**数据库系统project报告**

2021-2022学年第2学期（CST31106）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据库系统project任务书 | | | | | |
| 名称 | 数据库存储管理设计与模拟实现 | | | | |
| 类型 | □验证性 □设计性 综合性 | | | | |
| 内容 | 关系型数据库采用自描述的方式进行数据库系统管理，关于关系的相关数据也存储在数据库管理系统里。  针对关系型数据库存储模块，进行需求分析，并设计一个简单的数据库系统存储模块，根据设计模拟实现数据存储管理器，主要实现：数据库创建、表格创建、数据添加、删除、更新等操作过程中，数据库系统所进行的存储方面的管理；设计索引，并比较有索引和无索引的区别。  模拟实现采用：python或者Java实现具体功能，数据存储采用excel或者文本文件，可用文本文件模拟硬盘数据块，不需要实现SQL语句的执行编译，SQL语句采用函数实现。 | | | | |
| 要求 | （1）设计方案要合理；  （2）能基于该存储管理模块实现数据的存储和读取；  （3）设计方案有一定的效率分析。 | | | | |
| 任务时间 | 2022年4月28日至2022年5月20日 | | | | |
| 小组成员 | | | | | | |
| 20194208 | | 20194068 | 20194196 | 20194297 | 20194301 | |
| 王彦景 | | 张佳莹 | 谢双骏 | 张鸿宇 | 赵显聪 | |
| 项目评分表 | | | | | | |
| 序号 | | 评分项 | 分值 | 得分 | | |
| 1 | | 需求分析 | 3分 |  | | |
| 2 | | 综合设计与实现 | 4分 |  | | |
| 3 | | 团队协作 | 3分 |  | | |
| 项目总得分： | | | | | | |

课程项目评分标准（总分10分）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 评分项目 | 完成情况 | 得分 |
| 1 | 需求分析 | 分析合理 | 3分 |
| 分析较合理 | 2分 |
| 分析不合理 | 1分 |
| 分析完全错误 | 0分 |
| 2 | 综合设计与实现 | 设计完整，设计合理，工具使用熟练 | 4分 |
| 设计较完整，设计合理，工具使用较熟练 | 3分 |
| 设计较完整，设计较合理，工具使用较熟练 | 2分 |
| 设计较完整，设计不合理，工具使用不熟练 | 1分 |
| 抄袭、被抄袭 | 0分 |
| 3 | 团队协作 | 有团队，分工合理，密切协作 | 3分 |
| 有团队，分工合理，有一定协作 | 2分 |
| 有团队，分工不合理，无协作 | 1分 |
| 无团队，无协作 | 0分 |

目录

[第一章 前言 5](#_Toc104564953)

[一、 项目背景介绍 5](#_Toc104564954)

[二、 小组分工及实现思路 5](#_Toc104564955)

[第二章 需求分析 6](#_Toc104564956)

[第三章 综合设计 8](#_Toc104564958)

[一、 元数据及数据存储设计 8](#_Toc104564959)

[二、 具体代码实现 10](#_Toc104564960)

[2.1 代码实现思路 10](#_Toc104564961)

[2.2 添加模块 10](#_Toc104564962)

[2.3 删除模块 17](#_Toc104564966)

[2.4 修改模块 21](#_Toc104564970)

[2.5 查询模块 21](#_Toc104564972)

[第四章 功能测试 24](#_Toc104564977)

[第五章 总结 32](#_Toc104564977)

1. 前言
2. **项目背景介绍**

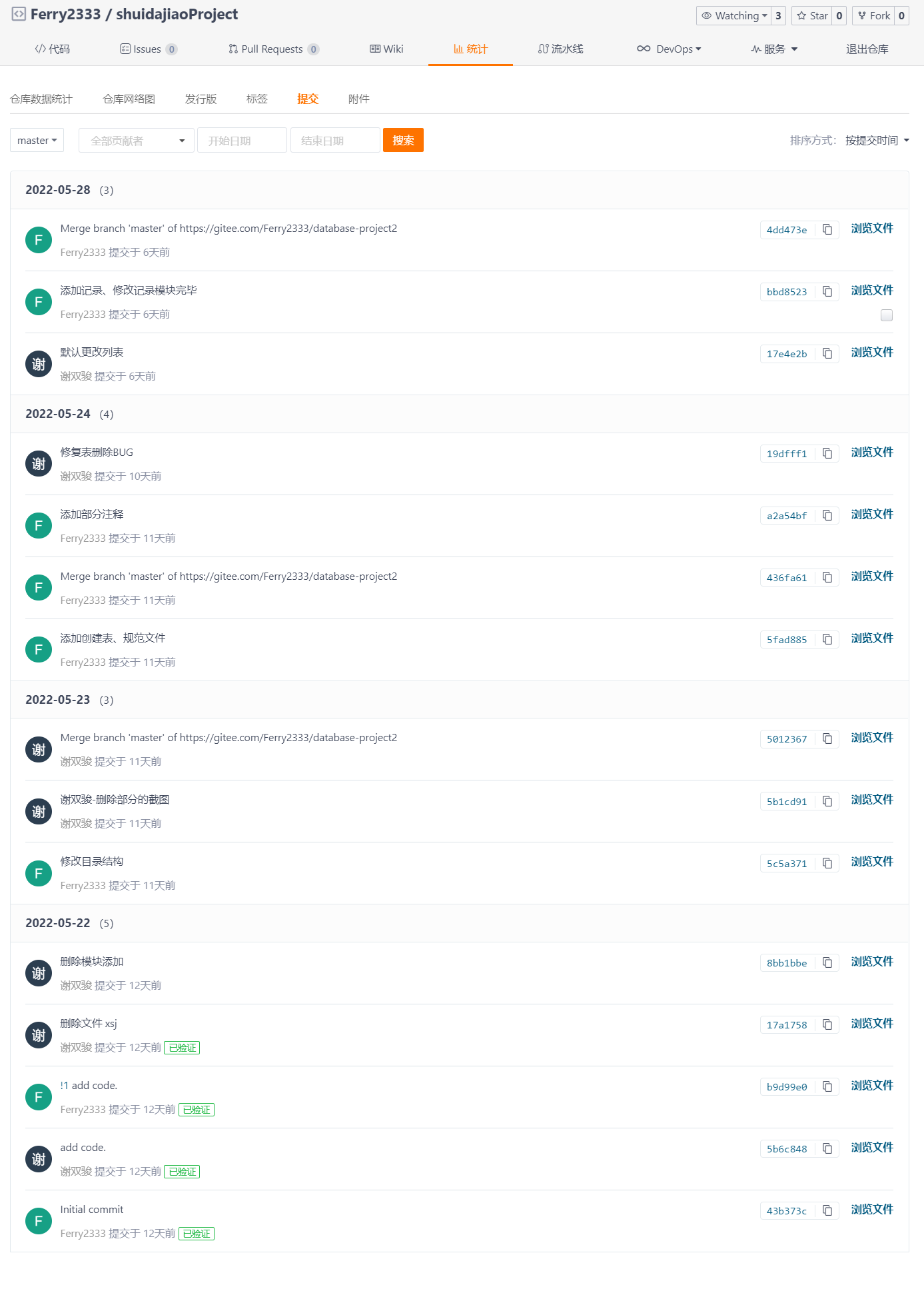
随着信息技术在各行各业得到广泛应用，有效地管理和存取大量数据资源变得尤为重要。数据库技术经不断发展，技术不断进步，对很多行业的发展起到极大地推动作用，解决了数据冗余不一致、查询困难关联度小和无法并发操作的问题。目前常见的数据库有关系型数据库和非关系型数据库，本项目参考关系型数据库设计并实现了数据库存储系统，采用关系模型存储数据，置于一组拥有正式描述性的表格，该表格装载着数据项的特殊收集体，可通过不同的方式存取或重新召集而不需要重新组织数据库表格。

