MYSQL C API 사용법

OneKick

Contents

OneKick

- How to use MySQL API
- 프로그래밍 흐름도
- 개발용 라이브러리 설치 후 경로 변경
- 간단한 예제 프로그램
- MySQL API
- 예제 프로그램 구현 실습

How to use MySQL API

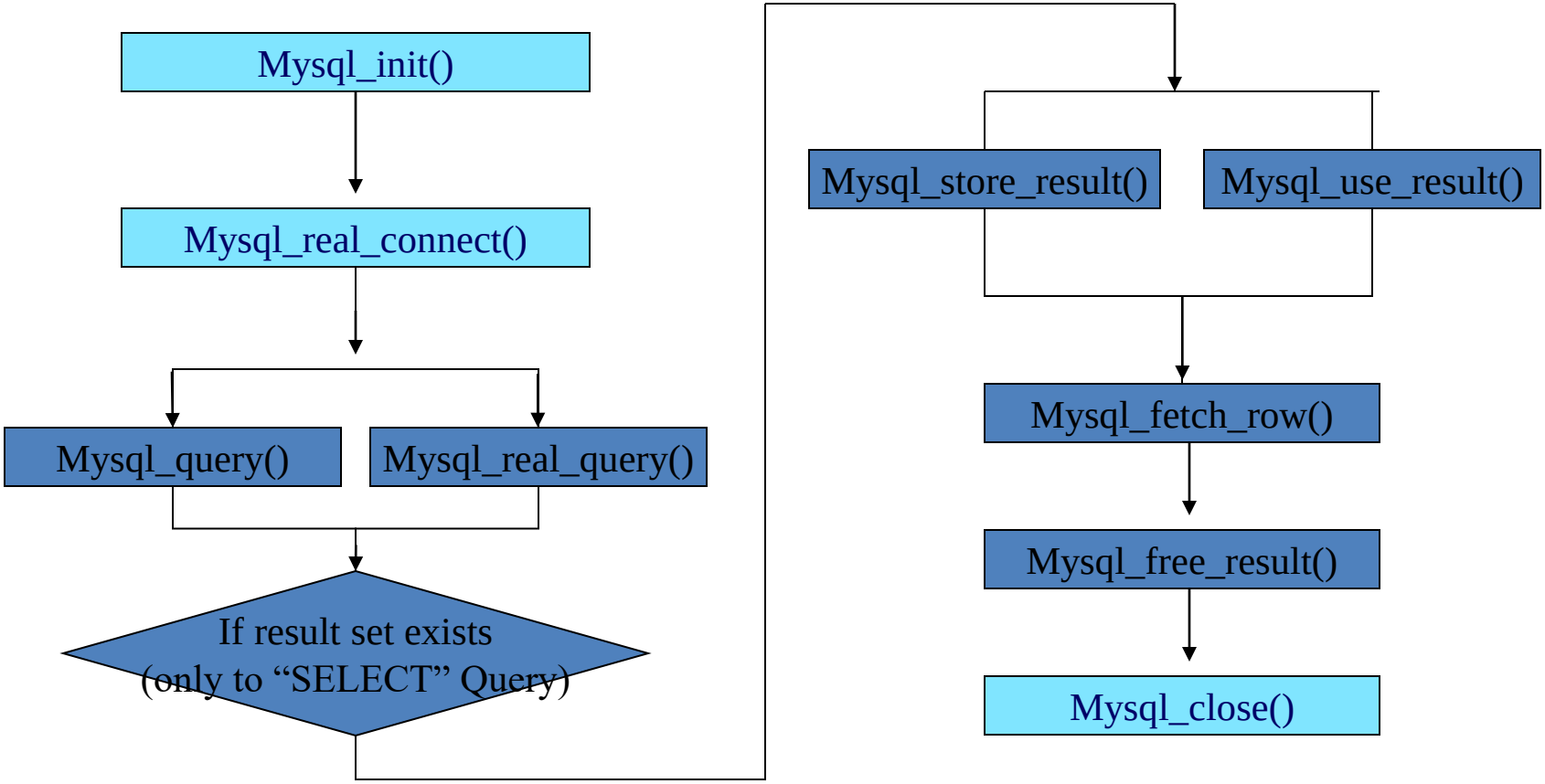
- Process

1.connect to SQL server

2.Quest Queries

3.manipulate result sets

프로그래밍 흐름도


Mysql_init()
Mysql_real_connect()
Mysql_query()
Mysql_real_query()
If result set exists
(only to “SELECT” Query)
Mysql_store_result()
Mysql_use_result()
Mysql_fetch_row()
Mysql_free_result()
Mysql_close()

