防疫系统——一个数据库应用

陈威宇 2000013052

潘骏跃 2000013056

(北京大学 信息科学技术学院,北京 100871);

E-mail: [2000013052@stu.pku.edu.cn](mailto:2000013052@stu.pku.edu.cn) [2000013056@stu.pku.edu.cn](mailto:2000013056@stu.pku.edu.cn)

摘要: 我们选择开发防疫系统，因为它在应对疫情方面起着重要作用。我们使用Python的Flask框架进行Web开发，并使用MySQL Connector/Python处理数据库操作。系统实现了地点管理、人员管理、记录管理和查询功能。用户可以在Update页面进行数据库修改操作，而在Query页面进行信息查询。通过这个项目，我们提升了数据库应用开发的技能，并注重安全性和用户体验。

# 引言

我们选择的题目是做一个防疫系统。为什么选择这个题目呢，因为一个稳定、高效的防疫系统可以对抗疫起到至关重要的作用。具体来讲，一个好的防疫系统可以监测疫情动态、跟踪患者和接触者、识别风险区域，从而及时采取必要的措施来减少病毒或病原体的传播，降低疫情爆发的风险，可以提供即时的信息和数据分析，使相关部门能够迅速作出反应和采取措施。它可以协助卫生部门进行病例追踪、隔离患者、控制传播源，并在需要时调动人力、物资和医疗资源，以控制和遏制疫情的蔓延。总体而言，建立一个防疫系统是为了加强疾病监测、预警和应对能力，以保护公众健康和社会稳定。

通过完成这个项目，我们获得了很多收获。我们对一个数据库应用的开发流程有了详细的了解，并且我们的代码能力得到了提升。我们认识到了开发过程中的一些看似细节但却重要的问题，比如对用户的输入内容的合法性的判断来防止对系统的攻击。此外，我们对ER图的理解很深了，对SQL的使用更熟练了。以及，我们明白了一个应用的开发应当从用户的角度去考虑设计，尽量让没有任何计算机背景知识的人在应用时不太会遇到困难，而不是从开发者的角度去考虑。此外我们也锻炼了一些辅助开发工具的使用，如git。

本文的组织：在第二部分中，我们介绍了本课程设计需要的背景知识，包括我们使用的一个python写的web框架flask(*Welcome to Flask — Flask Documentation (2.3.x)*)以及一些HTML的基本知识，还有python代码中操作数据库用到的 MySQL Connector/Python (*MySQL :: MySQL Connector/Python Developer Guide*)；第三部分描述了课程设计的系统框架；第四部分给出相关的关系数据库模式和相应的操作；第五部分总结了全文，并且描述了完成本课程设计的收获。

# 相关工作

简要介绍一下完成这个项目的相关技术。

首先是如何用python做web开发。我们用了flask这个python的web框架。Flask非常好用，只要对每个网页页面都写一个对应的函数就行了，函数返回的是网页显示的内容（HTML格式）。为了让用户方便输入，我们对每种用户的操作都返回一个画了几个框让用户填入参数的网页（即，用到了HTML的form），用户填完后点submit，网页就会跳转到原有网址后面再加上一串形如?A=xxx?B=yyy的网址（xxx是用户在A这个框填入的内容，yyy是用户在B这个框填入的内容）。我们的代码调用flask的request.args.get就能得到用户填的内容。

然后是如何对用户输入的内容去进行对应的操作，这部分就相当于是把用户给我们的命令转换成为SQL去对数据库做增删改查。这使用MySQL Connector/Python就可以非常方便地在代码里解决。调用cursor.execute直接传入想要执行的SQL字符串就行，如果是查询操作的话对cursor做for循环就得到查询结果的每一行作为一个python的tuple。

具体举个例子，增加一个核酸检测点的代码如下。代码中的test\_site\_name就是用户在框里输入的核酸检测点的名称，render\_template是用来返回一个画了几个框让用户填入参数的网页，form.html是个html网页模版。