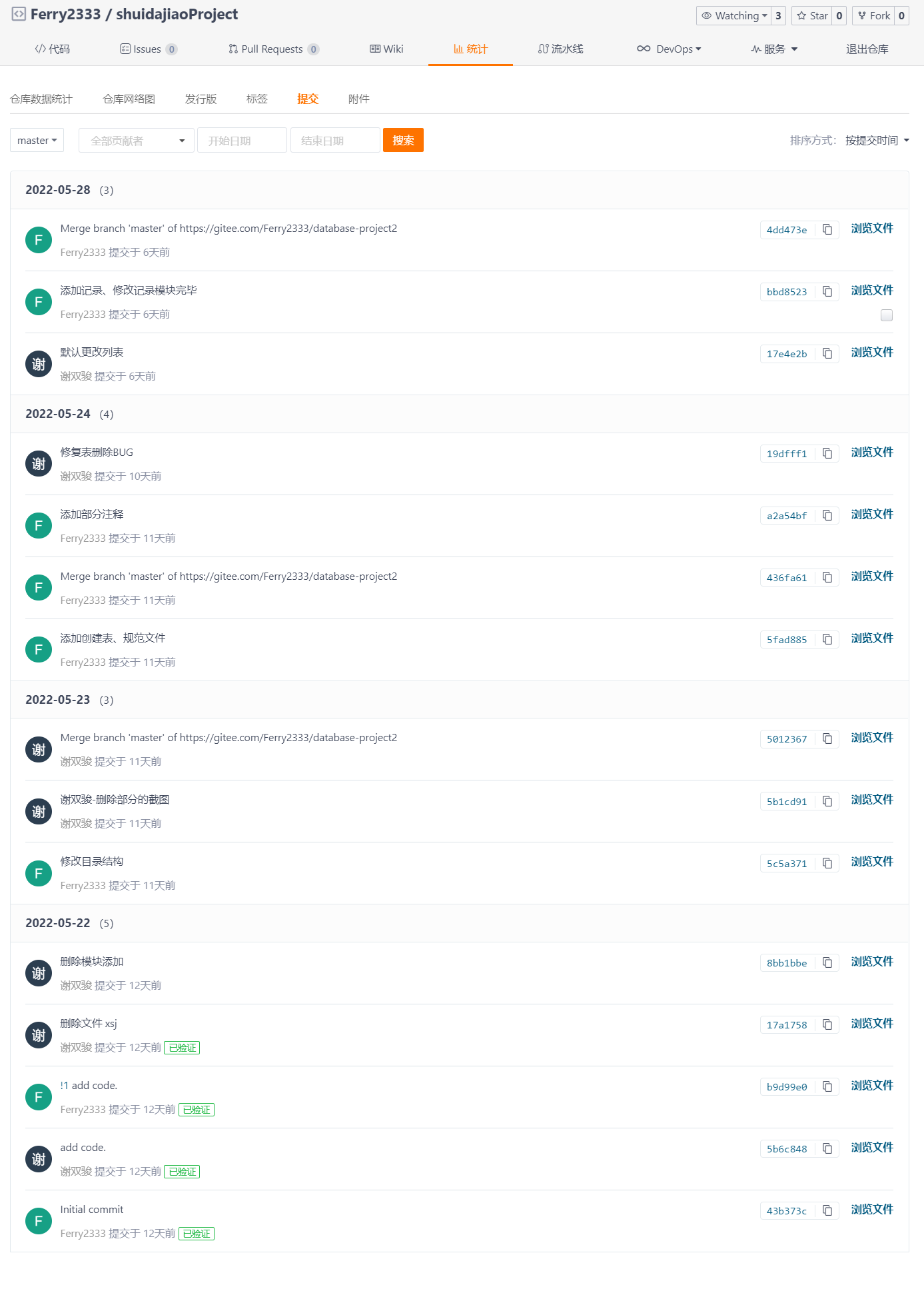
1. **小组分工及实现思路**

小组所有成员共同讨论出需求，根据需求设计数据库的功能，随后确定所用编程语言为python，使用pandas第三方库对csv文件进行增删改查模拟数据存储管理器操作，设定用户可以使用的语法与MySQL相同。

对于数据存储管理器的具体实现，添加模块、查询模块以及代码结构的搭建由王彦景和赵显聪完成；删除模块由谢双骏完成；修改模块以及添加记录的功能由张鸿宇完成；并且分别撰写项目报告的代码实现部分；最终的完整项目报告由张佳莹完成。

本项目全程使用gitee托管代码，方便各功能独立开发和代码整合。提交日志如下图所示：





1. **需求分析**

针对项目要求，我们小组进行需求分析，目标是明确数据存储管理器的功能需求，便于后续的综合设计。小组经过充分讨论，参考软件工程的设计流程，首先绘制了用例图，用于描述参与者和用例之间的关系。本数据存储管理器中，参与者只有一个，就是使用本数据存储管理器的用户；用例设计有11个，每个用例涉及的操作都需要被数据存储管理器的元数据进行组织、记录或修改。每个用例分析如下：

* **创建数据库**：用户可以通过create database db\_name语句创建名为db\_name的数据库；
* **创建数据表**：用户可以通过use db\_name语句选择名为db\_name的数据库，随后使用create table tb\_name(type value……)在db\_name数据库下创建表；
* **添加记录：**用户可以通过insert语句向选定数据库的表中添加记录；
* **删除数据库：**用户可以通过drop database db\_name删除名为db\_name的数据库；
* **删除数据表：**用户可以通过drop table tb\_name删除名为tb\_name的数据表；
* **删除记录：**用户可以通过delete语句在数据表中删除指定条件的记录；
* **修改记录：**用户可以通过update语句更新数据表中指定条件的记录；
* **查询记录：**用户可以通过select语句查询数据表中指定条件的记录；
* **查看数据库：**用户可以通过show databases语句查看现有的数据库；
* **查看数据表：**用户可以通过show tables语句查看某个数据库下面现有的数据表；
* **查看元数据：**用户可以通过show meta data语句查看数据存储管理器中的元数据；

绘制完整的用例图如图2.1所示：

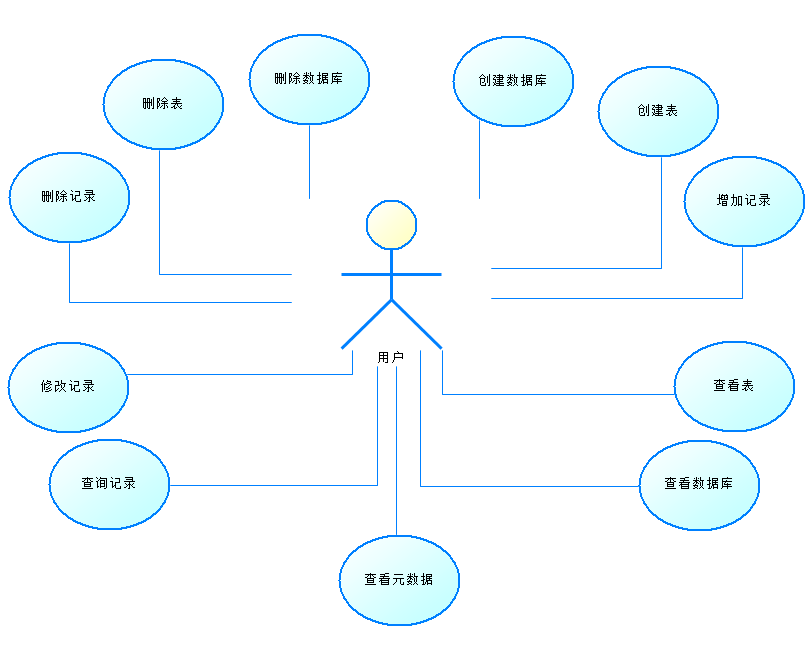


图2.1 用例图

1. **综合设计**
2. **元数据及数据存储设计**

数据库的元数据就是指定义数据库各类对象结构的数据，我们设计了两个csv文件作为元数据表来存储本数据存储管理器的元数据，分别是database\_meta\_data.csv存储数据库元数据，table\_meta\_data.csv存储数据表元数据。通过讲db\_id设置为table\_meta\_data表的外键来将两张元数据表关联起来。两张元数据表的具体字段说明如下：

表1 database\_meta\_data表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **约束** | **说明** |
| db\_id | INTEGER | PRIMARY KEY | 数据库id |
| db\_name | VARCHAR(100) | UNIQUE NOT NULL | 数据库名称 |
| create\_time | VARCHAR(100) | NOT NULL | 数据库创建时间 |
| is\_del | BOOLEAN | NOT NULL | 数据库是否被删除 |

表2 table\_meta\_data表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **字段类型** | **约束** | **说明** |
| db\_id | INTEGER | FOREIGN KEY | 数据库id  注：本字段引用自database\_meta\_data表的db\_id字段。 |
| table\_id | VARCHAR(100) | PRIMARY KEY | 数据表id |
| table\_name | VARCHAR(100) | UNIQUE NOT NULL | 数据表名称 |
| column\_list | VARCHAR | NOT NULL | 数据表属性 |
| type\_list | VARCHAR | NOT NULL | 数据表属性类型 |
| primary\_key | VARCHAR(100) | NOT NULL | 主键 |
| foreign\_key | VARCHAR(100) |  | 外键 |
| row\_num | INTEGER | NOT NULL | 数据表记录个数 |
| size\_in\_byte | INTEGER | NOT NULL | 数据表大小 |
| modify\_time | VARCHAR(100) | NOT NULL | 修改时间 |
| create\_time | VARCHAR(100) | NOT NULL | 创建shij |
| uid | INTEGER | NOT NULL | 用户id |
| gid | INTEGER |  | 组id |
| is\_del | BOOLEAN | NOT NULL | 数据表是否被删除 |

当用户登录进入数据库存储管理器后，可以使用create database语句创建数据库，也可以使用use database选择一个已有的数据库进行下面的操作。若使用create database db\_name语句，则创建一个名为db\_name的文件夹作为db\_name数据库的存储区域；同时在database\_meta\_data.csv文件中添加一行记录，包括db\_name数据库的id（db\_id），名称（db\_name）、创建时间（create\_time）以及删除标记初始化为false（is\_del）。

当用户使用create table tb\_name语句创建数据表时，会在选定的数据库文件夹下面创建一个名为tb\_name的csv文件，作为tb\_name表的存储区域；同时在table\_meta\_data.csv文件中添加一行记录，包括选定的数据库id（db\_id）、创建的数据表id（table\_id）、数据表名称（table\_name）、数据表中的属性（column\_list）、属性类型（type\_list）、该数据表的主键（primary\_key）、与该数据表关联的外键（foreign\_key）、该数据表所含的记录个数初始化为0（row\_num）、数据表的大小初始化为0（size\_in\_byte）、修改时间（modify\_time）、创建时间（create\_time）、用户id（uid）、组id（gid）、删除标记初始化为false（is\_del）。