개발용 라이브러리 설치 후 경로 변경

OneKick

1. 라이브러리 설치

   sudo apt-get install libmysqlclient-dev

   - 버전에 따라 개발자 라이브러리 이름 변경 가능
     - default-libmysqlclient-dev

2. 소스코드의 mysql 경로 변경

   #include <mysql/mysql.h>

3. 컴파일 명령

   gcc mysql.c -I /usr/include/mysql/mysql -Wall -g -lmysqlclient –g

root 비밀번호 설정

OneKick

- 라즈베리파이 root 비밀번호 설정 후 사용

```
pi@raspberrypi:~ $ sudo passwd root
새 UNIX 암호 입력:
새 UNIX 암호 재입력:
passwd: 암호를 성공적으로 업데이트했습니다
```

- 접속 시

```
pi@raspberrypi:~ $ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.1.37-MariaDB-0+deb9u1 Raspbian 9.0

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

(참고) MySQL 설치 경로 확인하기

OneKick

- 먼저 정확한 패키지 명 확인
  - 아래 명령으로 패키지 명을 확인합니다.

  >sudo dpkg -l | grep name-of-the-package

- 패키지 명으로 상세 설치 경로 확인

  >sudo dpkg -L package-name-that-you-got-from-previous-command

- MySQL 설치 경로 찾기 예시

  root@onekick:/bin# dpkg -l | grep mysql | grep dev
  ii libmysqlclient-dev 5.5.43-0+deb7u1 armhf MySQL database development files

  - 확인된 패키지 명: libmysqlclient-dev

  root@onekick:/bin# dpkg -L libmysqlclient-dev
  /.
  /usr
  /usr/bin
  /usr/bin/mysql_config
  … (생략)

```
#include <stdio.h>
#include "mysql.h“
Int main( )
{
        MYSQL mysql;
        MYSQL_RES *myresult;
        MYSQL_ROW row;
        unsigned int num_fields;
        unsigned int num_rows;
        char * string_query;
        mysql_init(&mysql);
        mysql_real_connect(&mysql,"localhost","please",“********","please”
                ,0,NULL,0);
        string_query = "select * from docmanager\n";
        mysql_query(&mysql,string_query);
        myresult = mysql_store_result(&mysql);
        while(row = mysql_fetch_row(myresult))
                printf("%s\t %s\n",row[0],row[1]);
        mysql_free_result(myresult);
        mysql_close(&mysql);
        return 0;
}
```

GCC 컴파일

OneKick

- gcc –o <실행파일명> <소스파일명.c>
  –L<라이브러리경로> -I<include경로> –lmysqlclient –lz
- 예시 – MySQL 설치경로가 /usr/local/mysql 인경우

```
gcc -o connect1 connect1.c
 -L/usr/local/mysql/lib/mysql
 -I/usr/local/mysql/include/mysql
 -lmysqlclient
 -lz
```

MySQL API

❑ C API Function Description

function	prototype
mysql_init()	MYSQL *mysql_init(MYSQL *mysql)
mysql_real_connect()	MYSQL *mysql_real_connect(MYSQL *mysql,const char *host,const char *user,const char *passwd,const char *db, uint port,const char *unix_socket,uint client_flag)
mysql_query()	Int mysql_query(MYSQL *mysql,const char * query)
mysql_real_query()	Int mysql_real_query(MYSQL *mysql,const char * query, unsigned int length)
mysql_store_result()	MYSQL_RES *mysql_store_result(MYSQL *mysql)
mysql_use_result()	MYSQL_RES *musql_use_result(MYSQL *mysql)
mysql_fetch_row()	MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
mysql_free_result()	Void mysql_free_result(MYSQL_RES *result)
mysql_close()	Void mysql_close(MYSQL *mysql)

MySQL API