@app.route('/update2') #add a new test site

def update2():

test\_site\_name = request.args.get('test\_site\_name', '')

if not test\_site\_name:

return render\_template('form.html', t1 = 'test\_site\_name')

insertion = ('''

INSERT INTO `testing sites`(test\_site\_number, test\_site\_name) VALUES

(0, %s);

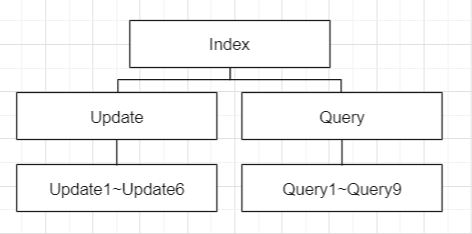
''')

cursor.execute(insertion, (test\_site\_name,))

cnx.commit()

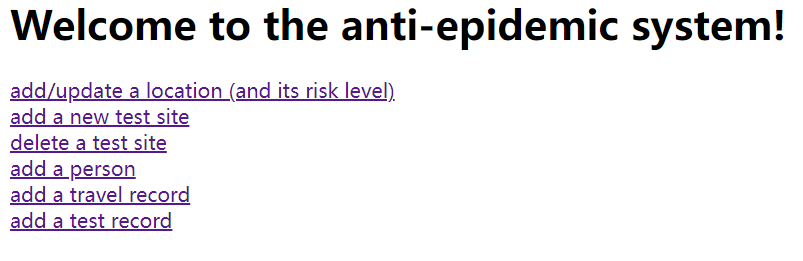
return 'insert into testing sites (?, %s) successfully' % (test\_site\_name,)

# 课程设计的系统框架



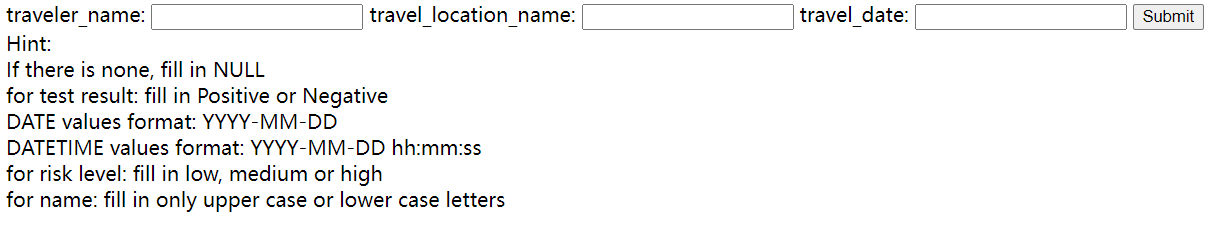
上图展示了该课程设计的一个大体框架。

具体来说，该系统进入的初始页面对应上图中的Index，在该界面中存在两个选项Update和Query，分别代表用户需要对已有数据进行更改和根据已有数据查询所需要的信息。例如下图便展示了在进入Update页面后该课程设计的显示。



在该页面中存在六个选项，是该课程设计中允许的对数据库进行更改的六种操作。六种操作分别对应Update1至Update6，为添加/更新地点及其风险级别、添加一个新的测试地点、删除一个测试地点、添加一个人员、添加一条出行记录、添加一条测试记录。

在选中某一种操作后就可以输入该操作所需的参数，从而对数据库进行修改，如下图所示，对应的是“添加一条出行记录”。



Query页面与Update页面类似，其中存在九个操作选项，选定操作选项并输入所需参数后就会展现出用户所需查询的信息（或是用户输入的参数有误）。

这些可选的操作选项均是我们针对用户的操作需求所设计的。我们的防疫系统实现了的功能如下：

1. 添加/更新地点及其风险级别
2. 添加一个新的测试地点
3. 删除一个测试地点
4. 添加一个人员
5. 添加一条出行记录
6. 添加一条测试记录
7. 查询地点的风险级别
8. 查询一个人在日期A和日期B之间去过的地点
9. 查询所有去过特定地点的人员和日期
10. 查询一个人的所有测试记录
11. 查询一个人最近的阳性记录
12. 查询某个测试地点的所有阳性记录
13. 查询某个时间之后的所有阳性记录
14. 查询在某一天之后去过危险地点但未在某段时间内进行过测试的所有人员
15. 查询某个辅导员所属的前十个在某一天进行了测试的学生