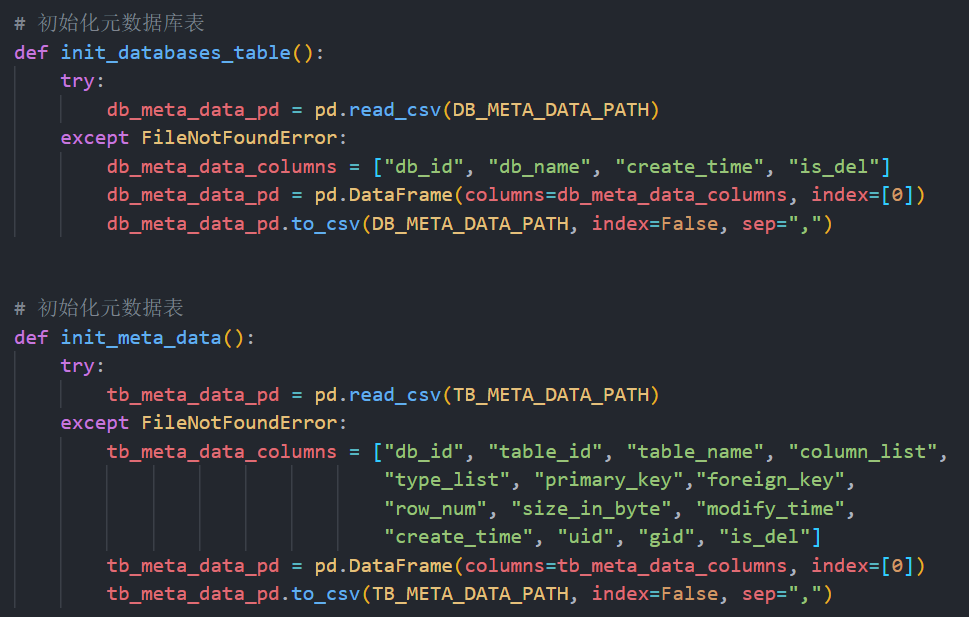
当用户使用其他命令时，修改两个元数据文件中的相关信息即可。其中删除操作我们设置为软删除，即当用户使用drop语句删除数据库或数据表时，仅仅将元数据表中的is\_del字段修改为true，这样更为安全，也更符合数据库设计思想。

1. **具体代码实现**

**2.1 代码实现思路**

在具体的代码中，我们将整个程序实现分为三步：

1. 首先初始化两个元数据表，程序读取database\_meta\_data.csv与table\_meta\_data.csv文件，若读取失败，则主动创建两个元数据表。



1. 其次当用户输入SQL语句时，程序对语句进行词法单元分析与语法分析，按照空格、左右括号、等号进行语句分割，得到每个SQL语句的词法单元，存储到tokens列表中，对词法单元进行语法分析，判断具体执行哪种操作。
2. 我们将数据存储管理器的功能划分为四个模块，分别是添加模块、删除模块、修改模块与查询模块，语法分析结束便可以调用对应模块的代码进行操作。其中：
   1. 添加模块包括：创建数据库、创建数据表、添加记录
   2. 删除模块包括：删除数据库、删除数据表、删除记录
   3. 修改模块包括：修改记录
   4. 查询模块包括：查看数据库、查看数据表、查看元数据、查询记录

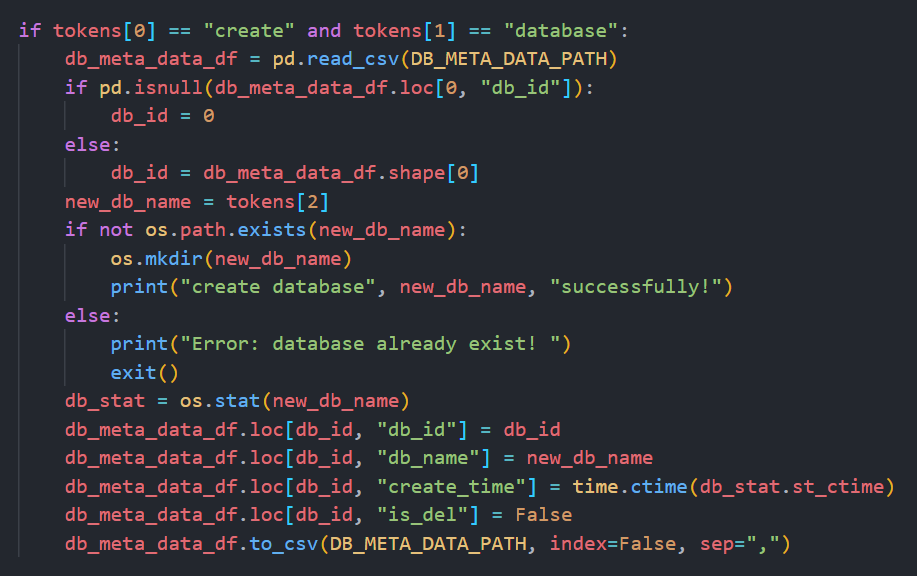
下面分别对上述模块的具体实现进行介绍。

**2.2 添加模块**

**2.2.1 创建数据库**

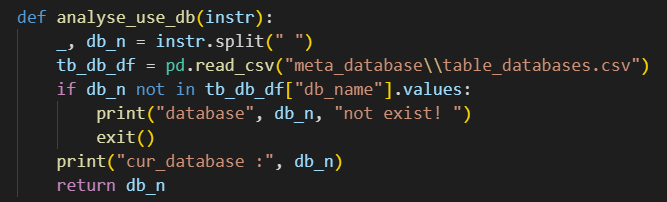
**① create database**

当用户输入create database指令时，程序首先会进入对应的分支，然后用户再输入数据库名称，程序在判断新建的数据库是否存在，若改数据库不存在则程序更新元数据表中的信息，然后再新建数据库，输出创建成功信息；若数据库已经存在，则输出数据库已经存在信息。

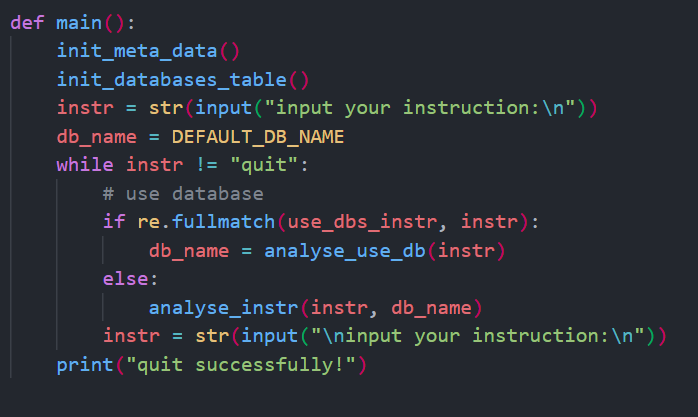


**② use database**

本项目的设计逻辑是一个文件夹代表一个数据库，一个csv文件代表一张关系表，所以在用户想要切换数据库时，只需要输入use [db\_name] 指令，程序就会把路径设置为相应数据库路径（在数据库db\_name 已经被创建的前提下）。解析Use database 指令的函数如下



Use database 指令运行逻辑如下：若程序识别到指令为use database指令，则会把db\_name 设置为相应的数据库，然后再传入到analyse\_sentence 函数中，以便执行后续操作。

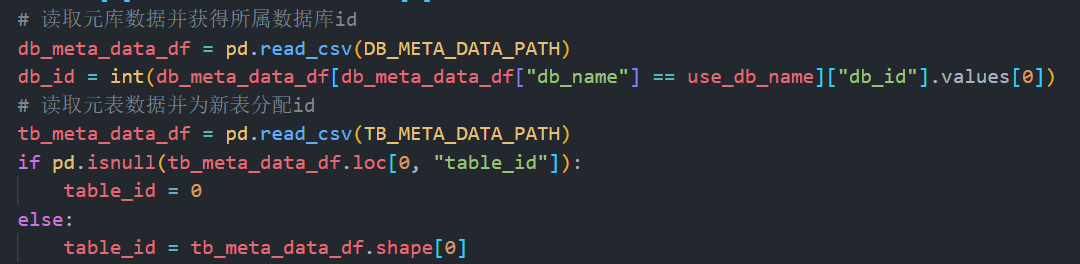


**2.2.2 创建数据表**

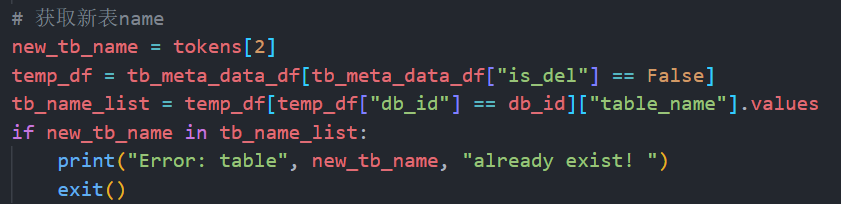
当用户使用create table tb\_name(type1 col1, type2 col2, primary key(col1), foreign key(cid) references course(cid))型语句创建名为tb\_name的数据表时，分析词法单元的第一个与第二个是否为create和table，是则执行下面的创建操作。



