GaussDB 数据库接口
实验指导书

1． 目录

前　言

实验环境说明

本实验环境为华为云环境。

采用 GaussDB(MySQL)、openGaussDB 数据库管理系统作为实验平台，数据库访问接口为 ODBC、JDBC、Connector/Python，编程语言可采用 C、C++、Java、Python 等。

当以云数据库 GaussDB(MySQL)为实验平台时，通过公网/IP 登录方式访问数据库。

1． 实验介绍

1.1 实验目的

- ✧ 了解数据库应用程序设计采用的 ODBC、JDBC、Connector/Python 等多种接口，掌握接口及相关软件配置能力。
- ✧ 编写数据库应用程序，通过数据库访问接口访问数据库，培养数据库应用程序开发能力。

1.2 实验原理

动态 SQL 与数据库应用编程接口

数据库应用程序设计是数据库应用开发的一个重要方面。数据库系统用户通过两种方式访问数据库：1）直接通过 DBMS 利用 SQL 语句交互式访问数据库；2）通过数据库应用程序，借助嵌入式 SQL 和高级程序设计语言，访问数据库

DBMS 支持 SQL 语言直接访问数据库，但与高级程序设计语言（例如 C、C++、Java 等）相比，SQL 语言数据处理能力较弱，因此需要将 SQL 与高级程序设计语言结合起来，利用 SQL 访问数据，并将数据传递给高级语言程序进行处理，处理结果再利用 SQL 写回数据库。

这种嵌在高级语言程序中的 SQL 语句称为嵌入式 SQL（或者称为 ESQL），ESQL 随着应用程序执行被调用，完成数据库数据读写等数据管理功能，而高级语言程序则负责对数据库中数据进行统计分析等深层次处理。ESQL 分成两种：

（1）静态 ESQL，在程序执行前 SQL 的结构就已经确定，但可以在执行时传递一些数值参数。

数据库系统执行静态 ESQL 时，首先利用预编译分离 C、C++、Java 等宿主程序语言中的 SQL 语句，

代之以过程或函数调用，然后对剩余程序正常编译和连接库函数等。分离出来的 SQL 语句则在数据库端进行处理，进行语法检查、安全性检查和优化执行策略，绑定在数据库上形成包（packet）供应用程序调用。

（2）动态 ESQL，数据库应用程序执行时才确定所执行的 SQL 语句的结构和参数，通过数据库应用编程接口 ODBC、JDBC、Connector/Python、ADO 等访问数据库。

数据库系统执行动态 SQL 时，无法事先确实知道是什么样的 SQL 语句，从而无法进行静态绑定。这种绑定过程只能在程序执行过程中生成了确定的要执行的 SQL 语句时才能进行，称为动态绑定。

本次实验面向动态 SQL，应用程序采用 ODBC、JDBC、Connector/Python 三种接口访问数据库。

1.3 实验内容

1．了解通用数据库应用编程接口（例如 Connector/python（可选）、JDBC、ODBC 等）的配置方法。

2．掌握获取、修改数据库连接时长的方法。

3．利用 C、C++、Java、python 等高级程序设计语言编程实现简单的数据库应用程序，掌握基于 Connector/python（可选）、ODBC、JDBC 接口的数据库访问的基本原理和方法，访问 LTE 网络数据库，执行查找、增加、删除、更新等操作，掌握基于应用编程接口的数据库访问方法。

2． 实验要求

2.1 实验内容要求

1. 基于 JDBC 接口或基于 ODBC 接口的数据库访问实验，二选一完成一个即可，Connector/python（这个为可选项，有兴趣可以尝试以下，课程设计中不做要求）；
2. 实验时选取 TD-LTE 数据库作为数据源，可参照后文中以 tbcell 表为数据源的访问样例，自行设计访问 TLE 网络数据库访问操作;
3. 实验中需要完成如下数据库访问操作。

（1） 连接时长的获取和修改

TD-LTE 数据库中 tbAdjCell、tbMRODat、tbATUData 等表中的数据很大，对这些表进行查询（尤其是多表查询）、更新等操作时会花费较多的时间。

为了避免连接中途断掉，可以修改数据库默认连接时长，这里涉及到两个参数，即 interactive_timeout 和 wait_timeout。

interactive_timeout 参数，定义了对于交互式连接，服务端等待数据的最大时间。如果超过这个时间，

服务端仍然没有收到数据，则会关闭连接。

wait_timeout 参数，定义对于非交互式连接，服务端等待数据的最长时间。如果超过这个时间，服务端仍然没有收到数据，则会关闭连接。在连接线程启动的时候，根据连接的类型，决定会话级的 wait_timeout 的值是初始化为全局的 wait_timeout，还是全局的 interactive_timeout。即如果是交互式连接，会话变量 wait_timeout 初始化为全局的 interactive_timeout，否则，初始化为全局的 wait_timeout。

i. 获取数据库默认连接时间；

```
show global variables where Variable_name in ('interactive_timeout', 'wait_timeout');
```

	Variable_name	Value
1	interactive_timeout	28800
2	wait_timeout	28800

这里数据库默认的连接时间，只能由 root 用户修改，如果你尝试修改

```
set global INTERACTIVE_TIMEOUT=10;
```

会返回以下结果。

```
8  cnx.cmd_query("set global INTERACTIVE_TIMEOUT=10")
Exception has occurred: ProgrammingError
1227 (42000): Access denied; you need (at least one of) the SUPER or SYSTEM_VARIABLES_ADMIN privilege(s) for this operation
  File "C:\Users\AYIN\Documents\python\mysqlconnector.py", line 8, in <module>
    cnx.cmd_query("set global INTERACTIVE_TIMEOUT=10")
```

但是每个用户连接的 session 都对应有这两个参数，可以通过修改 session 对应的 interactive_timeout 和 wait_timeout 来修改连接时长限制

```
show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');
```

（获取会话对应的参数值）

	Variable_name	Value
1	interactive_timeout	28800
2	wait_timeout	28800

set session interactive_timeout=10;（设置当前会话的 interactive_timeout)

然后再次 show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');（获取会话对应的参数值）

	Variable_name	Value
1	interactive_timeout	10
2	wait_timeout	28800

ii. 在默认连接时间下，完成“4.（建议的）对 TD-LTE 配置数据库的访问操作”中给出的执行时间较长的查询（三选一），观察是否会超时；

iii. 增大、降低默认连接时间，观察是否超时。

iv. 不要求返回完整的数据，只需观察查询是否成功即可。

（2） 查询

选取一张表或几张表执行查询操作，并打印出数据（几行数据即可）。

（3） 插入

选取一张表执行插入操作，插入成功后打印出新插入的数据。

（4） 更新

选取一张表执行更新操作，更新成功后打印出更新后的数据。

（5） 删除

选取一张表执行删除操作，并检查操作是否成功。

2.2. 对 TD-LTE 配置数据库的访问操作（建议）

可以从“实验三 GaussDB(for MySQL)数据查询与修改”中选取涉及多表访问、执行时间较长的 SQL 语句，用于本实验中连接时长修改、数据库查询/增/删/改实验。也可以选取以下 3 个针对 tbAdjCell、tbMRODat、tbATUData 等数据量较大的表的查询，其执行时间（可能）较长，可用于实验内容“连接时长获取和修改”，也可用于后续的增删改查实验内容。

✧ 查询 1：对 tbAdjCell 的连接访问

要求：根据 tbAdjCell 表给出的小区 1 阶邻区关系，计算小区间的二阶邻区表 tbSecAdjCell。

原理：如果<a,b>∈tbAdjCell, <a,c>∈tbAdjCell，即 b 为 a 的 1 阶邻区，c 为 a 的 1 阶邻区，则 b 与 c 间存在 2 阶邻区关系。

方法：

```
cnx.cmd_query("create table test Select T1.N_Sector_ID as  N_Sector_ID, T2.N_Sector_ID as M_Sector_ID From `tbadjcell`
 as T1, `tbadjcell` as T2 Where T1.S_Sector_ID = T2.S_Sector_ID")
```

说明：'2.tbadjcell'是本实验数据库中存放小区一阶邻区关系的表，实际操作请按自己表名为准,cnx 是本实验建立的连接名，cmd_query 是调用接口执行 sql 语句

✧ 查询 2：对 tbMROData 的 update 操作

要求：针对 tbMROdata 表，如果邻小区的频点为 38400（即 LteNcEarfcn=38400），并且邻小区的 PCI 范围在 0 到 300 之间（即 LteNcPci between 0 and 300），则将主小区的参考信号接收信号强度 LteScRSRP 的值增加 1.