```
MYSQL *mysql_init(MYSQL *mysql)
```

설명

mysql_real_connect()함수에 맞는 **MYSQL** 오브젝트를 할당하고 초기화한다. 만약 **mysql** 가 **NULL** 포인터라면, 함수는 새로운 오브젝트를 할당하고 초기화하고 리턴한다. 그렇지 않으면, 오브젝트는 초기화되고, 오브젝트의 어드레스는 리턴된다. 만약 **mysql_init()** 함수가 새로운 오브젝트를 할당한다면, **mysql_close()** 함수가 연결을 종료하기 위해 호출될 때 제거된다.

리턴 값

초기화된 **MYSQL*** 핸들(다루기). 새로운 오브젝트를 할당하기에 메모리가 부족하다면, **NULL**.

에러

메모리가 부족한 경우, **NULL** 이 리턴된다.

MySQL API

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char
*host, const char *user, const char *passwd, const char
*db, unsigned int port, const char *unix_socket,
unsigned long client_flag)
```

설명

mysql_real_connect() 함수는 **host** 에서 구동하는 MySQL 데이터베이스 엔진으로 연결을 시도한다. **mysql_real_connect()** 함수는 사용자가 유효한 **MYSQL** 연결 핸들 구조를 요구하는 다른 어떤 API 함수를 실행할 수 있기 전에 성공적으로 완료되어야 한다.

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s₩n",
          mysql_error(&mysql));
}
```

MySQL API

```
int mysql_query(MYSQL *mysql, const char *query)
```

설명

Null로 종결된 문자열 **query** 에 의해 지적된 SQL 쿼리를 실행한다. 일반적으로, 문자열은 싱글 SQL 명령문으로 구성된다. 그리고 사용자는 명령문에 종결 세미콜론(';') 또는 **\g** 를 추가할 수 없다. 다중 명령문 실행이 가능했다면, 문자열은 세미콜론들로 나뉘어진 몇몇 명령문을 포함할 수 있다.

mysql_query() 함수는 바이너리 데이터를 포함하는 쿼리들의 경우 사용될 수 없다; 대신에 사용자는 **mysql_real_query()** 함수를 사용할 수 있다. (바이너리 데이터는 Binary data may contain the '**\0**' 캐릭터를 포함할 수 있다. **mysql_query()** 함수는 쿼리 문자열의 끝으로 해석될 수 있다.)

리턴 값

쿼리가 성공하면, 0(Zero) 에러가 발생하면, Non-zero.

MySQL API