下面读取database\_meta\_data.csv得到数据库元数据，根据用户选择的数据库或者默认数据库名称获取对应数据库id；再读取table\_meta\_data.csv得到数据表元数据，为即将要新创建的数据表分配table\_id。



下面从词法单元中获取新表的名称并判断对应数据库中是否有未被删除且重名的表，进行判错。

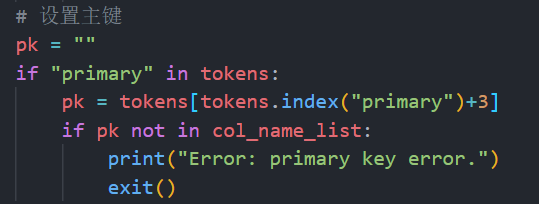


下面从词法单元中获取新表的属性和对应的属性类型，并且判断属性数量与类型数量是否相同、类型是否合法。





下面从词法单元中获取主键并进行设置，同时判断主键属性是否存在。



下面从词法单元中获取外键，并且读取外键关联的表，进行与外键相关的几个判错，代码注释中有详细说明。





下面创建以table\_name命名的csv文件，并将该表相关的数据更新到table\_meta\_data元数据表中，创建数据表完毕。



**2.2.3 添加记录**

添加记录的基本思路如下：

（1）先对insert的sql语句进行正则匹配，判断语句是否是insert语句：

#insert语句

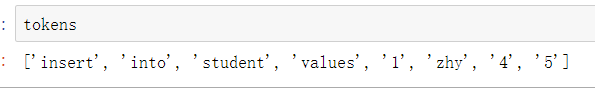
insert\_instr="insert\\b\\sinto\\b\\s[a-zA-Z0-9\_]+\\b\\svalues\((([0-9]\*)|([a-zA-Z0-9\_]\*))?(\,([0-9]\*)|([a-zA-Z0-9\_]\*))\*\)"

（2）成功匹配后对语句进行解析

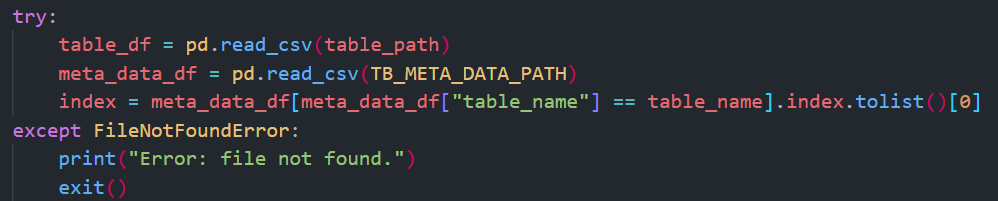
elif re.fullmatch(insert\_instr, instr):

tokens=[i for i in re.split(r"([ ,();=])", instr.lower().strip()) if i not in[' ',',',')','(','']]