✧ 查询 3：对 tbATUData 的 delete 操作

要求：从 tbATUData 表中，删除同时满足下述条件的元组：

time=16:23,

Latitude between 33.75314 and 33.75316,Longitude between 112.82840 and 112.82842,

PCI=166, EARFCN=38400

3． 实验步骤

按照下述步骤完成本实验。

步骤 1.实验准备：

以课堂所学关于 SQL 语言相关内容为基础，课后查阅、自学 Connector/python 、ODBC、JDBC 等接口有关内容，包括体系结构、工作原理、数据访问过程、主要 API 接口的语法和使用方法等。

步骤 2. 数据库访问接口环境配置

根据实验所选的应用编程接口 ODBC、JDBC、Connector/Python，分别从不同网站下载接口驱动程序，安装配置接口环境，为后续实验做准备。

具体样例参照 4.1、4.2、4.3 各节。

步骤 4. 连接时长获取及修改

参照 4.1、4.2、4.3 各节样例，编写 C、C++、Java、Python 应用程序，查询系统默认的 ODBC/JDBC/Connector/Python 接口的连接时长配置，并根据实验需要修改连接时长。

步骤 5. 数据库连接及访问

参照 4.1、4.2、4.3 各节给出的样例，针对实验二建立的 LTE 网络数据库，编写 C、C++、Java、Python 应用程序，通过 ODBC、JDBC、Connector/Python 接口，连接数据库，对数据库内容进行查询、插入、删除、更新等操作，观察记录实验结果。

考虑到本实验可选择三种不同的数据库接口，后面将分别给出采用 ODBC、JDBC、Connector/Python 接口时的程序示例。这些程序参照了 TLE 提供的以 tbcell 表据库为访问对象的 ODBC、JDBC 接口访问程序。同学们可以参考这些示例，针对本次实验对 LTE 网络数据库的具体访问要求，编写程序各项实验内容。

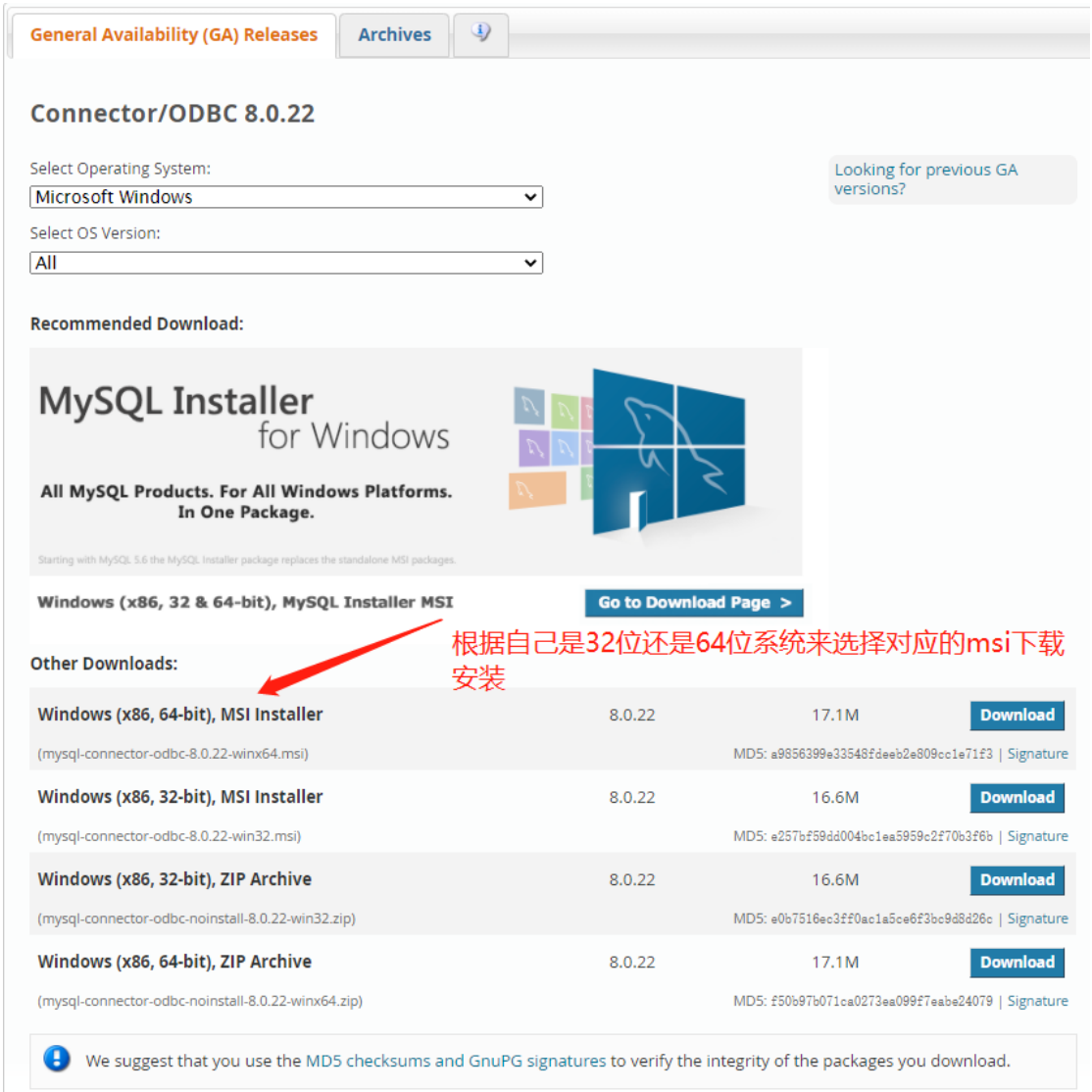
4． 示例

4.1 ODBC 接口访问

4.1.1 环境配置

4.1.1.1 下载及 ODBC 安装驱动

根据网址 https://dev.mysql.com/downloads/connector/odbc/，访问网站下载 Connector/ODBC 9.0.22。

