    return 0;
}
```

מעבר על השדות
שיש ברשומה

הדפסת שם השדה

הדפסת תוכן
(המידע)

שאילתת השליפה

```
char* sqlStatement = "SELECT * FROM PERSONS;";
char *errorMessage = nullptr;
res = sqlite3_exec(db, sqlStatement, callback, nullptr, &errorMessage);
if (res != SQLITE_OK) {
    return false;
}
```

הרצת השאילתה

מצביע לפונקציית ה-callback

הפלט:

```
C:\WINDOWS\system32\cmd.exe
ID = 1 , LAST_NAME = Levi , FIRST_NAME = Shoshana , EMAIL = shoshana@mail.com ,
ID = 2 , LAST_NAME = Cohen , FIRST_NAME = Moshe , EMAIL = moshe@mail.com ,
ID = 3 , LAST_NAME = Israeli , FIRST_NAME = Israel , EMAIL = israel@mail.com ,
Press any key to continue . . .
```

דוגמה שניה:

בדוגמה הראשונה קיבלנו אוסף רשומות שנשלפו כתוצאה מהרצת שאילתת SELECT והדפסנו את התוצאה. ברוב המקרים שנשלף רשומות נרצה לבצע עיבוד נוסף. במקרים אלו נרצה לשמור את המידע שנשלף בזיכרון המחשב בצורה נוחה לעיבוד

איך נממש זאת?

דרך מקובלת היא באמצעות מחלקות המכונות Entities. מחלקת Entity היא מחלקה או מבנה שמהווה שכפול של המידע שנשלף מהטבלה ומשתמשים בה על מנת לשמור את המידע שנשלף. במהות זו המחלקה "טיפשה" וכל תפקידה לשמור מידע בזיכרון בצורה מונחית עצמים. אין בה אלגוריתמים למעט בדיקת תקינות קלט במידת הצורך, חישוב ערך השדות במידת צורך (לדוגמא: שדה המחזיק גיל המחושב על פי תאריך הלידה) או סינון (filtering) הנתונים שיוצאים ממנה (לדוגמא: הדפסת ארבעת הספרות האחרונות של כרטיס האשראי).

הפעם נשלוף מידע מהטבלה PERSONS, נבנה אוסף של אובייקטים מהמחלקה Person ונדפיס אותם למסך:

Person.h :

```
class Person;
typedef list<Person> persons;
typedef persons::iterator persons_iter;

class Person
{
public:
    Person(int id, string lastName, string firstName);
    int getId() const;
    void setId(int id);

    string getFirstName() const;
    void setFirstName(const string& name);

    string getLastName() const;
    void setLastName(const string& name);

    string getEmail() const;
    void setEmail(const string& email);

private:
    int m_id;
    string m_firstName;
    string m_lastName;
    string m_email;
};
```

Person.cpp:

```
Person::Person(int id, string lastName, string firstName)
:m_id(id), m_lastName(lastName),m_firstName(firstName)
{}

int Person::getId() const
{
    return m_id;
}

void Person::setId(int id)
{
    m_id = id;
}

string Person::getFirstName() const
{
    return m_firstName;
}

void Person::setFirstName(const string& firstName)
{
    m_firstName = firstName;
}

string Person::getLastName() const
{
    return m_lastName;
}

void Person::setLastName(const string& lastName)
{
    m_lastName = lastName;
}

string Person::getEmail() const
{
    return m_email;
}

void Person::setEmail(const string& email)
{
    m_email = email;
}
```

פונקציית ה callback :

```
persons personList;

int callback(void *data, int argc, char **argv, char **azColName)
{
    Person person;
    for (int i = 0; i < argc; i++) {
        if (string(azColName[i])=="ID") {
            person.setId(atoi(argv[i]));
        }
        else if (string(azColName[i])=="LAST_NAME") {
            person.setLastName(argv[i]);
        }
        else if (string(azColName[i])=="FIRST_NAME") {
            person.setFirstName(argv[i]);
        }
        else if (string(azColName[i])=="EMAIL") {
            person.setEmail(argv[i]);
        }
    }
    personList.push_back(person);
    return 0;
}
```

פונקציית הדפסה:

```
void print()
{
    persons_iter iter = personList.begin();
    while (iter != personList.end())
    {
        cout << iter->getLastName() << " , "
              << iter->getFirstName() << " , "
              << iter->getEmail() << endl;
        ++iter;
    }
}
```

הרצת השאילתה:

```
char* sqlStatement = "SELECT * FROM PERSONS;";
char *errorMessage = nullptr;
res = sqlite3_exec(db, sqlStatement, callback, nullptr, &errorMessage);
if (res != SQLITE_OK) {
    return false;
}
print();
```

פלט:

```
C:\WINDOWS\system32\cmd.exe
Levi , Shoshana , shoshana@mail.com
Cohen , Moshe , moshe@mail.com
Israeli , Israel , israel@mail.com
Press any key to continue . . .
```

שימו לב!

1. האובייקט `personList` הוא גלובאלי. זהו פתרון לא מומלץ ומוצג כאן רק לצורך פשטות הדוגמא. פתרון טוב יותר יהיה שימוש בפרמטר הרביעי של הפונקציה `sqlite3_exec`. לא נתעכב עליו בשלב זה ותוכלו ללמוד עוד על השימוש בו בעזרת התיעוד הרלוונטי. בכל מקרה, לא מומלץ להגדיר משתנים גלובאליים כשעובדים עם פונקציות `callback`.
2. שמות השדות (`FIRST_NAME`, `LAST_NAME` וכו') קבועים בקוד וזאת רק לצורך הדוגמא. מומלץ להגדירם כ- `define`.
3. כאשר רוצים להשוות ערכים של אובייקטי `string` ניתן להשתמש גם באופרטור השוויון.

תרגול מעשי 5 – שליפת רשומות

אופן ההגשה:

- ❖ המשיכו לעבוד על ה `private repository` ב- `GitLab` בשם `"DB_Labs"` שפתחתנו במסמך 11-SQL.DOCX
- ❖ המשיכו לעבוד על הפרויקט `Visual Studio` שפתחנו במסמך 13-C++-SQLite.docx1.
- ❖ צרו ענף בשם `feature/Lab3_CPP` היוצא מ `develop`.
- ❖ לאחר שסיימתם בצעו `commit` אינפורמטיבי לתוך `feature/Lab3_CPP`.

הדפיסו את רשימת כל האנשים בעלי שם פרטי מסוים (וודאו שהוספתם ל-DB לפחות שני אנשים עם שם פרטי זהה).

הגשת התרגיל לבדיקה

אופן ההגשה:

- ❖ וודאו שהפרויקט מוכן להגשה (הכל מתקמפל ועובד טוב) בענף `feature/Lab3_CPP`
- ❖ כנסו ל `repo` באתר `GitLab` בצעו `Merge Request` מ `feature/Lab3_CPP` אל תוך `develop`.