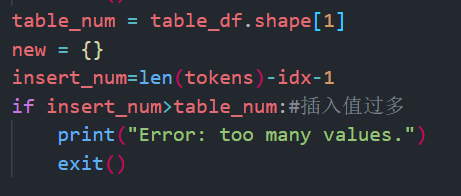
（3）举例结果如下，此时获得表名和待插入的数据



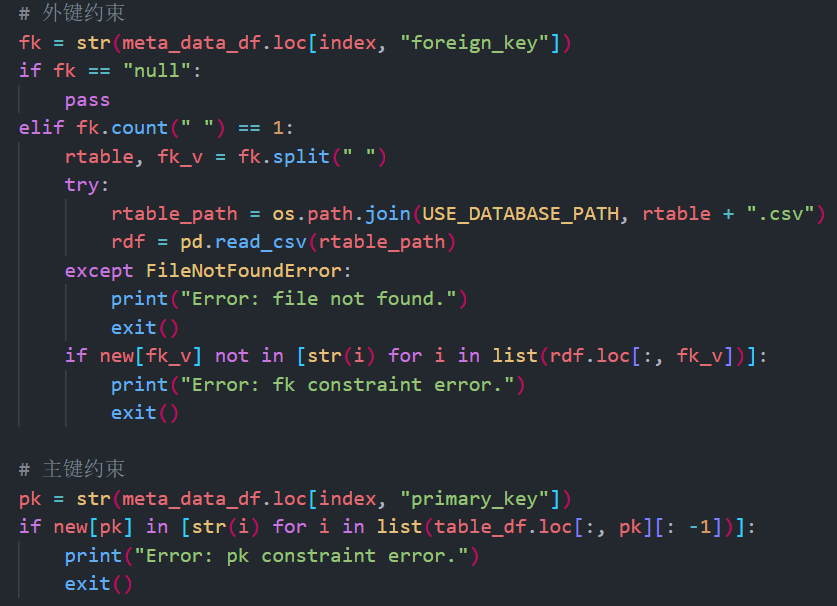
1. 读取csv数据文件

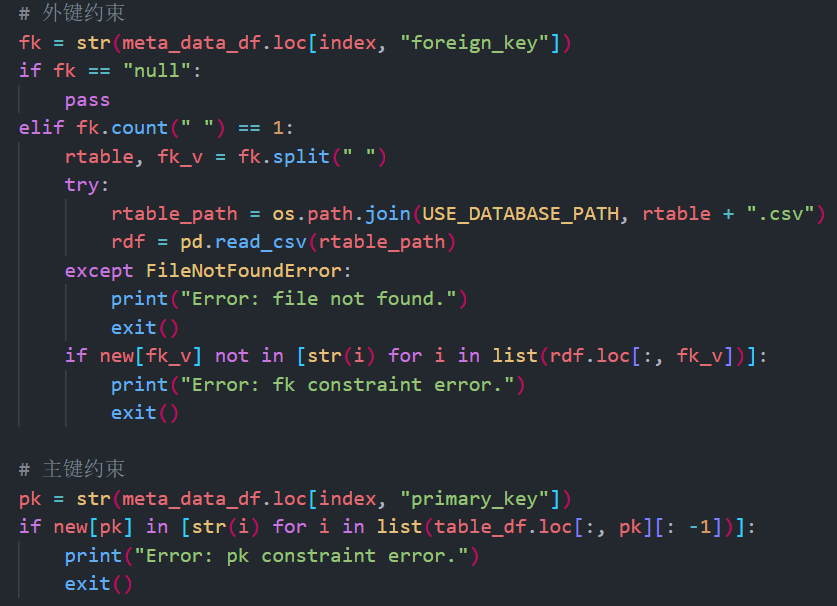


1. 检查插入数据是否过多，如果插入值多于表的属性，则报错

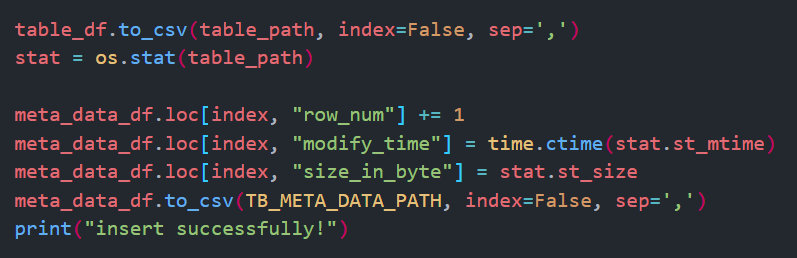


（5）外键和主键检查，查看是否合法





（6）保存数据并更新修改时间、数据大小等信息

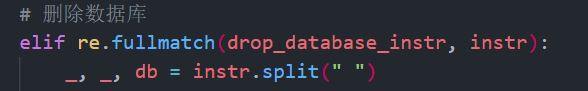


**2.3 删除模块**

**2.3.1 删除数据库**

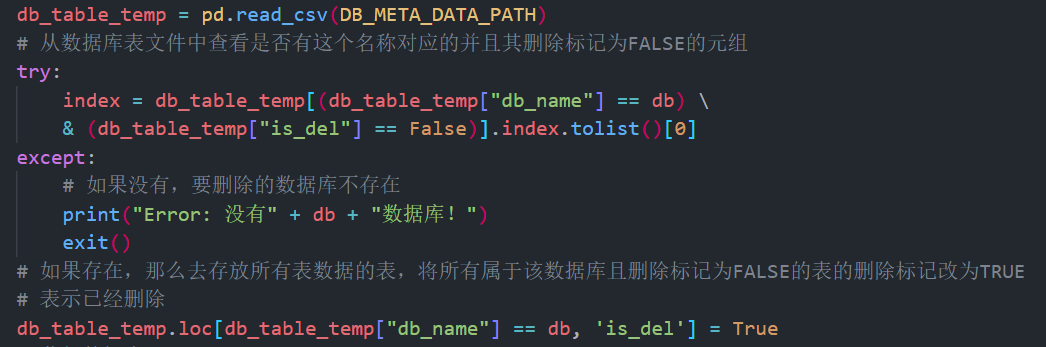
删除数据库的基本思路如下：

（1）解析sql语句，如果符合删除数据库的正则表达式，那么从语句中得到要删除的数据库的名称



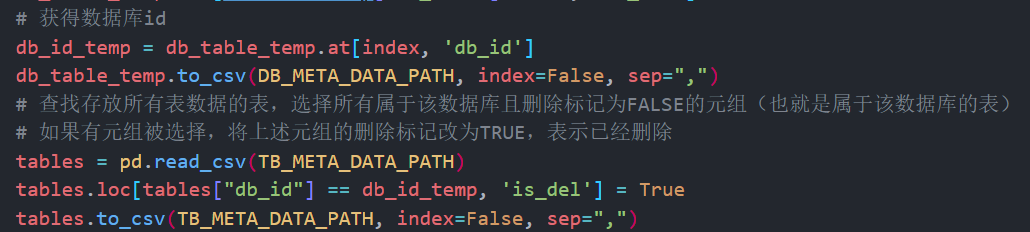
（2）从数据库表文件中查看是否有这个名称对应的并且其删除标记为FALSE的元组

* 1. 如果没有，要删除的数据库不存在
  2. 如果存在，将该元组的删除标记改为TRUE



（3）查找存放所有表数据的表，选择所有属于该数据库且删除标记为FALSE的元组（也就是属于该数据库的表）

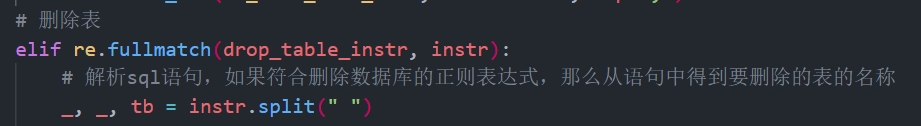
* 1. 如果有元组被选择，将上述元组的删除标记改为TRUE，表示已经删除
  2. 如果没有元组被选择，提示要删除的表不存在



**2.3.2 删除数据表**

删除数据表的基本思路如下：

（1）解析sql语句，如果符合删除数据库的正则表达式，那么从语句中得到要删除的表的名称



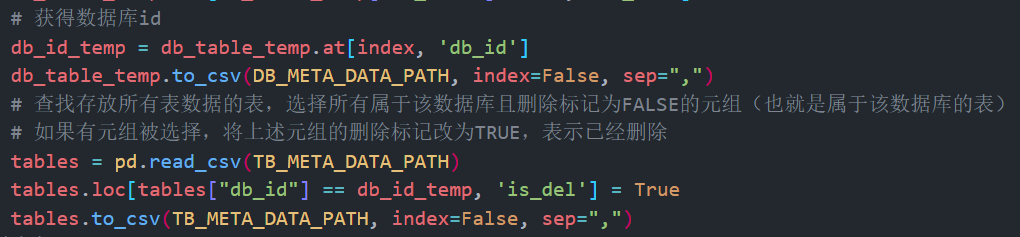
（2）从全局变量获得现在选择的数据库名称

* 1. 如果数据库表文件中有这个名称对应的并且其删除标记为FALSE的元组，继续
  2. 否则，提示选择数据库



（3）查找存放所有表数据的表，选择所有属于该数据库且表名为需要删除的表名，删除标记为FALSE的元组

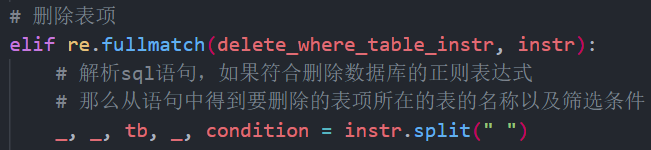
* 1. 如果有元组被选择，将上述元组的删除标记改为TRUE，表示已经删除
  2. 如果没有元组被选择，提示要删除的表不存在



**2.3.3 删除记录**

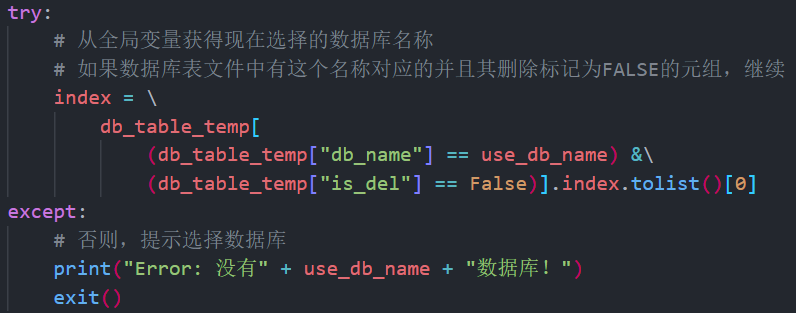
删除记录的基本思路如下：

（1）解析sql语句，如果符合删除数据库的正则表达式，那么从语句中得到要删除的表项所在的表的名称以及筛选条件



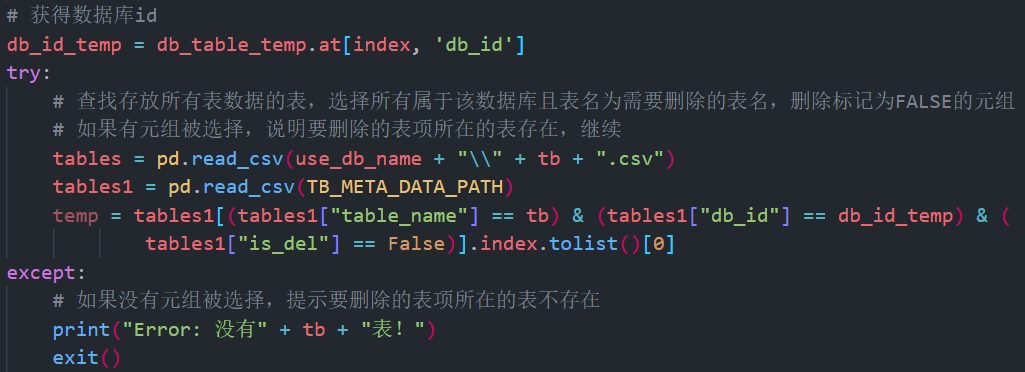
（2）从全局变量获得现在选择的数据库名称

* 1. 如果数据库表文件中有这个名称对应的并且其删除标记为FALSE的元组，继续
  2. 否则，提示选择数据库



（3）查找存放所有表数据的表，选择所有属于该数据库且表名为需要删除的表名

* 1. 如果有元组被选择，说明要删除的表项所在的表存在，继续
  2. 如果没有元组被选择，提示要删除的表项所在的表不存在



（4）读取要删除的表项所在的表，然后删除所有满足筛选条件的行



**2.4 修改模块**

（1）先对update的sql语句进行正则匹配，判断语句是否是update语句：

#update语句

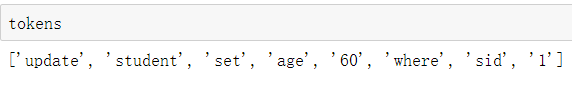
update\_instr="update\\b\\s[a-zA-Z0-9\_]+\\b\\sset\\b\\s[a-zA-Z0-9\_]+[>=<][a-zA-Z0-9\_]+\\b\\swhere\\b\\s[a-zA-Z0-9\_]+[>=<][a-zA-Z0-9\_]+"

1. 成功匹配后对语句进行解析

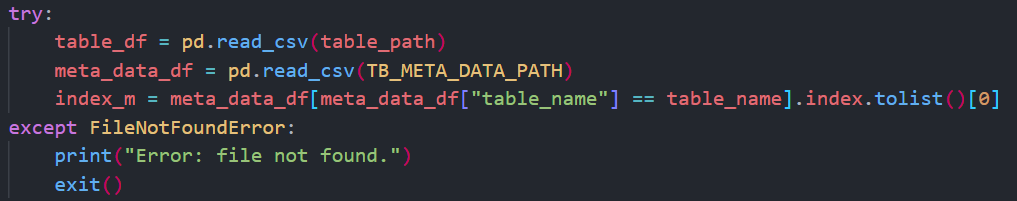
elif re.fullmatch(update\_instr, instr):

tokens=[i for i in re.split(r"([ ,();=])", instr.lower().strip()) if i not in[' ',',',')','(','','=']]