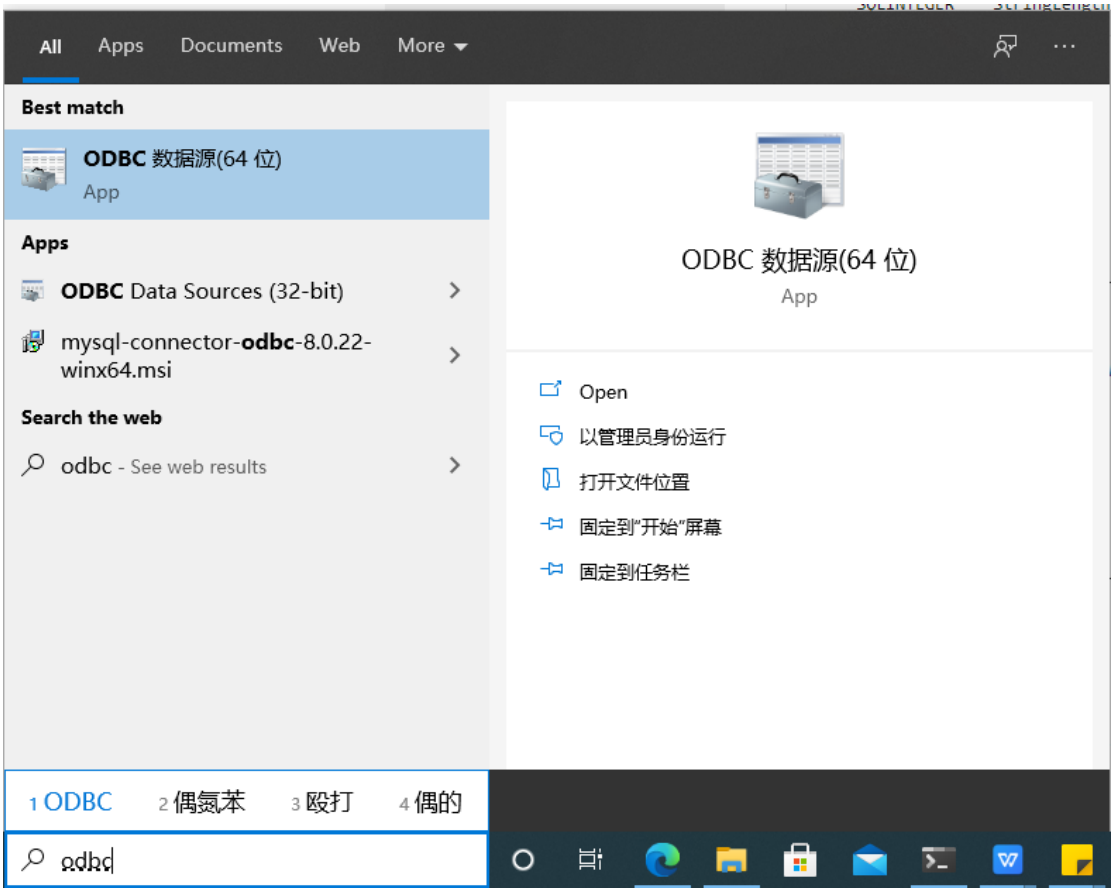


选择对应系统的 msi 文件，点击安装，安装选项默认。

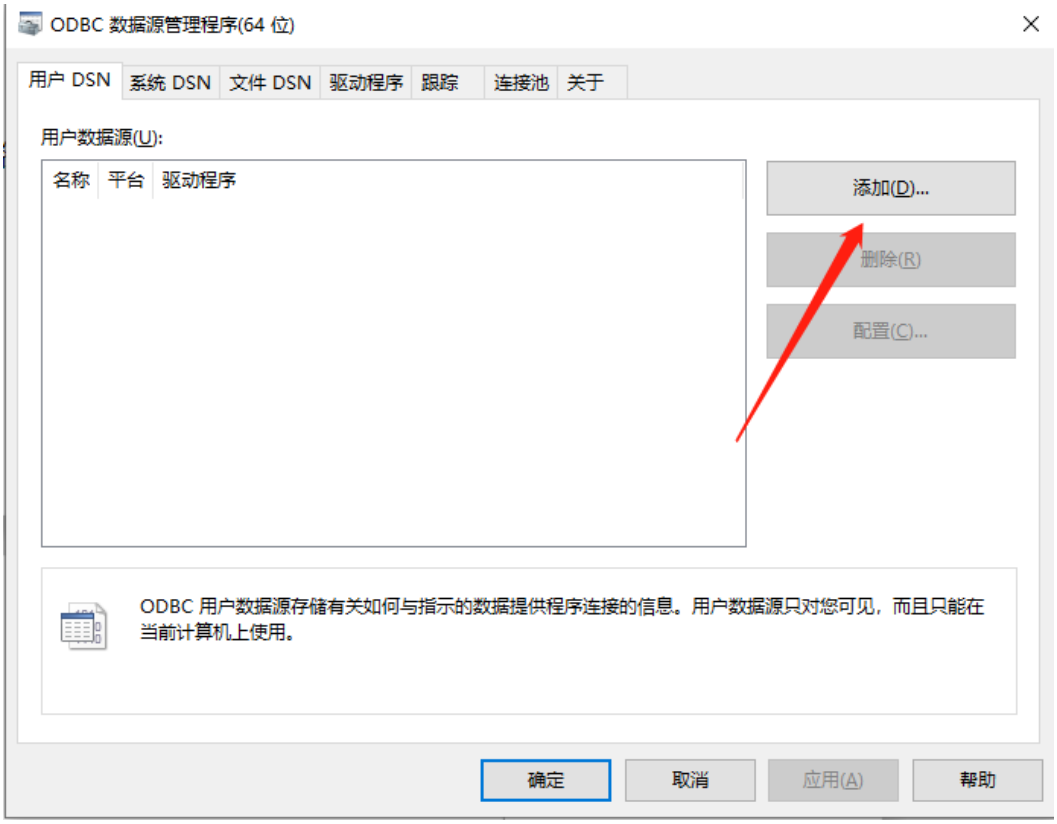
打开 ODBC 数据源后点击“添加”，添加一个名为“university”（名字不限，但注意要与后面程序中的名字对

应）的数据源；

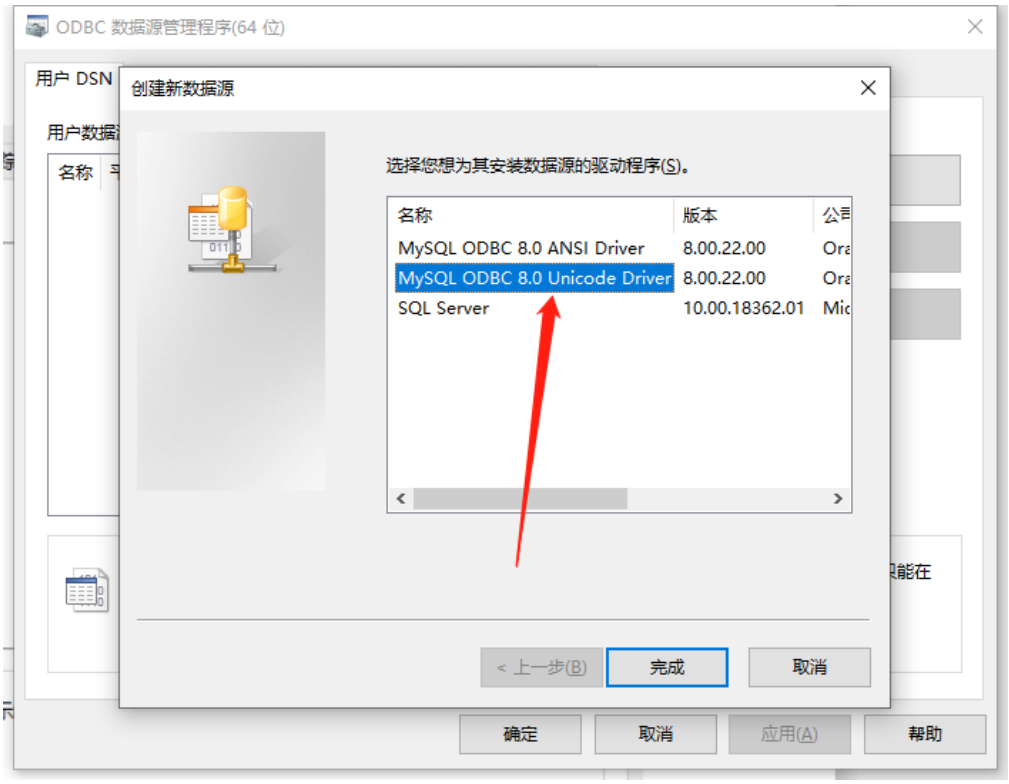
在 win10 中搜索 odbc：



点击打开：



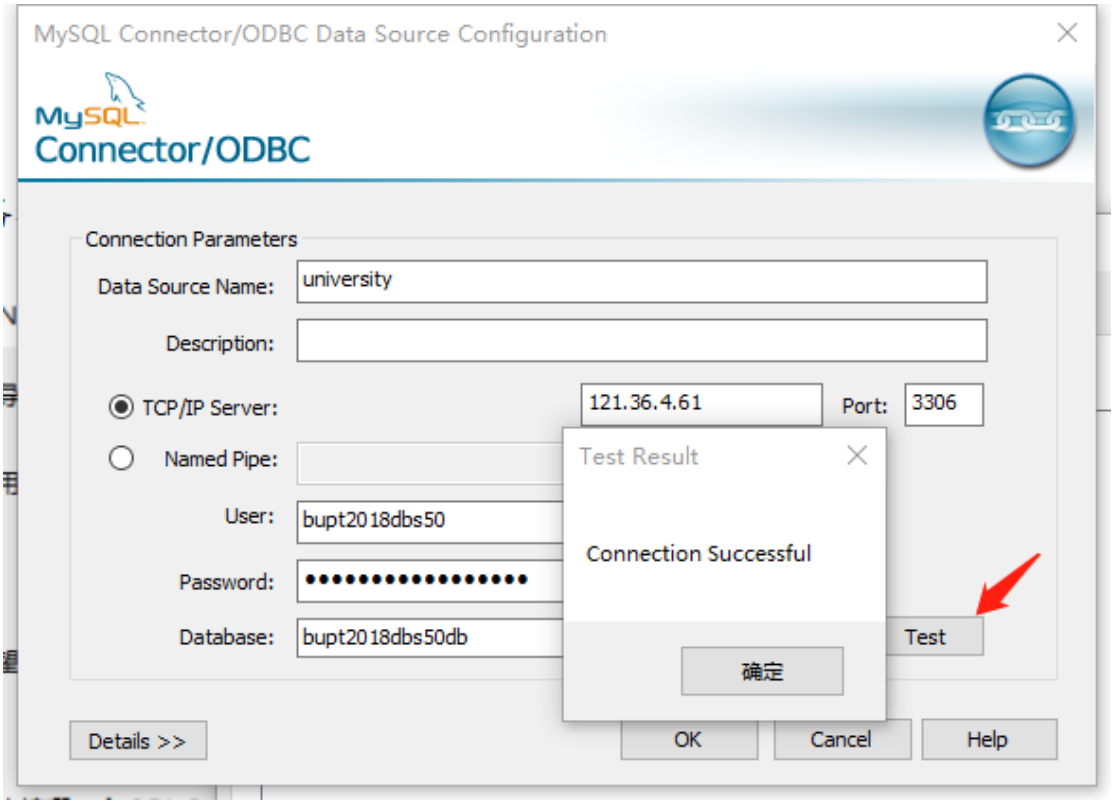
选择驱动程序：



注意事项：建议 ansi 和 unicode 都添加一个数据源，用其中一个数据源输出乱码时，尝试连接另一个数据

源。

4.1.1.2 设置数据源信息，IP,用户名,密码，数据库按照前面给每组同学分配的数据库设置，点击 test 测试连接；



出现如图所示的弹窗说明输入数据库信息无误，可继续进行实验。

4.1.2 连接时长的获取和修改

```
char sqlquery[] = "show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');";ame:" +
sector_name);
```

获取 interactive_timeout 和 wait_timeout 的值:

```
interactive_timeout 28800
wait_timeout 28800
```

```
char s1[] = "set session INTERACTIVE_TIMEOUT = 10000";
char sqlquery[] = "show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');";
```

修改 interactive_timeout 的值之后再次获取 interactive_timeout 和 wait_timeout 的值:

```
interactive_timeout 10000
wait_timeout 28800
```

4.1.3 编写 C++程序访问数据库

ODBC3.x 和 ODBC2.x 函数名字对应表

✧ **注意事项**

教科书所给例子的函数属于 ODBC 早期版本，从 Windows7 开始，内置的 ODBC 版本都为 3.X(目前 win10 电脑都仍是 3.x 版本)，根据实验所选的环境和编译器，例子所示程序有可能运行出结果，也有可能报错。

下图为不同版本的函数对比

An ODBC 3.*x* application working through the ODBC 3.*x* Driver Manager will work against an ODBC 2.*x* driver as long as no new features are used. Both duplicated functionality and behavioral changes do, however, affect the way that the ODBC 3.*x* application works on an ODBC 2.*x* driver. When working with an ODBC 2.*x* driver, the Driver Manager maps the following ODBC 3.*x* functions, which have replaced one or more ODBC 2.*x* functions, into the corresponding ODBC 2.*x* functions.