```
int mysql_real_query(MYSQL *mysql, const char *query,
unsigned long length)
```

설명

`length` 바이트 길이의 문자열인 `query` 이 지시한 SQL 쿼리를 실행한다. 일반적으로, 문자열은 싱글 SQL 문으로 구성되어 있고 명령문에 종결 세미콜론(';') 또는 `\g` 로 추가할 수 없다. 다중 명령문 실행이 가능하다면, 문자열은 세미콜론으로 나뉘어진 몇몇 명령문을 포함할 수 있다.

바이너리 데이터가 '\0' 캐릭터를 포함하고 있을 수 있기 때문에, 바이너리 데이터를 포함한 쿼리의 경우 `mysql_query()` 함수보다 `mysql_real_query()` 함수를 사용해야 한다. 덧붙여, 쿼리 문자열에서 `strlen()` 을 호출하지 않기 때문에, `mysql_real_query()` 함수가 `mysql_query()` 함수보다 더 빠르다.

리턴 값

쿼리가 성공했으면, 0(Zero). 에러가 발생하면, Non-zero.

MySQL API

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

설명

데이터 검색에 성공한 쿼리마다 **mysql_store_result()** 또는
mysql_use_result() 함수를 호출해야 한다.
모든 경우에 **mysql_store_result()** 함수를 호출한다면,
그것이 해가 되거나 어떤 성능의 하락을 일으키지는 않는다.
mysql_store_result() 함수는 클라이언트로의 쿼리 결과 전체를 읽고,
MYSQL_RES 체계를 할당하며, 이 체계에 결과를 정돈한다.
쿼리가 결과 셋을 리턴하지 않으면 null 포인터를 리턴한다.

mysql_store_result() 함수를 호출하여 null 포인터가 아닌 결과를 얻기만 하면,
결과 셋에 로우가 얼마나 있는지 찾기 위해 **mysql_num_rows()** 함수를 호출해도 된다.

리턴 값

결과들과 함께 **MYSQL_RES** 결과 체계. 에러가 발생하면, **NULL**.

MySQL API

MYSQL_RES *mysql_use_result(MYSQL *mysql)

설명

데이터(SELECT, SHOW, DESCRIBE, EXPLAIN) 검색을 성공한 모든 쿼리에서
mysql_store_result() 또는 **mysql_use_result()** 함수를 호출해야 한다..

mysql_use_result() 함수는 결과 셋 검색을 초기화하지만 실제로
mysql_store_result() 함수가 하듯이 클라이언트로 결과셋을 읽지는 않는다.
대신에, 각각의 로우는 **mysql_fetch_row()**함수로 호출함으로써 개별적으로
검색되어야 한다. 이것은 임시 테이블이나 로컬 버퍼에 저장하지 않고 서버로부터 직접 쿼리
결과를 읽어서 다소 빠르고, **mysql_store_result()**함수보다 메모리를 적게 사용한다.
반면, 클라이언트 쪽의 각각의 로우에서 많은 프로세싱을 한다면,
서버를 구속하여 다른 쓰레드가 데이터가 나오는 테이블을 업데이트하는 것을 방해한다.

mysql_use_result()함수를 사용할 때, **NULL** 값이 리턴될 때까지
mysql_fetch_row() 함수를 실행해야 한다. 그렇지 않으면, 호출되지 않은 로우들은
다음 쿼리 중에 결과 셋의 일부분으로 리턴된다. C API 는 에러가 발생한다.

리턴 값 **MYSQL_RES** 결과 구조. 에러가 발생하면, **NULL**.

MySQL API

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

설명

결과 값의 다음 로우를 검색한다. `mysql_store_result()`함수 후에 사용될 때, `mysql_fetch_row()` 함수는 더 이상 검색한 로우가 없으면 `NULL` 을 리턴한다.

`mysql_use_result()`함수 후에 사용될 때, `mysql_fetch_row()` 함수는 더 이상 검색할 로우가 없거나 에러가 발생하면, `NULL` 을 리턴한다.

로우에서 필드 값의 길이는 `mysql_fetch_lengths()`함수를 호출함으로써 얻을 수 있다. 빈 필드 또는 `NULL` 을 포함한 필드는 둘 다 길이가 0이다; 사용자는 필드 값의 포인터를 체크함으로써 이것을 구별할 수 있다. 포인터가 `NULL` 이라면, 필드는 `NULL` 이다; 그렇지 않으면 필드는 비어있다.

리턴 값

다음 로우의 `MYSQL_ROW` 구조. 검색한 로우가 더 이상 없거나 또는 에러가 발생한 경우`NULL`

MySQL API

```
void mysql_free_result(MYSQL_RES *result)
```

설명

`mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()` 등등에 의한 결과 셋에 할당된 메모리를 제거한다. 사용자가 결과 세트로 실행할 때, `mysql_free_result()`함수 호출에 의해 사용하는 메모리를 제거해야 한다.

그것을 제거한 후 결과 셋에 접근하려 하지 말 것.

리턴 값 없음.

에러 없음.

MySQL API