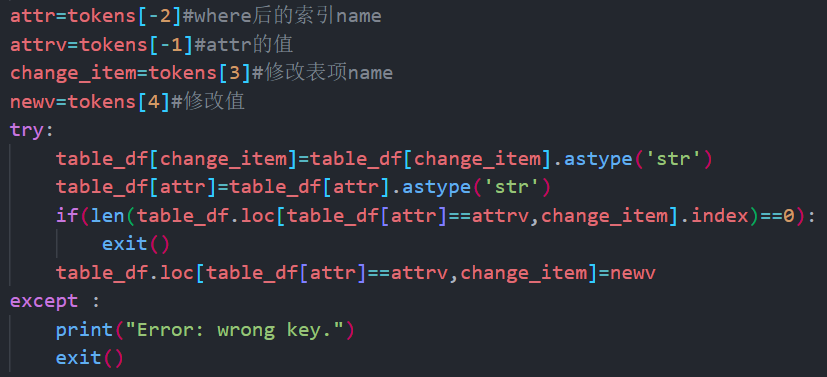
举例结果如下，此时获得表名和待插入的数据：



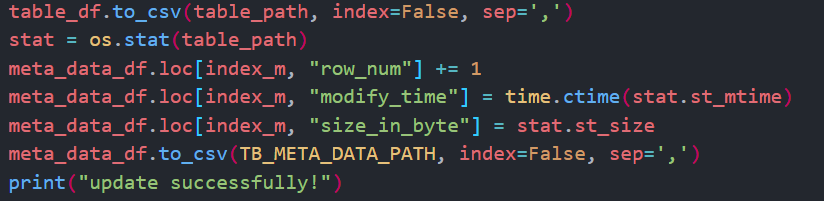
（3）读取csv数据文件



（4）查看键值是否合法，并修改矩阵对应的值



（5）保存数据并更新修改时间、数据大小等信息



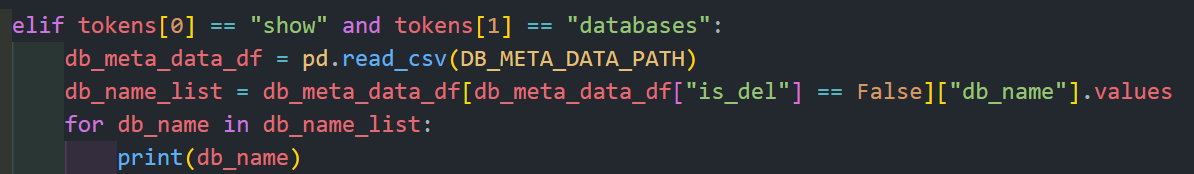
**2.5 查询模块**

**2.5.1 查看数据库**

当用户使用show databases语句时，分析词法单元的第一个与第二个是否为show和databases，是则执行下面的操作。



读取database\_meta\_data.csv得到元数据库数据，查找未被标记删除的记录的db\_name字段并打印显示。

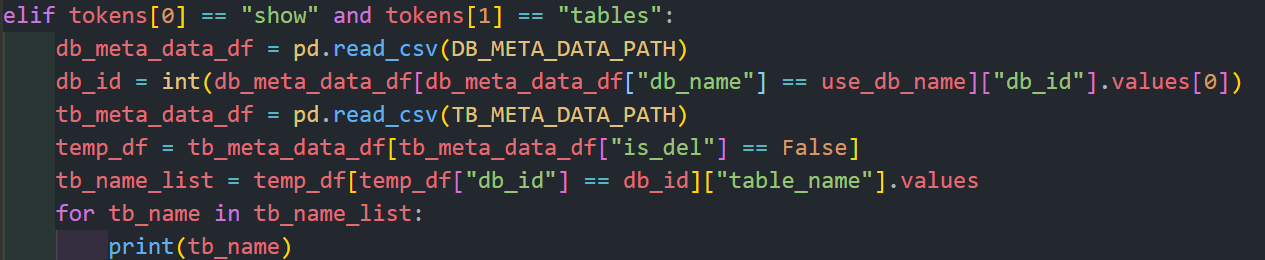


**2.5.2 查看数据表**

当用户使用show tables语句时，分析词法单元的第一个与第二个是否为show和tables，是则执行下面的操作。

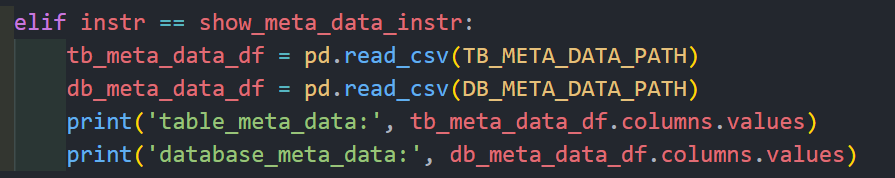


读取database\_meta\_data.csv得到元数据库数据，根据当前选择的数据库名称字段得到对应的数据库id。读取table\_meta\_data.csv得到元数据表数据，查找未被标记删除且db\_id字段为所属数据库id记录的db\_name字段并打印显示。



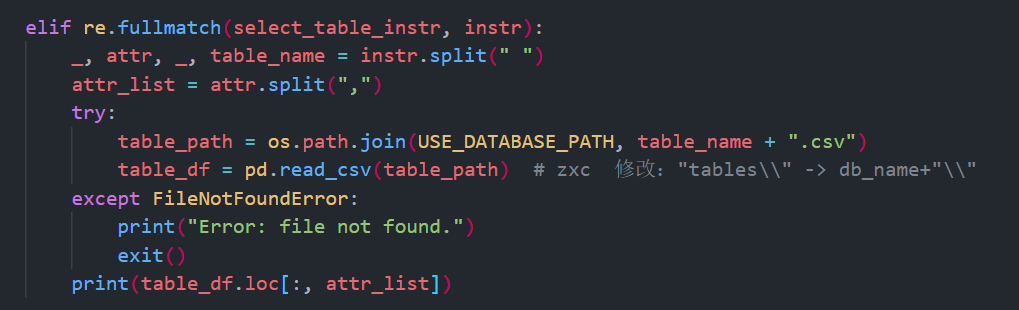
**2.5.3 查看元数据**

当用户使用show meta data语句时，分析是否是预定义的show\_meta\_data\_instr 指令，是则读取database\_meta\_data.csv得到元数据库数据，读取table\_meta\_data.csv得到元数据表数据，分别进行打印显示。

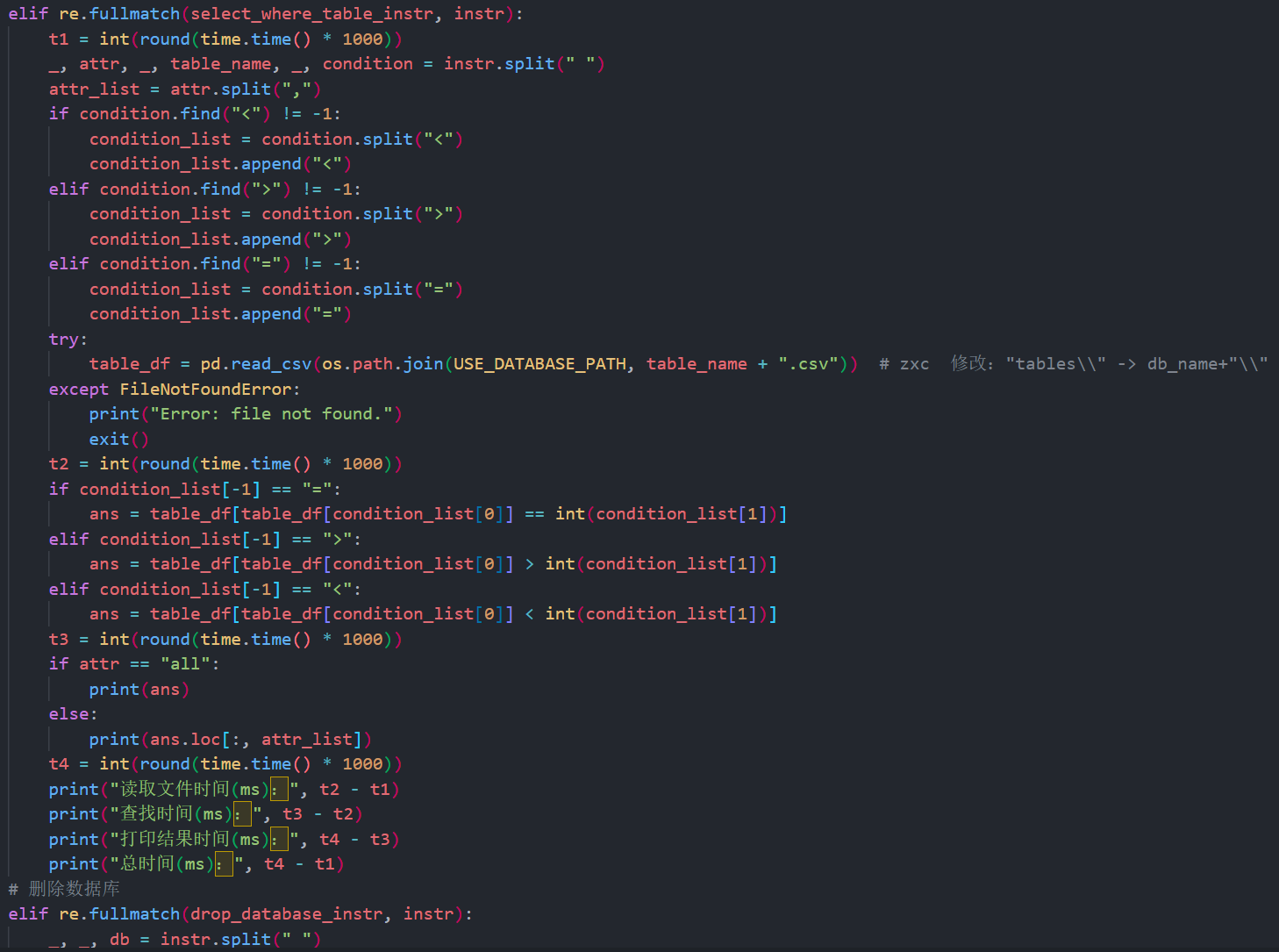


**2.5.4 查询记录**

当用户使用select col\_1, ... , col\_n from tb\_name 语句时，程序先解析指令得到表名和列名，然后显示相应的属性，分别进行打印显示。程序片段如下：