ODBC 3.*x* function	ODBC 2.*x* function
SQLAllocHandle	**SQLAllocEnv**, **SQLAllocConnect**, or **SQLAllocStmt**
SQLBulkOperations	**SQLSetPos**
SQLColAttribute	**SQLColAttributes**
SQLEndTran	**SQLTransact**
SQLFetch	**SQLExtendedFetch**
SQLFetchScroll	**SQLExtendedFetch**
SQLFreeHandle	**SQLFreeEnv**, **SQLFreeConnect**, or **SQLFreeStmt**
SQLGetConnectAttr	**SQLGetConnectOption**
SQLGetDiagRec	**SQLError**
SQLGetStmtAttr	**SQLGetStmtOption**[1]
SQLSetConnectAttr	**SQLSetConnectOption**
SQLSetStmtAttr	**SQLSetStmtOption**[1]

本次实验所用函数部分说明

➢ **分配函数 SQLAllocHandle**

```
SQLRETURN SQLAllocHandle(
      SQLSMALLINT   HandleType,
      SQLHANDLE     InputHandle,
      SQLHANDLE *   OutputHandlePtr);
```

参数说明:

HandleType：句柄类型，取值为 SQL_HANDLE_ENV（环境）、SQL_HANDLE_DBC（连接）、SQL_HANDLE_STMT（语句）等;

InputHandle：要分配新句柄的上下文中的输入句柄。如果 HandleType 是 SQL_HANDLE_ENV，其值为 SQL_NULL_HANDLE。另外，SQL_HANDLE_DBC 对应环境句柄，SQL_HANDLE_STMT 对应连接句柄；

OutputHandlePtr：指向存储当前分配句柄的变量的指针。

- **释放函数 SQLFreehandle**

```
SQLRETURN SQLFreeHandle(
     SQLSMALLINT   HandleType,
     SQLHANDLE     Handle);
```

参数说明:

HandleType：同上；

Handle：需要释放的句柄。

- **设置环境属性函数 SQLSetEnvAttr**

```
SQLRETURN SQLSetEnvAttr(
     SQLHENV      EnvironmentHandle,
     SQLINTEGER   Attribute,
     SQLPOINTER   ValuePtr,
     SQLINTEGER   StringLength);
```

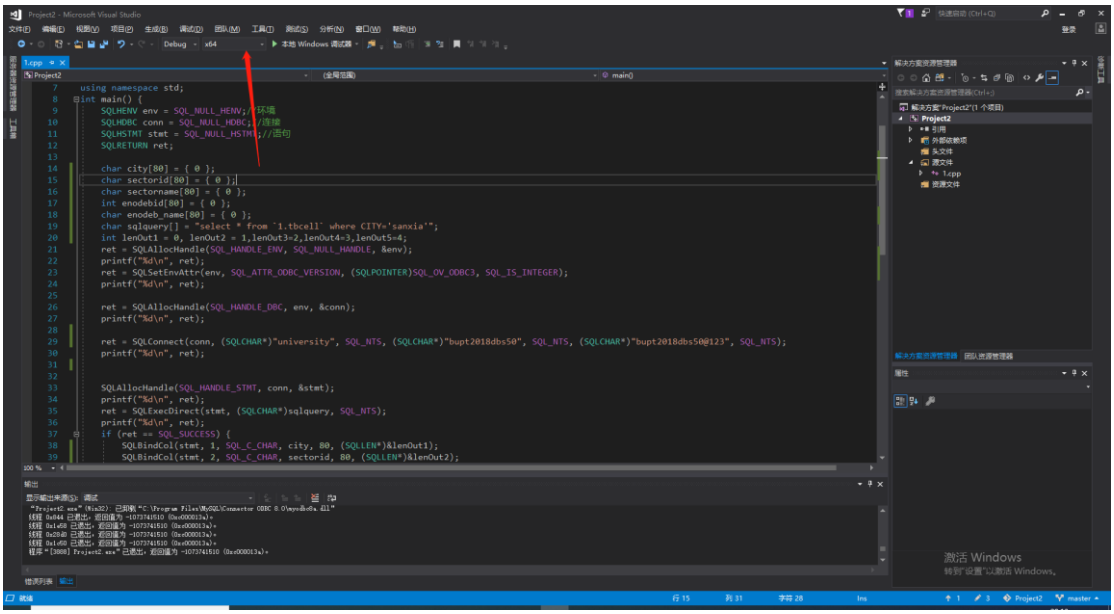
参数分别表示（环境句柄，需设置的环境属性名称，前一个属性的取值，ValuePtr 指向的数据的长度<如果是字符串则为字符串的长度，整数则忽略.>）。

代码实现

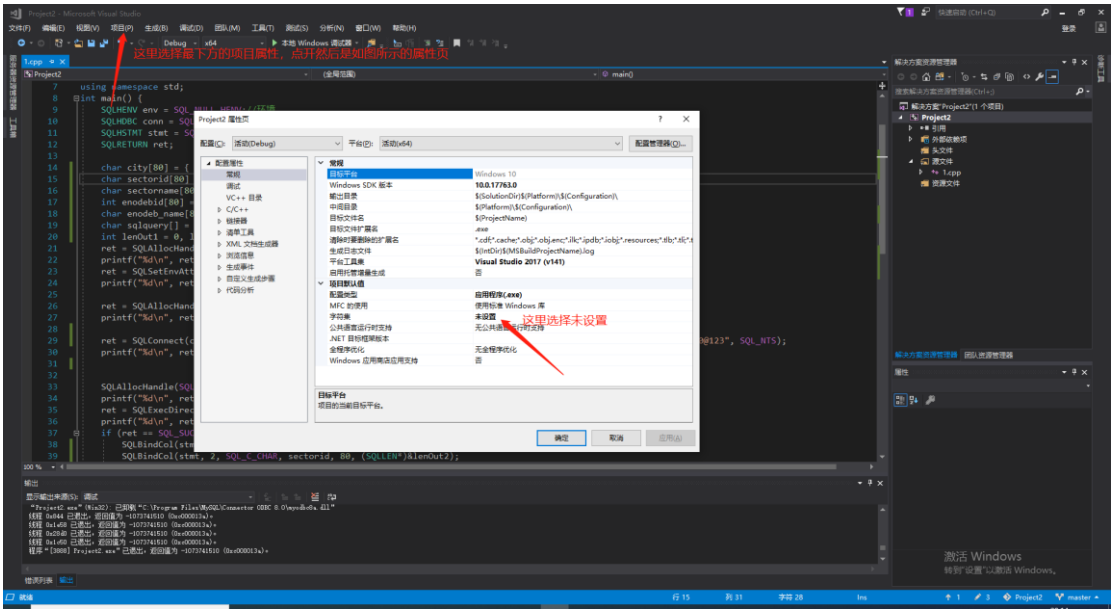
(这里的代码如果运行不通，使用 visual studio 创建项目时，应创建空项目，且

```
#include<windows.h>
#include<iostream>
#include <assert.h>
#include<sql.h>
#include <sqlext.h>
using namespace std;
int main() {
    SQLHENV env = SQL_NULL_HENV;//环境
    SQLHDBC conn = SQL_NULL_HDBC;//连接
    SQLHSTMT stmt = SQL_NULL_HSTMT;//语句
    SQLRETURN ret;

    char city[80] = { 0 };
    char sectorid[80] = { 0 };
    char sqlquery[] = "select * from `1.tbcell` where CITY='sanxia'";
    int lenOut1 = 0, lenOut2 = 1;
```

```
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    printf("%d\n", ret);//打印出来 ret 值为 0 即成功，-1 是失败
    ret = SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);
    printf("%d\n", ret);

    ret = SQLAllocHandle(SQL_HANDLE_DBC, env, &conn);
    printf("%d\n", ret);

    ret = SQLConnect(conn, (SQLCHAR*)"university", SQL_NTS, (SQLCHAR*)"bupt2018dbs50", SQL_NTS,
(SQLCHAR*)"bupt2018dbs50@123", SQL_NTS);//"university"与前面的数据源名字对应
    printf("%d\n", ret);
    SQLAllocHandle(SQL_HANDLE_STMT, conn, &stmt);
    printf("%d\n", ret);
    ret = SQLExecDirect(stmt, (SQLCHAR*)sqlquery, SQL_NTS);
    printf("%d\n", ret);
    if (ret == SQL_SUCCESS) {
        SQLBindCol(stmt, 1, SQL_C_CHAR, city, 80, (SQLLEN*)&lenOut1);
        SQLBindCol(stmt, 2, SQL_C_CHAR, sectorid, 80, (SQLLEN*)&lenOut2);
        printf("success");
        while (SQLFetch(stmt) == SQL_SUCCESS) {
            printf("%s %s\n", city, sectorid);
        }
    }
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);//释放语句
    SQLDisconnect(conn);//断开连接
    SQLFreeHandle(SQL_HANDLE_DBC, conn);//释放连接
    SQLFreeHandle(SQL_HANDLE_ENV, env);//释放环境
    getchar();
    return 0;
}
```