```
void mysql_close(MYSQL *mysql)
```

설명

이전에 오픈된 연결을 종료하라. `mysql_close()` 함수는 `mysql` 에 의해 지정된 연결 핸들이 `mysql_init()` 또는 `mysql_connect()`함수에 의해서 자동적으로 할당된 경우 이를 해제할 수 있다.

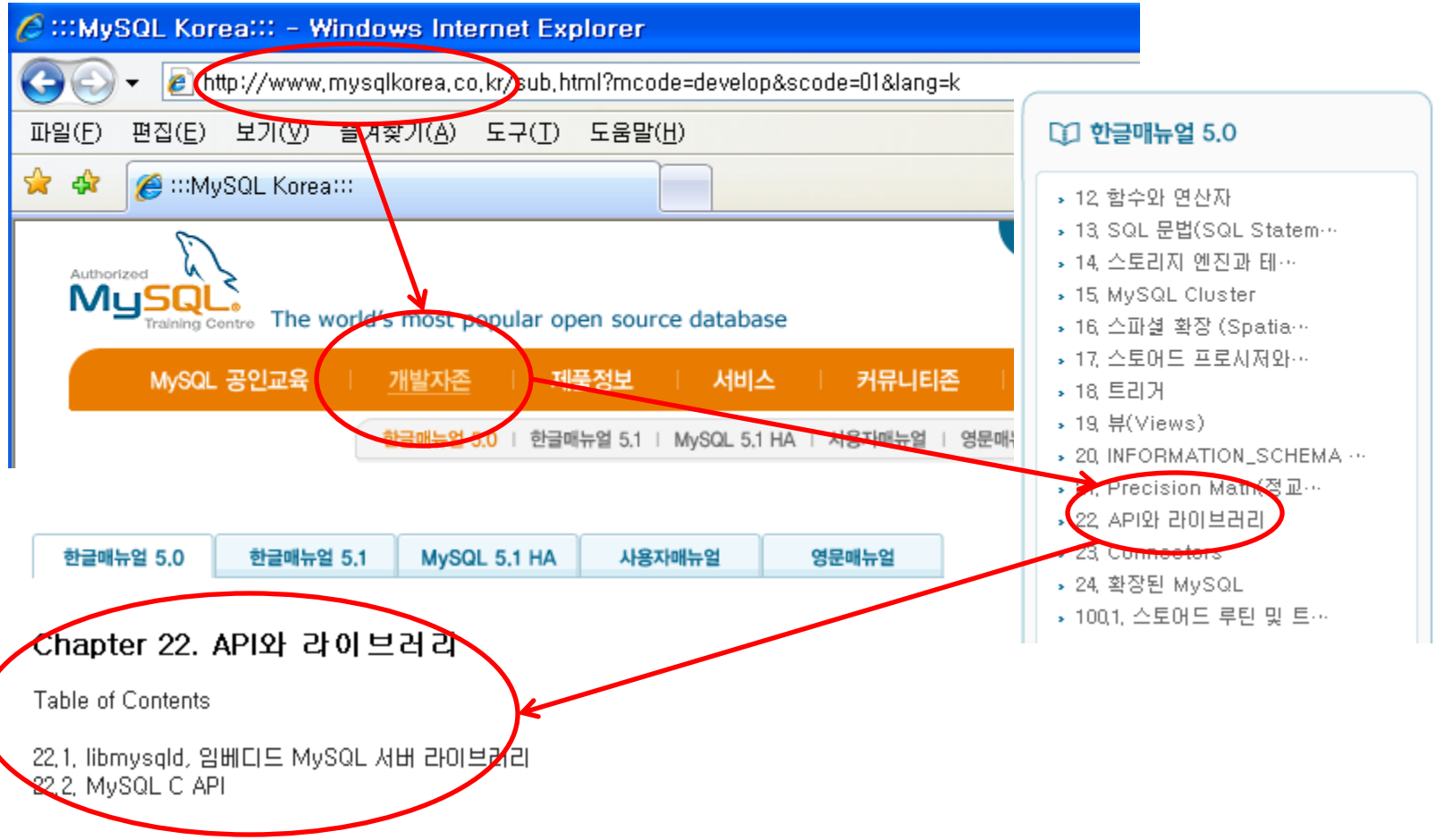
리턴 값

없다

에러

없다

MySQL API – 그 밖의 API들



예제 프로그램 구현 실습

•일부 시스템의 경우 LDFLAGS 줄에 다음의 옵션을 추가 해야 함.

-L/usr/lib/mysql

•Makefile (각 예제 프로그램을 한번에 컴파일 할 경우)

```
CONNECT=connect1 connect2
INSERT=insert1 insert2
SELECT=select1 select2 select3 select4
UPDATE=update1

ALL=$(CONNECT) $(INSERT) $(SELECT) $(UPDATE)

all: $(ALL)