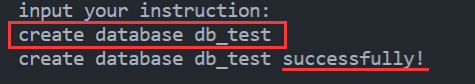
当用户使用select [cols] from [tb\_name] where [conditions] 语句时，程序先解析指令得到表名和列名和条件condition(>、<、=)，显示相应的属性，然后会根据conditions分别进行打印显示符合的数据样本，最后再输出查询时间。程序片段如下：



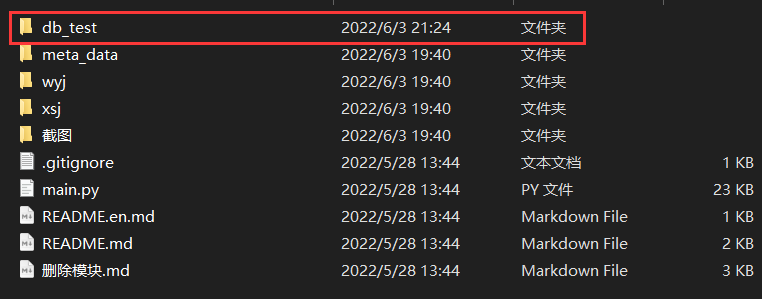
1. **功能测试**

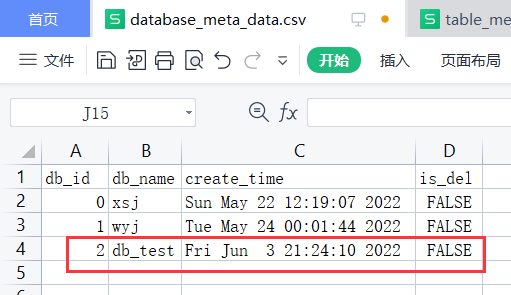
* **创建数据库**：

通过create database db\_test语句创建名为db\_test的数据库；



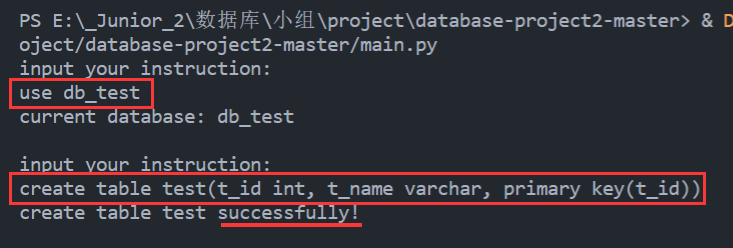
创建后，会在根目录下生成名为db\_test的文件夹，在database\_meta\_data中也可看到该数据库的相关信息。



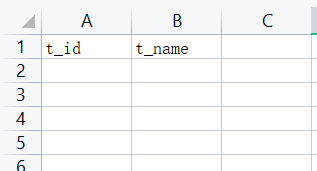
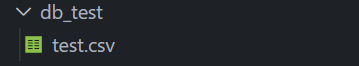


* **创建数据表**：

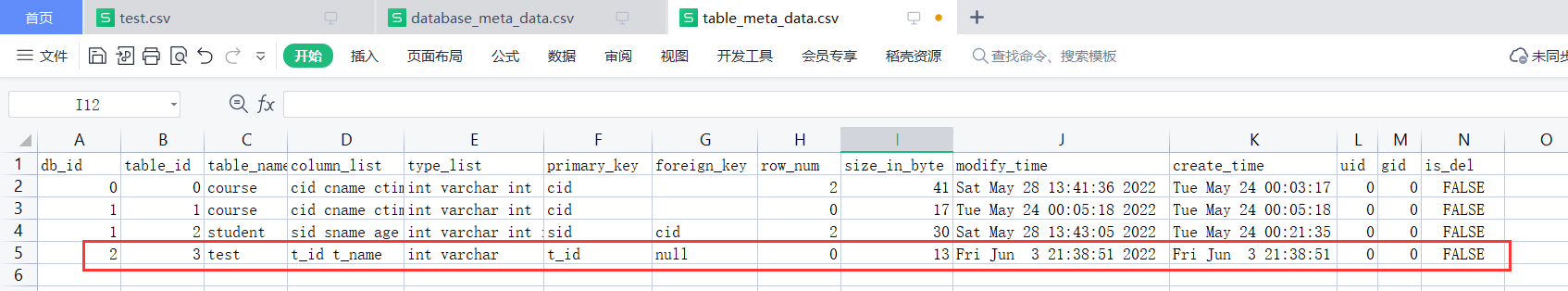
通过use db\_test语句选择名为db\_name的数据库，随后使用create table test(t\_id int, t\_name varchar, primary key(t\_id))在db\_test数据库下创建表；



创建后，db\_test文件夹下会新建一个表，表头为test表的属性：

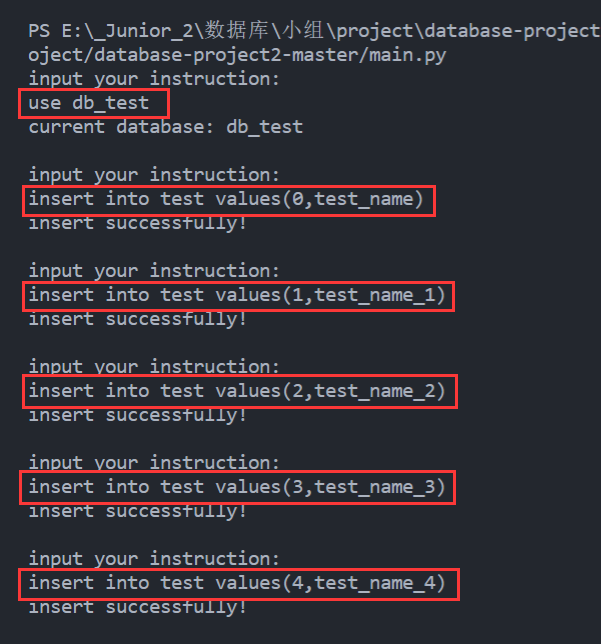


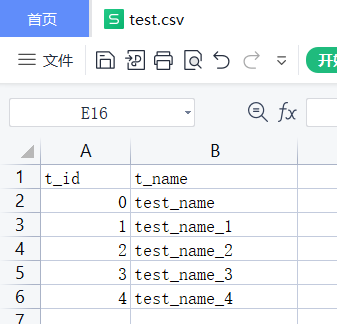
Table\_meta\_data表中会新增下面一项记录表的信息：



* **添加记录：**

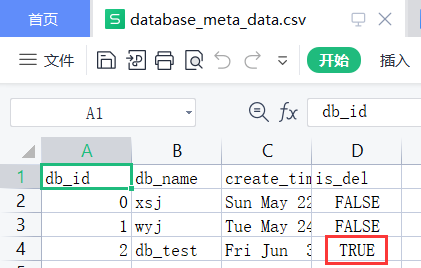
通过use db\_test选定数据库为db\_test，然后使用insert into test values(1,test\_name)语句db\_test的表test中添加记录：





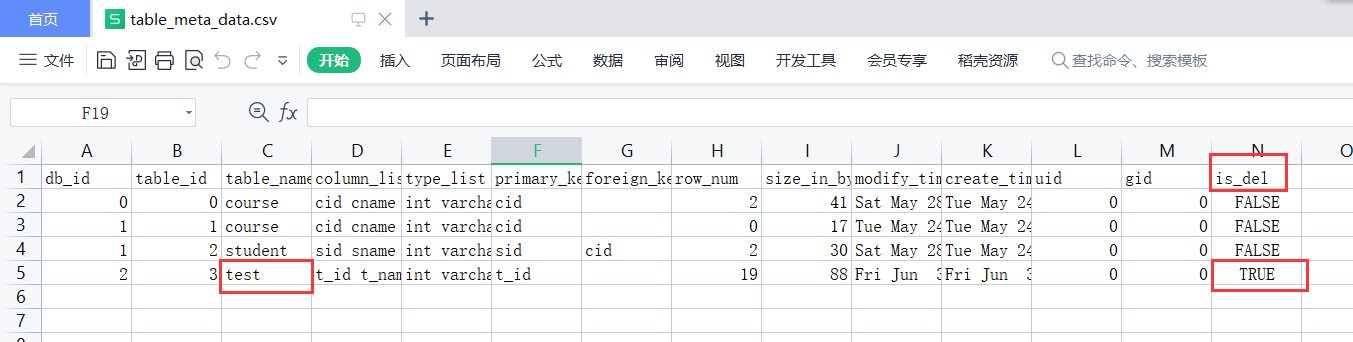
* **删除数据库：**

通过drop database db\_test删除名为 db\_test的数据库；本操作为软删除，在database\_meta\_data中将is\_del标记为TRUE。