注意这里的 debug 应是 x64 而不是 x86，如果还不行，请按照如图所示）

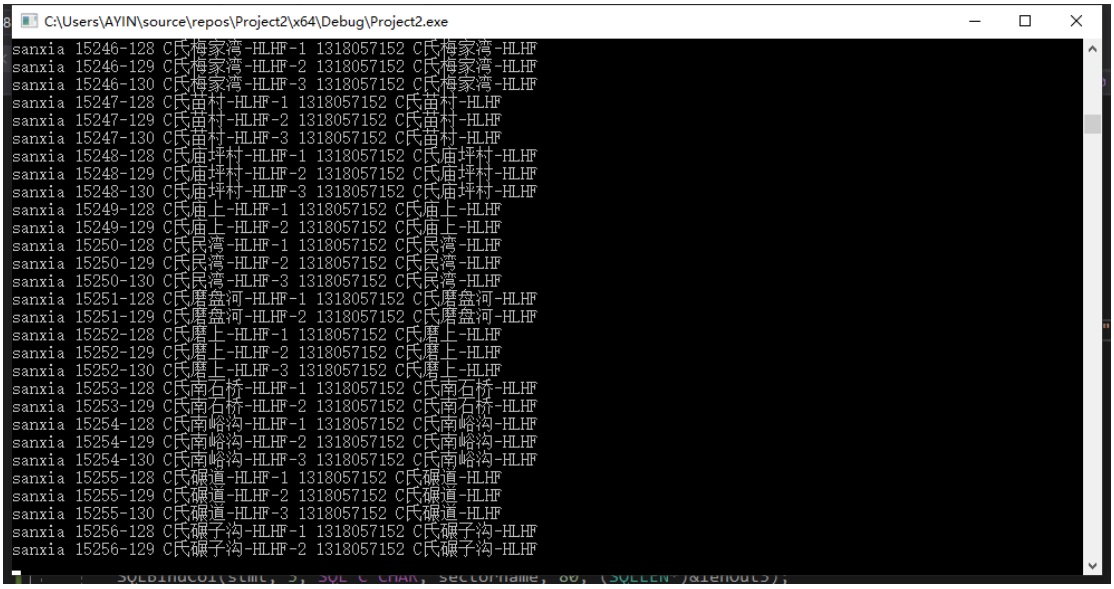


程序说明

a. 实验环境：OS：Windows10，编译器：vsstudio2017；

b. 在每一步都检查返回值是好习惯，便于程序失败时定位错误点，上述例子在运行时总是连接出错（后来证明是与所选的编译器有关，在创建新项目时需选择“空项目”），目前的检查比较粗略，可用 SQLGetDiagRec 获得更准确的信息。

c. 运行结果



注意：不同版本的 ODBC 驱动程序实现细节有所不同，可以对比上述访问程序与教科书上所给出的下样例在实现方法上的差异。

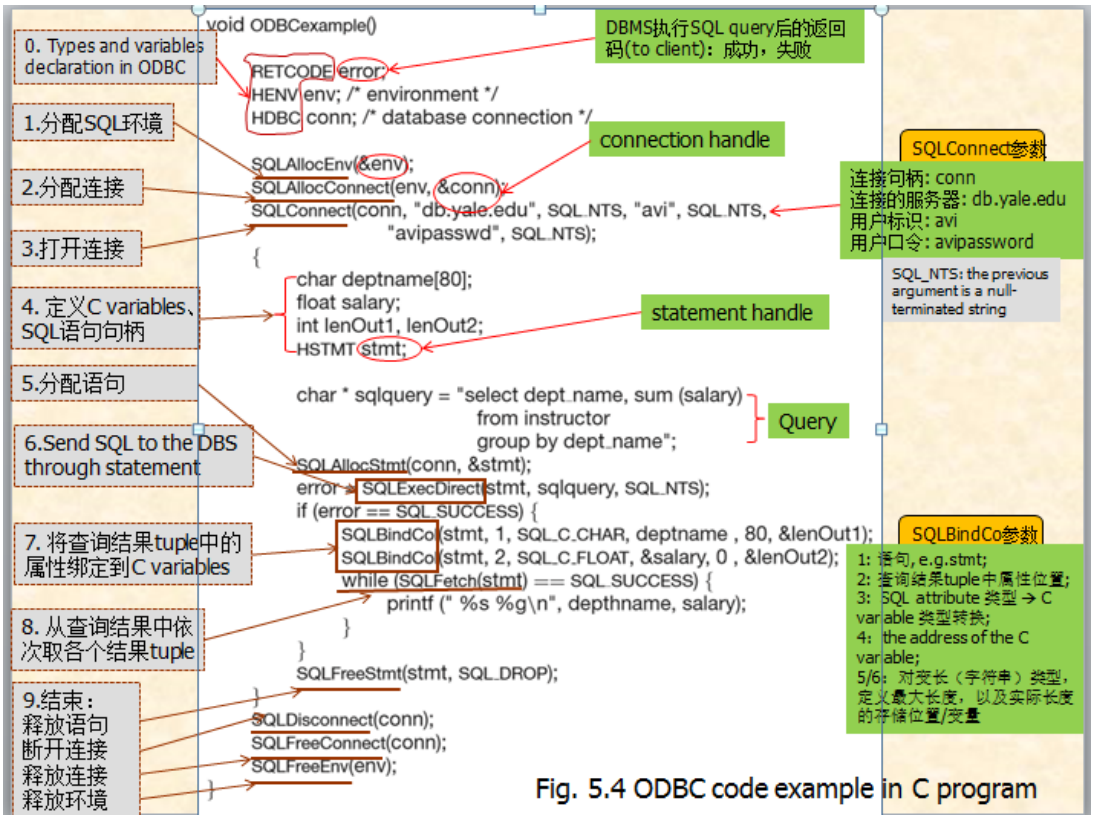


Fig. 5.4 ODBC code example in C program

4.1.4 其他数据库访问操作

（1）添加一条'beijing','000001', 'haidian-HLHF-1',000001,'haidian-HLHF'的数据到数据库。

```
char *sqlquery =”insert into `1.tbcell`(city, sector_id, sector_name, enodebid, ENODEB_NAME)
values('beijing','000001', 'haidian-HLHF-1',000001,'haidian-HLHF')”;
```

（2）查询，查 sanxia 城市中，经度（longitude）大于 111.5 的所有基站 id 和名字，经度（enodebid, enodeb_name,longitude),且按照经度降序排列注意相同的只显示一次。

```
char *sqlquery = “select ENODEBID,enodeb_name,LONGITUDE from `1.tbcell` where LONGITUDE>111.5 group
by ENODEBID order by LONGITUDE desc”;
```

（3）更新，将（1）中插入的数据，中的 earfcn，pci，pss，sss，tac 更新为 12345,32,1,10,10000,并打印该行信息

```
char *sqlquery=”update `1.tbcell` set EARFCN=12345,pci=32,pss=1,sss=10,tac=10000 where enodebid=1”;
```

（4）删除，删除（1）插入的信息，并打印整张表。

```
char *sqlquery=”delete from `1.tbcell` where CITY='beijing'”;
```

4.2 JDBC 接口访问

4.2.5 实验准备

下载 jdbc 驱动，mysql-connector-java-8.0.22.jar，下载地址：

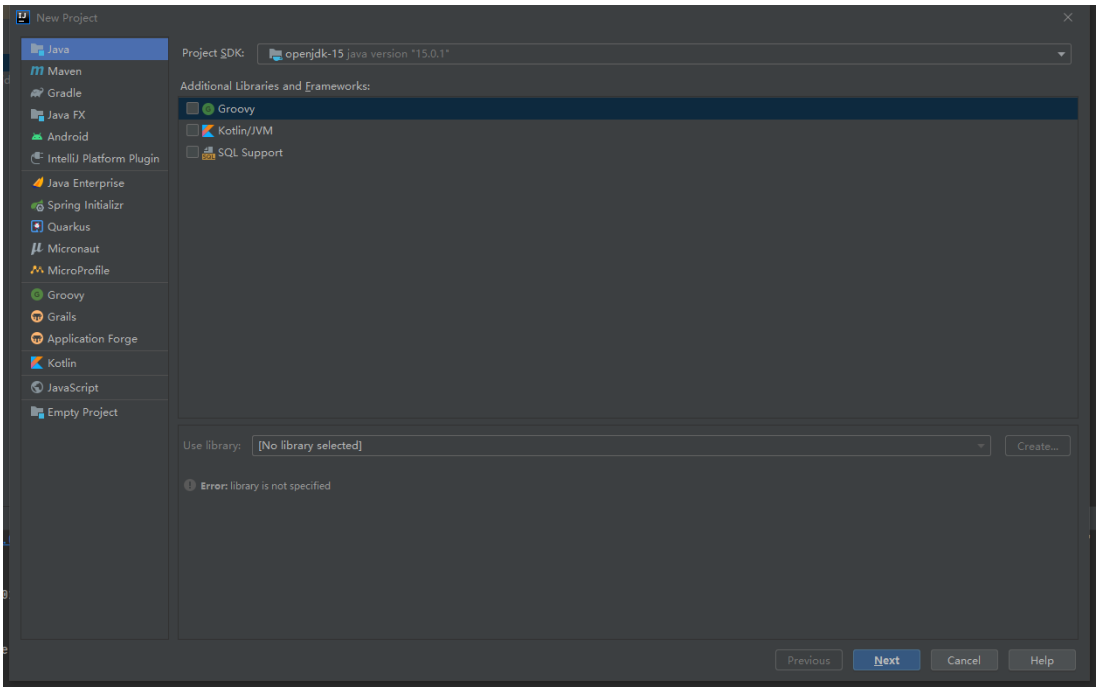
https://cdn.mysql.com//Downloads/Connector-J/mysql-connector-java-8.0.22.zip