CFLAGS=-I/usr/include/mysql -Wall -g
LDFLAGS=-lmysqlclient -g
# LDFLAGS=-lmysqlclient -g -L/usr/lib/mysql



clean:
                @rm -f $(ALL) *~
```

ControlDB.c

OneKick

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <stdlib.h>
#include <mysql/mysql.h>
#include <time.h>


// MySQL DB를 초기화 하고 사용할 수 있도록 연결하는 함수
int initDB(MYSQL * mysql, const char * host, const char * id, const char * pw, const char * db)
{
 printf("(i) initDB called, host=%s, id=%s, pw=%s, db=%s\n", host, id, pw, db);
 mysql_init(mysql); // DB 초기화
        if(mysql_real_connect(mysql, host, id, pw, db,0,NULL,0)) // DB 접속
        {
         printf("(i) mysql_real_connect success\n");
         return 0; // 성공
        }
        printf("(!) mysql_real_connect failed\n");
        return -1; // 실패
       }
```

```
// DB에 쓰는 함수 - 정수형 인자 5개는 DB Table의 각 field라고 가정
int writeDB(MYSQL * mysql, int door, int gas, int flame, int fan, int pin)
{
	char strQuery[255]=""; // 쿼리 작성에 사용할 버퍼
	 // 삽입 쿼리 작성, time 필드는 DATE 타입의 현재 시각
	 sprintf(strQuery, "INSERT INTO twoteam(id, door, gas, flame, fan, pin, time)
	                    values(null, %d, %d, %d, %d, %d, now())", door, gas, flame, fan, pin);
	 int res = mysql_query(mysql, strQuery); // 삽입 쿼리의 실행

	if (!res) // 성공
	 {
		printf("(i) inserted %lu rows.\n", (unsigned long)mysql_affected_rows(mysql));
	 }
	 else // 실패
	 {
		fprintf(stderr, "(!) insert error %d : %s\n", mysql_errno(mysql),
	mysql_error(mysql));
		return -1;
	 }
	 return 0;

}
```

```
// DB에서 읽어오기, id 가 primary key이고 읽은 결과는 buf에 구분자|를 이용해 반환
int readDB(MYSQL * mysql, char * buf, int size, int id)
{
	char strQuery[256]=""; // select query를 작성할 버퍼
	buf[0]=0; // 반환할 값은 strcat() 함수를 이용할 수 있도록 첫 바이트에 null을 넣는다.

	 // select query 작성
	 sprintf(strQuery, "SELECT door, gas, flame, fan FROM twoteam WHERE
	id=%d;",id);
	 int res = mysql_query(mysql, strQuery); // query의 실행
	 if(res!=0) // 실패
	 {
	  return  -1;
	 }
	 else // 성공
	 {
	  // select 문의 처리 결과는 커서 형태로 시스템에 의해 관리
	  // 만약 결과가 여러개일 경우 반복문을 돌면서 모든 row에 대해 처리해야 함
		MYSQL_RES * res_ptr = mysql_use_result(mysql); // 시스템에 결과셋을 맡김
		MYSQL_ROW sqlrow = mysql_fetch_row(res_ptr);  // 결과셋에서 한줄만 받음
```

```
// make a string for return
 unsigned int field_count=0;
 while(field_count<mysql_field_count(mysql)) // 가져온 한줄의 모든 필드 값을 꺼내옴
 {
        char buf_field[256]=""; // 각 필드에 보관된 개별 값을 저장할 임시 버퍼
        if(sqlrow[field_count])
              sprintf(buf_field,"|%s", sqlrow[field_count]); // 각 필드의 값을 문자열 형태로 가져
      옴
        else sprintf(buf_field,"|0"); // 값이 null 인 경우 0으로 표시
        strcat(buf,buf_field); // 가져온 필드의 값을 반환버퍼에 붙임.(string append)
        field_count++; // 다음 필드로 계속 진행
 }

 if(mysql_errno(mysql)) // 만약 에러가 있었으면
 {
              fprintf(stderr, "(!) error: %s\n", mysql_error(mysql)); // 원인 표시
              return -1; // 에러값 반환
 }
 mysql_free_result(res_ptr); // 시스템에 맡겨놓았던 결과셋을 버림
}
 return 0; // 정상종료 반환
}
```

```
// DB 접속 종료하기
int closeDB(MYSQL * mysql)
{
        mysql_close(mysql); // db 연결 해제
        return 1;
}
```

IoTestApp.c

OneKick

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <stdlib.h>
#include <mysql/mysql.h>
#include <time.h>

// 아래 4개 함수는 사실 ControlDB.c 파일에 구현
extern int initDB(MYSQL*, const char * host, const char * id, const char * pw, const char * db);
extern int writeDB(MYSQL*, int door, int gas, int flame, int fan, int pin);
extern int readDB(MYSQL*, char * buff, int size, int idRow);
extern int closeDB(MYSQL *);

int main()
{
        printf("(i) This is a test program for the ControlDB.\n");
        // DB에 삽입할 4가지 정수형 정보들
        int randomDoor = 0;
        int randomGas = 0;
        int randomFlame = 0;
        int randomFan = 0;
```