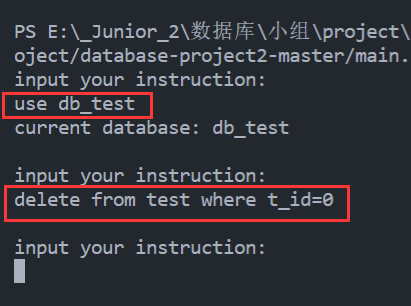
* **删除数据表：**

通过drop table test删除名为test的数据表；本操作为软删除，在table\_meta\_data中将is\_del标记为TRUE。

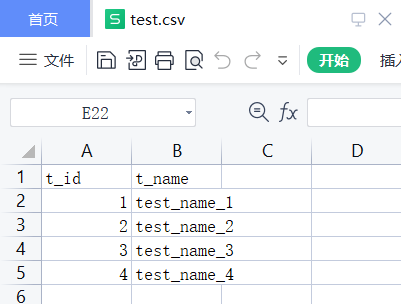
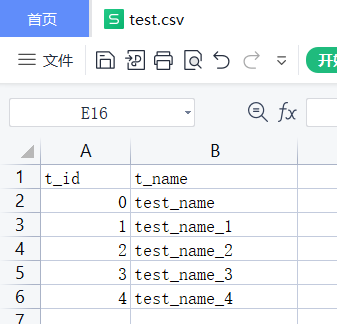


* **删除记录：**

通过delete语句在数据表test中删除t\_id=0的记录；

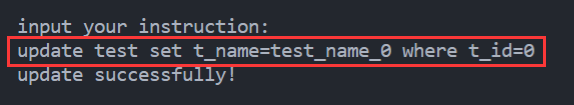


删除前后的截图如下所示：

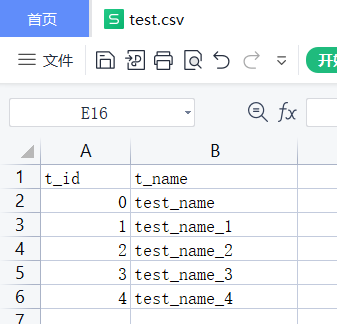
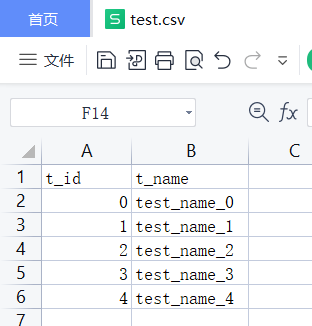


* **修改记录：**

通过use db\_test选定数据库为db\_test，然后使用update语句更新数据表中指定条件的记录：

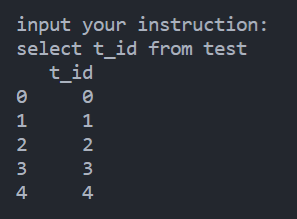
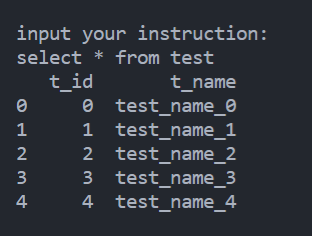


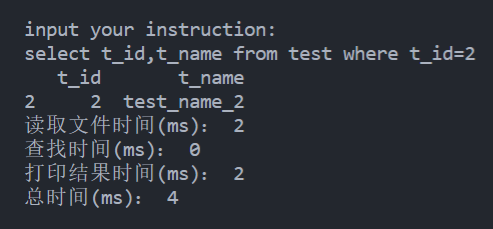
将t\_id为0的记录，t\_name改为test\_name\_0，左图为改之前，有图为改之后：

* **查询记录：**

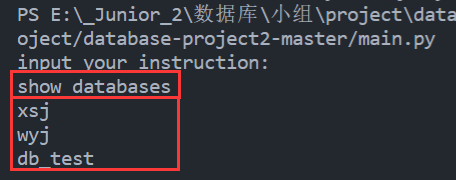
通过select语句查询数据表中的全部记录、某个属性的记录、按条件查询记录；





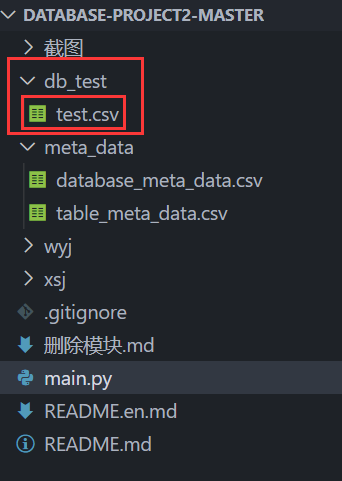
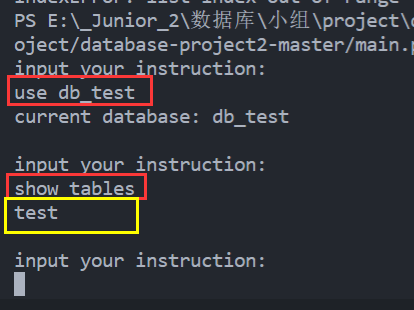
* **查看数据库：**

通过show databases语句查看现有的数据库；



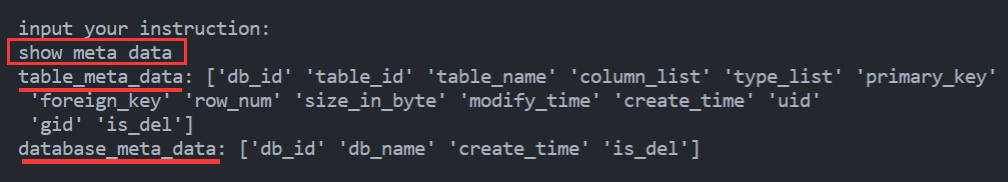
* **查看数据表：**

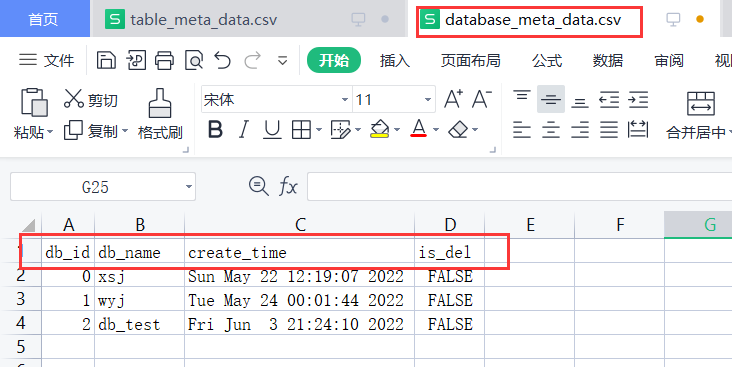
通过show tables语句查看db\_test数据库下面现有的数据表；

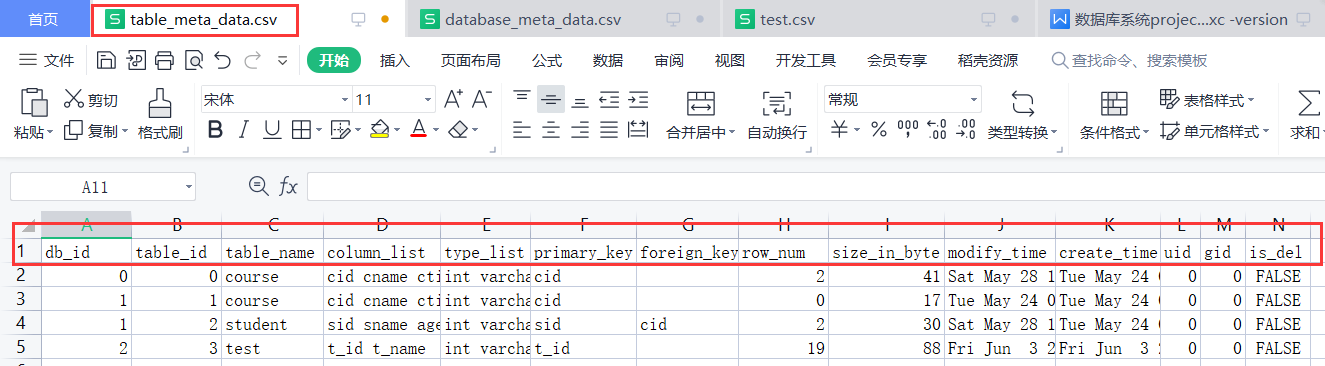


* **查看元数据：**

通过show meta data语句查看数据存储管理器中的元数据；







1. **总结**

本次数据库存储管理器的设计与实现，我们小组最初不理解什么是数据存储管理器，为什么需要这样一个管理器，通过分头查阅文献，多次的小组讨论，对该数据存储管理器进行需求分析，用例建模，渐渐的理解了它的功能，以及为什么需要这样一个数据存储管理器。该数据存储管理器相当于在管理描述关系型数据库的元数据，我们设计了数据库元数据表与数据表元数据表来分别管理数据库和数据表，通过对这两张表中元数据的操作来实现数据库创建、数据表创建、数据添加、删除等操作。

在设计与实现的过程中，我们也遇到了不少问题，包括选择什么样的方式来具体实现数据存储管理器，设计什么样的元数据能够在满足要求的前提下尽可能提高效率，在具体的代码实现中对于一条sql语句是如何解析的，也包括git的使用等。对于这些问题，我们都是先记录下来，然后分几次讨论会集中解决，让每次讨论会开的有效率有价值。通过这样一个项目，一方面我们使用git从零到一进行了一次团队协作的项目开发，提高了团队合作与沟通能力；另一方面我们相当于是站在一个数据库管理系统的设计的角度去思考，去代码实现，让我们从更底层的原理上理解数据库，这也许才是我们真正计算机科学与技术人才做的工作。