下载之后将压缩包解压，查看文件夹中是否有 mysql-connector-java-8.0.22.jar 包

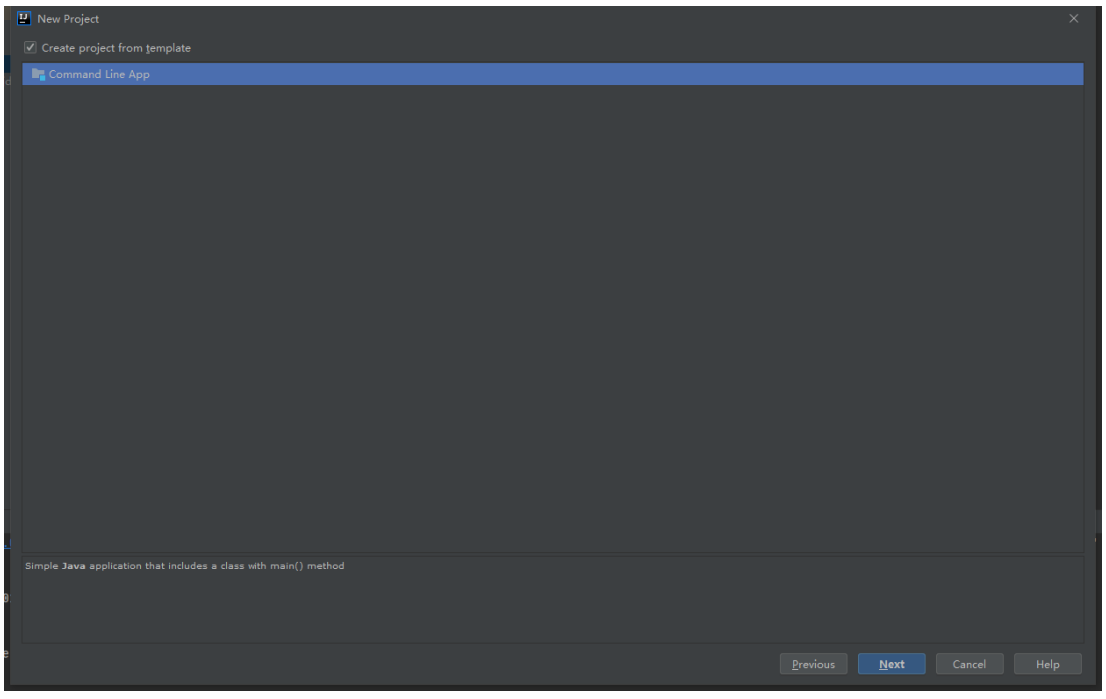
4.2.6 安装 JDBC 驱动

下面提供的样例为在开发环境 Intellij 中进行实验的步骤，学生可以尝试用其他 IDE，只要保证能顺利完成各项实验内容即可。

首先创建新的 java 项目。

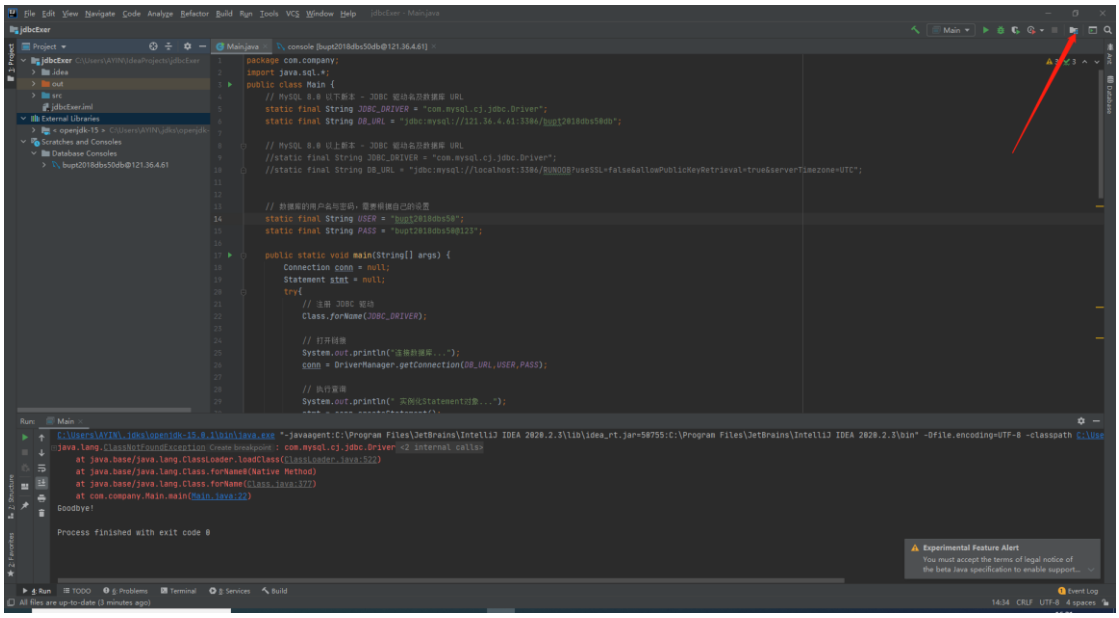


如图所示选择 Java 项目，选择 Project SDK 中 openjdk-15 点击 next

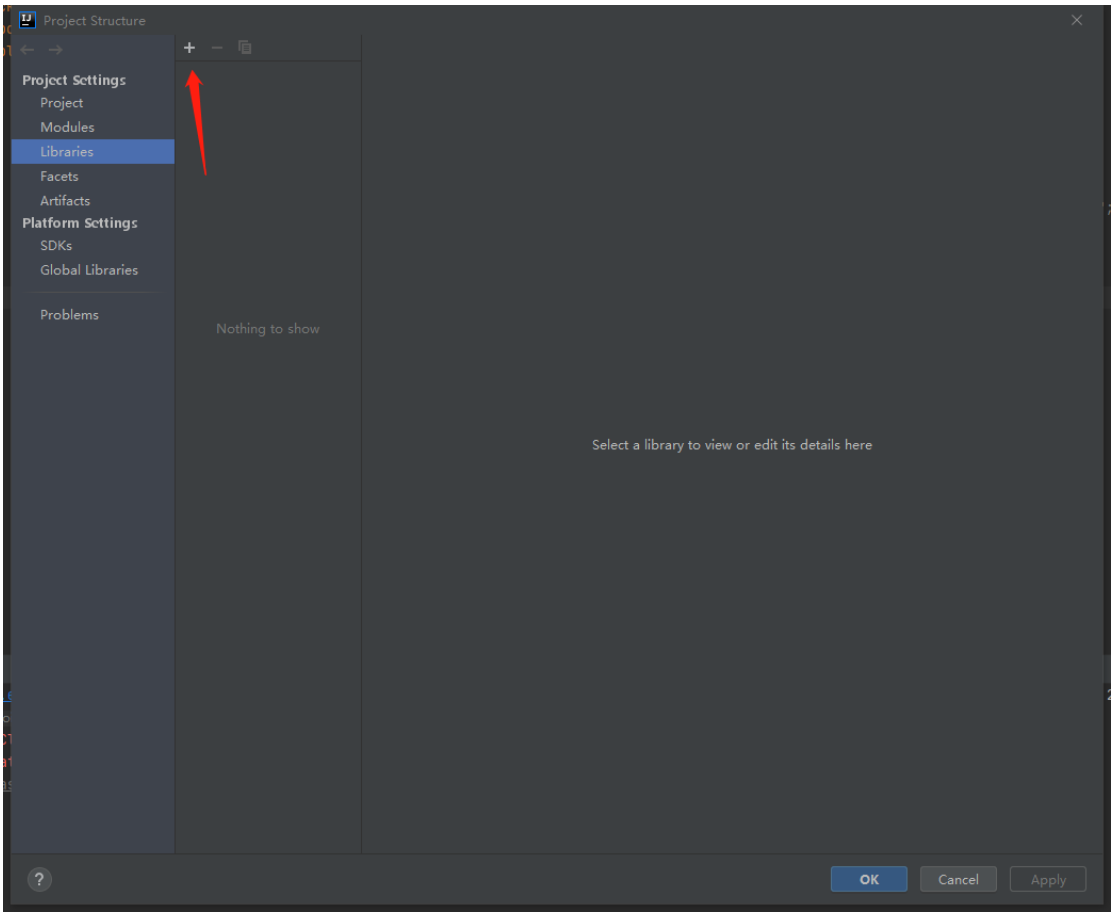


继续 next->finish 完成项目创建

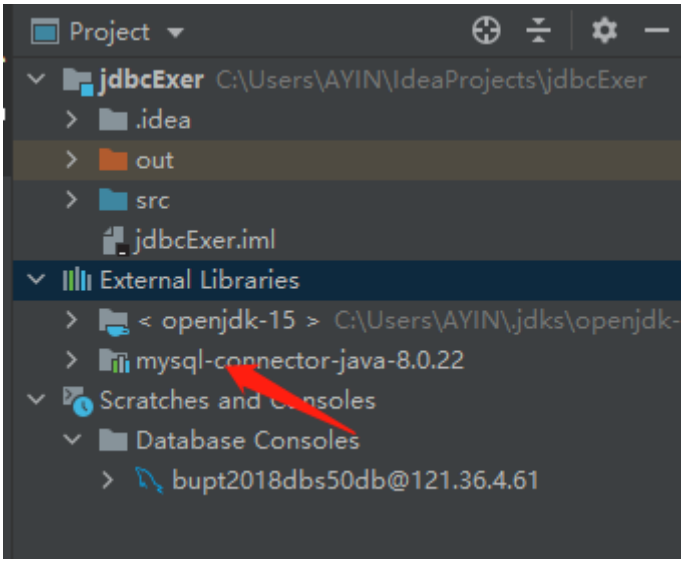
点击 project structure



点击+号，找到并选择自己下载之后解压文件夹中的 mysql-connector-java-8.0.22.jar 包



项目中出现这个说明导入成功



4.2.7 设置超时时间

```
sql = "show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');";
ResultSet rs = stmt.executeQuery(sql);
System.out.println(rs.getString(1)+" "+rs.getString(2));
```