其中，前六个选项为Update所对应的选项，后九个选项为Query所对应的选项。

针对Update，我们希望用户能做到的事是：能对每个城市进行风险等级的更改，在每次存在出行记录、核酸检测记录后能被逐条加入。

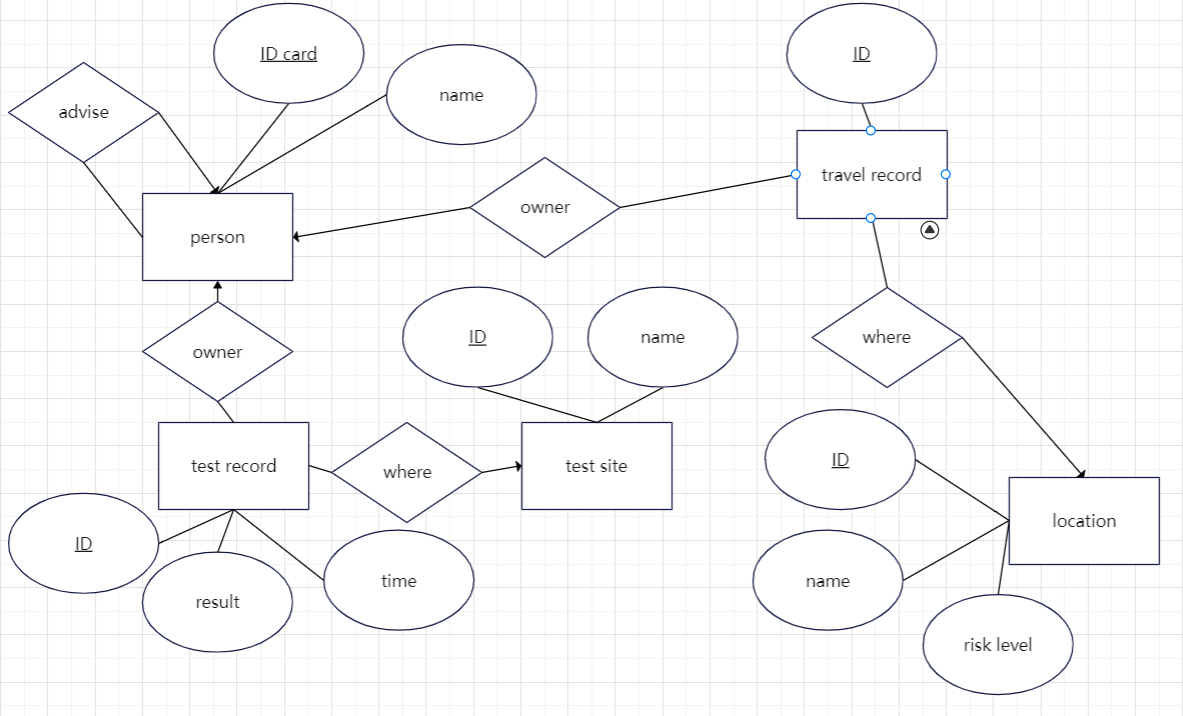
针对Query，我们希望用户能做到的事是：针对城市，如果某个城市变为高风险，那么能查询出前去旅行的所有人员；针对个人，能够查询他所有的出行记录、测试记录以及其是否阳过、最近一次阳的时间；针对总体，能查询出所有存在较大风险的人员，在我们的想法中，即在某天之后去过风险等级较高地区但自从某个时段起从未进行过核酸检测的人员；针对某些特殊需求，能查询某个辅导员管理的最积极完成核酸检测的人员。