```
// 4가지 정보는 테스트를 위해 랜덤으로 만들어 둔다.
randomDoor = rand()%256;
randomGas = rand()%256;
randomFlame = rand()%256;
randomFan = rand()%256;

// 잘 만들어 졌는지 한번 확인
printf("(i) random: door=%d, gas=%d, flame=%d, fan=%d\n",
        randomDoor, randomGas, randomFlame, randomFan);

MYSQL mysql; // mysql DB의 연결 상태를 관리할 구조체 변수 생성

// DB에 연결. 로컬에 접속하고, id는 root, 비번은 1234, DB명 ssw
if(initDB(&mysql, "localhost","root","12341","ssw")<0)
{
        printf("(!) initDB failed\n"); // 실패
        return -1;
}
else    printf("(i) initDB successd!\n"); // 성공
```

```
int pin = 6; // 추가적으로 필요한 필드
 // DB에 쓰기 - 성공 시 0, 실패 시 -1 반환
 int res = writeDB(&mysql, randomDoor, randomGas, randomFlame, randomFan,pin);

 if(res<0) {
                printf("(!) writeDB failed\n");
                return -1;
 }
 else           printf("(i) writeDB success!, \n");


 // 이제 삽입한 row의 값을 읽어오기
 char buf[256]=""; // 결과를 받아올 문자열 버퍼
 int rowId = 1; // 테스트로 id가 1인 row의 값을 읽어온다.
 res = readDB(&mysql, buf, 256, rowId); // 읽기 실행

 if(res<0) // 실패
 {
                printf("(!) readDB failed\n");
                return -1;
 }
 else           printf("(i) readDB success!, buf=%s\n", buf); // 성공
```

```
 if(closeDB(&mysql)<0) // DB 연결 끊기
 {
            printf("(!) clseDB failed\n"); 실패
            return -1;
 }
 else       printf("(i) closeDB success\n"); 성공


 // 종료 알림
 printf("(i) This program will be closed.\n");
 return 0;
}
```

실행

OneKick

Makefile 만들기

OneKick

```
CC=gcc
SOURCES = IoTestApp.c  ControlDB.c
OBJECTS = ${SOURCES:.c=.o}
CFLAGS=-I /usr/include/mysql -Wall -g
LDFLAGS=-lmysqlclient -g
RM = rm -f
OUT = IoTestApp
$(OUT): $(OBJECTS)
	$(CC) -o $(OUT) $(OBJECTS) $(LIBS) $(CFLAGS) $(LDFLAGS)
clean:
	$(RM) $(OUT) *.o
```

////////tab키로 이동

테이블 생성

```
>>mysql –u root

>>show databases;
>>create database ssw;
>>create table twoteam(
      id int primary key auto_increment, door int, gas int,
     flame int, fan int, pin int, time date);
```

(참고 예제) MySQL C API로 구현한 도서관리시스템

OneKick

- 소스코드 주소
  - https://cafe.naver.com/android21/1640

1. Requirement 폴더 : 요구사항 정의 파일

2. Analysis 폴더: 요구사항 분석 파일

3. Design 폴더: 순서도, 구조도, ERD 파일

4. Implementation 폴더: 소스코드 최종본, Makefile

5. Test 폴더: 비어있음

6. Deployment 폴더: 최종 실행파일

(참고 예제) 기능설명

OneKick

1. 프로젝트명: 도서관리시스템(BookManagement)

2. 제공할 기능
   1) 회원관리
   (1) 등록
   (2) 조회
   (3) 변경
   (4) 탈퇴
   2) 책관리
   (1) 추가
   (2) 조회
   (3) 변경
   (4) 삭제
   3) 대여관리
   (1) 대여
   (2) 조회
   (3) 반납

3. 개발환경
   1) 운영체제: 우분투 14.04
   2) 사용자인터페이스: 터미널, 텍스트 기반
   3) DataBase: MySQL 5.X
   4) 개발언어: C, MySQLClient API
   5) 편집기: VI Editer