获取 interactive_timeout 和 wait_timeout 的值

```
interactive_timeout 28800
wait_timeout 28800
Goodbye!
```

```
String sql1="set session INTERACTIVE_TIMEOUT=10000";
String sql="show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');";
stmt.executeQuery(sql1);
ResultSet rs = stmt.executeQuery(sql);
```

修改 interactive_timeout 的值之后再次获取 interactive_timeout 和 wait_timeout 的值:

```
interactive_timeout 10000
wait_timeout 28800
Goodbye!
```

4.2.8 编写 java 程序访问数据库

注意MySQL版本的问题，以MySQL8.0以上版本为例，加载驱动和URL路径的语句分别为

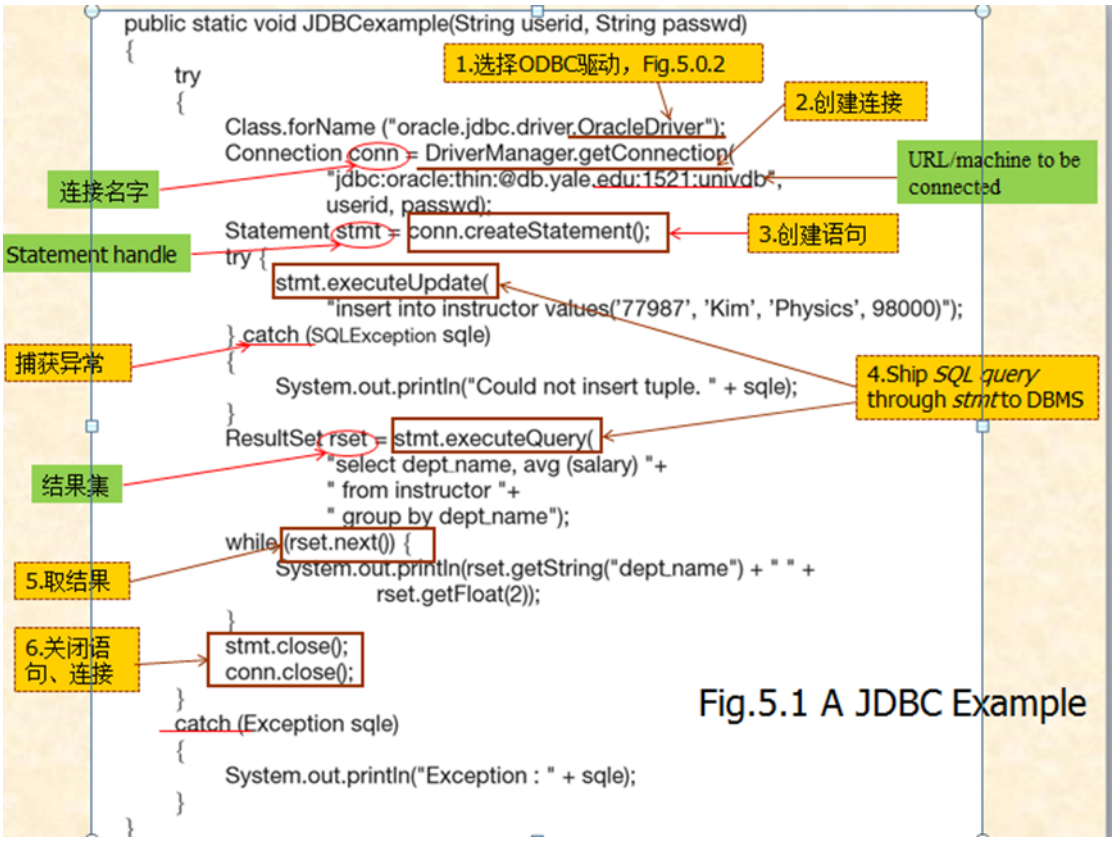
```
        static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://121.36.4.61:3306/bupt2018dbs50db";
```

全部代码为：

```
package com.company;
import java.sql.*;
public class Main {
    // MySQL 8.0 以下版本 - JDBC 驱动名及数据库 URL
    static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://121.36.4.61:3306/bupt2018dbs50db";

    // MySQL 8.0 以上版本 - JDBC 驱动名及数据库 URL
    //static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    //static final String DB_URL =
"jdbc:mysql://localhost:3306/RUNOOB?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";


    // 数据库的用户名与密码，需要根据自己的设置
    static final String USER = "bupt2018dbs50";
    static final String PASS = "bupt2018dbs50@123";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            // 注册 JDBC 驱动
            Class.forName(JDBC_DRIVER);
            // 打开链接
            System.out.println("连接数据库...");
            conn = DriverManager.getConnection(DB_URL,USER,PASS);
            // 执行查询
            System.out.println(" 实例化 Statement 对象...");
            stmt = conn.createStatement();
            String sql;
            sql = "select * from `1.tbcell` where CITY='beijing';";
            ResultSet rs = stmt.executeQuery(sql);
            // 展开结果集数据库
            while(rs.next()){
                // 通过字段检索
                String city    = rs.getString("city");
                String sector_id = rs.getString("sector_id");
                String sector_name = rs.getString("sector_name");
```

```
                // 输出数据
                System.out.print("city: " + city);
                System.out.print("    sector_id: " + sector_id);
                System.out.print("    sector_name:" + sector_name);
                System.out.print("\n");
            }
            // 完成后关闭
            rs.close();
            stmt.close();
            conn.close();
        }catch(SQLException se){
            // 处理 JDBC 错误
            se.printStackTrace();
        }catch(Exception e){
            // 处理 Class.forName 错误
            e.printStackTrace();
        }finally{
            // 关闭资源
            try{
                if(stmt!=null) stmt.close();
            }catch(SQLException se2){
            }// 什么都不做
            try{
                if(conn!=null) conn.close();
            }catch(SQLException se){
                se.printStackTrace();
            }
        }
        System.out.println("Goodbye!");
    }
```

注意：不同版本的JDBC驱动程序实现细节有所不同，可以对比上述访问程序与教科书上所给出的下样例在实现方法上的差异。



4.2.9 其他访问操作

（1）添加一条'beijing','000001', 'haidian-HLHF-1',000001,'haidian-HLHF'的数据到数据库。

```
sql=”insert into `1.tbcell`(city, sector_id, sector_name, enodebid, ENODEB_NAME) values('beijing','000001',
'haidian-HLHF-1',000001,'haidian-HLHF');”;
```

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS
1	beijing	000001	haidian-HLHF-1	1	haidian-HLHF	<null>	<null>	<null>

（2）查询，查 sanxia 城市中，经度（longitude）大于 111.5 的所有基站 id 和名字，经度（enodebid，enodeb_name,longitude),且按照经度降序排列注意相同的只显示一次。

	ENODEBID	enodeb_name	LONGITUDE
1	246333	G安水泥厂F-HLH	113.034
2	246506	G安芦院学校D-HLH	113.029
3	254642	G安铁门营业厅D-HLH	113.025
4	11429	G安玉梅F-HLH	113.024
5	236141	G安刘扬F-HLH	113.019
6	246341	G安千唐志斋F-HLH	113.018
7	246335	G安铁门镇F-HLH	113.018
8	254572	G安铁门村F-HLH	113.01
9	246703	F阳盐镇北F-HLH	113.006

```
sql="select ENODEBID,enodeb_name,LONGITUDE from `1.tbcell` where LONGITUDE>111.5 group by ENODEBID order by LONGITUDE desc;";
```

（3）更新，将（1）中插入的数据，中的 earfcn，pci，pss，sss，tac 更新为 12345,32,1,10,10000,并打印该行信息

```
sql="update `1.tbcell` set EARFCN=12345,pci=32,pss=1,sss=10,tac=10000 where enodebid=1"
```

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS	SSS	TAC
1	beijing	000001	haidian-HLHF-1	1	haidian-HLHF	12345	32	1	10	10000