综合考虑这些需求，我们设计了如上15个用户所能使用的操作选项。这些选项在sql中的具体实现将会在4.3中进行描述。

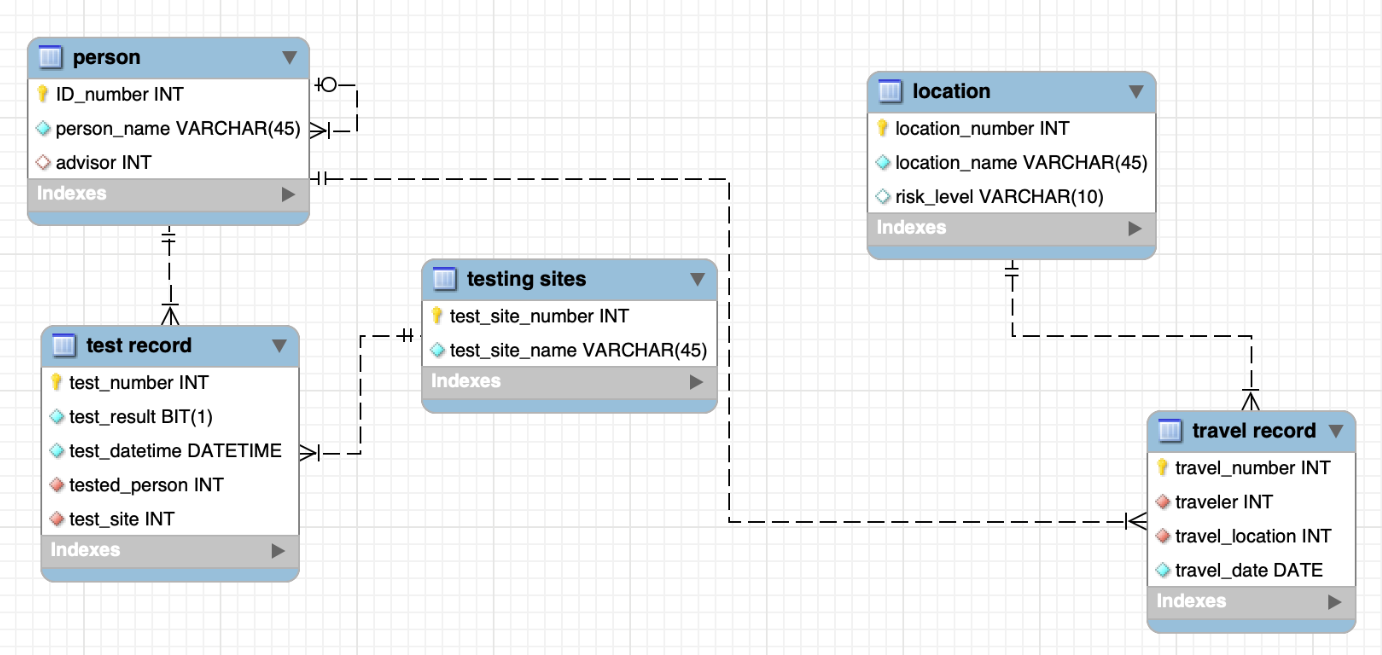
最后，我们还希望该系统能防范一定程度上的注入攻击。

# 课程设计对应的关系数据库模式

## ER图

 上图即为该课程设计对应的概念模式设计图。可以看到该图中存在五个实体，分别代表人员、测试记录、测试地点、出行记录与出行地点。可以看到，每个测试记录与测试地点、人员相关联，且内部还有测试结果、测试时间等属性。出行记录类似，与出行地点、人员相关联，且内部还有出行日期的属性。人员与人员之间也存在“辅导员-学生”的联系。每个人员、出行地点、测试地点也有其独有的编号以及名字属性。每个实体所对应的主码均为其ID。

## 关系数据库模式描述

 上图即为该课程设计对应的物理模式设计图。

对于人员，其主键为ID\_number，即其个人拥有一个特定的数字ID。该ID是需要在添加人员时给出的，类似于身份证号。在尝试添加人员时，我们会对其ID\_number预先判重。