（4）删除，删除（1）插入的信息，并打印整张表。

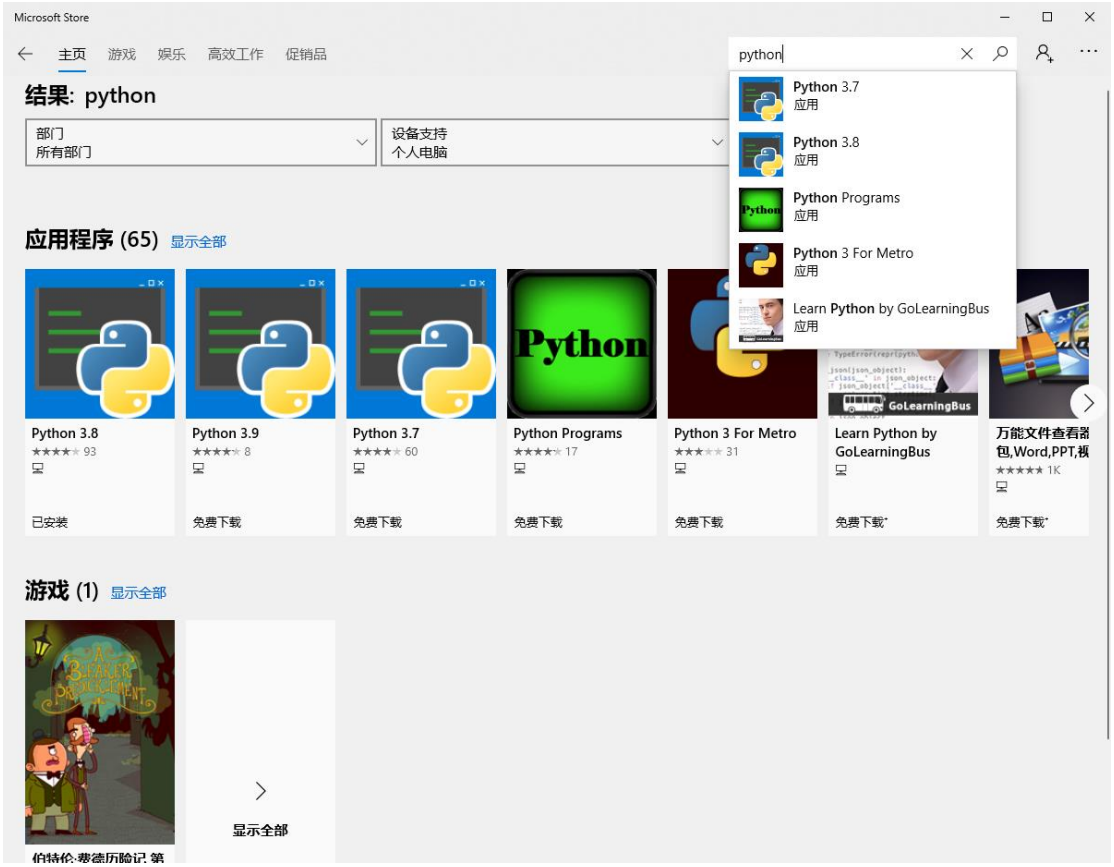
```
sql="delete from `1.tbcell` where CITY='beijing';"
```

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS
1	sanxia	124672-0	A池刘果-HLHF-1	124672	A池刘果-HLHF	38400	32	
2	sanxia	124672-1	A池刘果-HLHF-2	124672	A池刘果-HLHF	38400	30	
3	sanxia	124672-2	A池刘果-HLHF-3	124672	A池刘果-HLHF	38400	31	
4	sanxia	124673-0	A池张沟村-HLHF-1	124673	A池张沟村-HLHF	38400	200	
5	sanxia	124673-1	A池张沟村-HLHF-2	124673	A池张沟村-HLHF	38400	198	
6	sanxia	124673-2	A池张沟村-HLHF-3	124673	A池张沟村-HLHF	38400	199	
7	sanxia	124674-0	A池苏门-HLHF-1	124674	A池苏门-HLHF	38400	327	
8	sanxia	124674-1	A池苏门-HLHF-2	124674	A池苏门-HLHF	38400	329	
9	sanxia	124674-2	A池苏门-HLHF-3	124674	A池苏门-HLHF	38400	328	

4.3 Python/connector 接口访问

4.3.1 实验准备

下载 python3.8(或其他 3.+版本），可以选择通过 win10 系统微软商店安装，



选择一个安装，之后按 win+R，输入 cmd，打开命令行，输入 python --version

```
PS C:\Users\AYIN> python --version
Python 3.8.6
```

显示如下结果，即 python 安装成功，然后通过包管理器 pip 安装 mysql-connector-python

```
PS C:\Users\AYIN> python -m pip install  mysql-connector-python
```

Or

```
PS C:\Users\AYIN> pip install mysql-connector-python
```

之后输入 python 进入命令行

不报错，就说明 python/connector 安装成功

```
PS C:\Users\AYIN> python
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector
>>>
```

4.3.2 在 pycharm 或 vscode 中测试连接是否成功

首先创建一个 python 文件，编写如下代码：

```
import mysql.connector

cnx = mysql.connector.connect(user='bupt2018dbs50', password='bupt2018dbs50@123',
                              host='121.36.4.61',
                              database='bupt2018dbs50db')
print(cnx)
cnx.close()
```

显示如下的输出说明连接成功。

```
PS C:\Users\AYIN\Documents\python>  cd 'c:\Users\AYIN\Documents\python'; & 'C:
\debugpy\launcher' '60710' '--' 'c:\Users\AYIN\Documents\python\mysqlconnector
<mysql.connector.connection.MySQLConnection object at 0x0000028F68268850>
```

4.3.3 设置超时时间

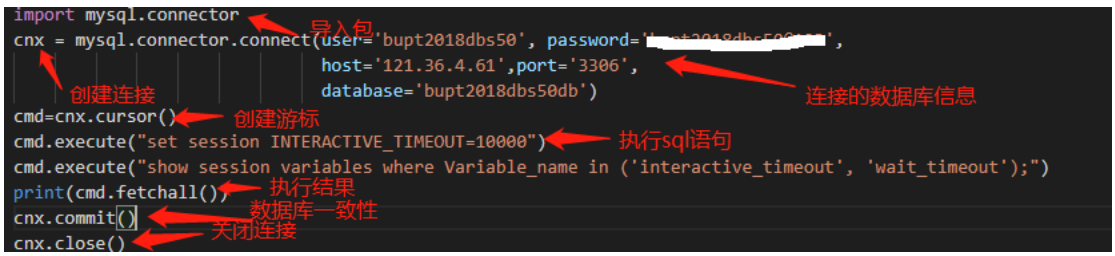
在 python/connector 中可以通过 cmd_query 函数来执行 sql 语句，因此，

```
import mysql.connector

cnx = mysql.connector.connect(user='bupt2018dbs50', password='bupt2018dbs50@123',
                              host='121.36.4.61',
                              database='bupt2018dbs50db')
cnx.cmd_query("set session INTERACTIVE_TIMEOUT=10000")
cnx.commit()//数据库有更新必须用到此语句
cnx.close()
```

可以通过上述方式来设置 interactive_timeout 参数。

4.3.4 编写 python 程序访问数据库

代码逻辑与所用部分函数说明如下图：



全部代码为：

```
import mysql.connector
cnx = mysql.connector.connect(user='bupt2018dbs50', password='bupt2018dbs50@123',
                              host='121.36.4.61',port='3306',
                              database='bupt2018dbs50db')
cmd=cnx.cursor()
cmd.execute("show session variables where Variable_name in ('interactive_timeout', 'wait_timeout');")
print(cmd.fetchall())
cnx.commit()
cnx.close()
```

4.3.5 通过 python/connector 接口访问数据库

（1）添加一条'beijing','000001', 'haidian-HLHF-1',000001,'haidian-HLHF'
的数据到数据库。

```
cmd.execute("insert into `1.tbcell`(city, sector_id, sector_name, enodebid, ENODEB_NAME) values('beijing','000001',
'haidian-HLHF-1',000001,'haidian-HLHF');")
```

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS
1	beijing	000001	haidian-HLHF-1	1	haidian-HLHF	<null>	<null>	<null>

（2）查询，查 sanxia 城市中，经度（longitude）大于 111.5 的所有基站 id 和名字，经度（enodebid, enodeb_name,longitude),且按照经度降序排列注意相同的只显示一次。

```
cmd.execute("select ENODEBID,enodeb_name,LONGITUDE from `1.tbcell` where LONGITUDE>111.5 group by
ENODEBID order by LONGITUDE desc")
```

	ENODEBID	enodeb_name	LONGITUDE
1	246333	G安水泥厂F-HLH	113.034
2	246506	G安芦院学校D-HLH	113.029
3	254642	G安铁门营业厅D-HLH	113.025
4	11429	G安玉梅F-HLH	113.024
5	236141	G安刘扬F-HLH	113.019
6	246341	G安千唐志斋F-HLH	113.018
7	246335	G安铁门镇F-HLH	113.018
8	254572	G安铁门村F-HLH	113.01
9	246703	F阳盐镇北F-HLH	113.006

（3）更新，将（1）中插入的数据，中的 earfcn，pci，pss，sss，tac 更新为 12345,32,1,10,10000,并打印

该行信息。

```
cmd.execute("update `1.tbcell` set EARFCN=12345,pci=32,pss=1,sss=10,tac=10000 where enodebid=1")
```

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS	SSS	TAC
1	beijing	000001	haidian-HLHF-1	1	haidian-HLHF	12345	32	1	10	10000

（4）删除，删除（1）插入的信息，并打印整张表。

```
cmd.execute("delete from `1.tbcell` where CITY='beijing';")
```

1-500 of 501+ Tx: Auto DDL DML Comma-...d (CSV)

	CITY	SECTOR_ID	SECTOR_NAME	ENODEBID	ENODEB_NAME	EARFCN	PCI	PSS
1	sanxia	124672-0	A池刘果-HLHF-1	124672	A池刘果-HLHF	38400	32	
2	sanxia	124672-1	A池刘果-HLHF-2	124672	A池刘果-HLHF	38400	30	
3	sanxia	124672-2	A池刘果-HLHF-3	124672	A池刘果-HLHF	38400	31	
4	sanxia	124673-0	A池张沟村-HLHF-1	124673	A池张沟村-HLHF	38400	200	
5	sanxia	124673-1	A池张沟村-HLHF-2	124673	A池张沟村-HLHF	38400	198	
6	sanxia	124673-2	A池张沟村-HLHF-3	124673	A池张沟村-HLHF	38400	199	
7	sanxia	124674-0	A池苏门-HLHF-1	124674	A池苏门-HLHF	38400	327	
8	sanxia	124674-1	A池苏门-HLHF-2	124674	A池苏门-HLHF	38400	329	
9	sanxia	124674-2	A池苏门-HLHF-3	124674	A池苏门-HLHF	38400	328	

5． 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。