对于测试记录，其主键为test\_number，即每条记录被分配的一个编号。该编号是自动生成的。

对于测试地点、出行地点、出行记录也是类似的，其主键也为自动生成的一个编号。

严格来说，上图给出的设计实际上甚至不满足第一范式。因为第15个操作需要查询某个指定日期内最快完成测试的同学，而我们的表中事实上存储的是datetime，仅就“在某个指定日期”内这个限制似乎只需要考虑它的日期而不需要具体的时刻。

如果抛开这一点来看，该设计满足第三范式的要求，因为其中不存在非主属性对码的部分依赖（主码甚至只有一个），也不存在传递依赖的情况。但它不满足BCNF，因为在location表中，location\_name的值也能完全决定risk\_level的值。

## 关系数据库模式之上的操作

接下来考虑如何实现上述我们提到的希望实现的15种操作，以及如何在一定程度上防范注入攻击。

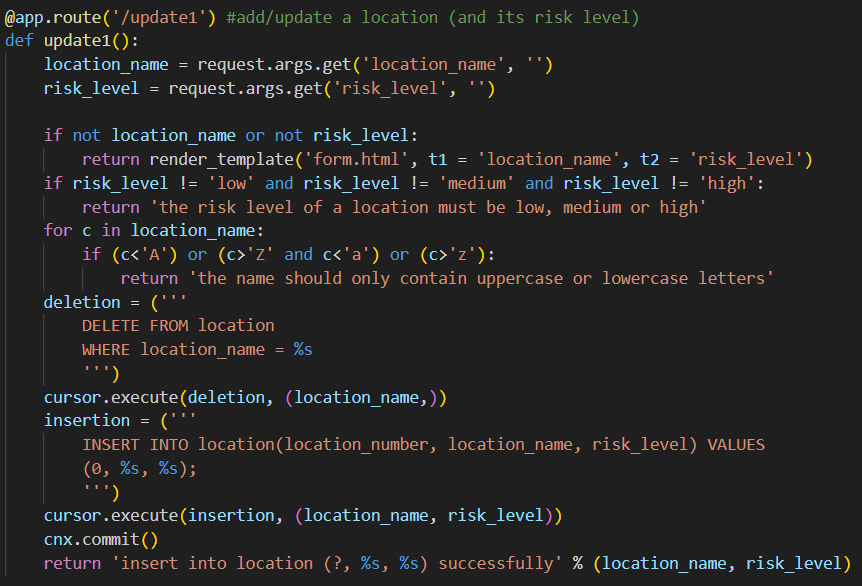
我们先实现了针对人员、测试地点、出行地点中从数字形式的编号到字符串形式的姓名的相互转换，以便后续操作的实现。具体来说，我们会通过在人员、测试地点、出行地点的表中通过SELECT查询编号对应的姓名或是姓名对应的编号，然后将其返回。

在表内的存储中，我们都通过其编号而非姓名代表该人员、测试地点抑或是出行地点。

为了在一定程度上防范注入攻击，我们要求输入的每一个参数符合我们指定的格式。例如风险等级必须是low, medium, high之一，姓名必须仅由大小写字母构成，ID必须是数字等等。我们指定使用编号而非姓名来进行指代也有一定防范攻击的原因在其中：因为姓名毕竟能由所有大小写字母构成，仍然是容易进行攻击的，但如果我们只使用数字来指代该成员，那就能保证姓名只在转换到数字时被sql使用，那么只需保证此处的安全性理论上就能防范通过姓名输入字符串来攻击。

对于前六个Update操作，大体上只需要通过sql的INSERT与DELETE操作实现。值得一提的是，第1种操作同时实现了添加/更新地点及其风险级别，这是通过直接在表中将原地点（不管是否存在）删除再加入新地点来实现的；添加一个人员时，需要额外判断其ID\_number是否出现过；以及添加辅导员和添加一些记录的人员、地点时，都需要判断该人员、地点是否已经被加入过。

以下是一个Update的具体例子。

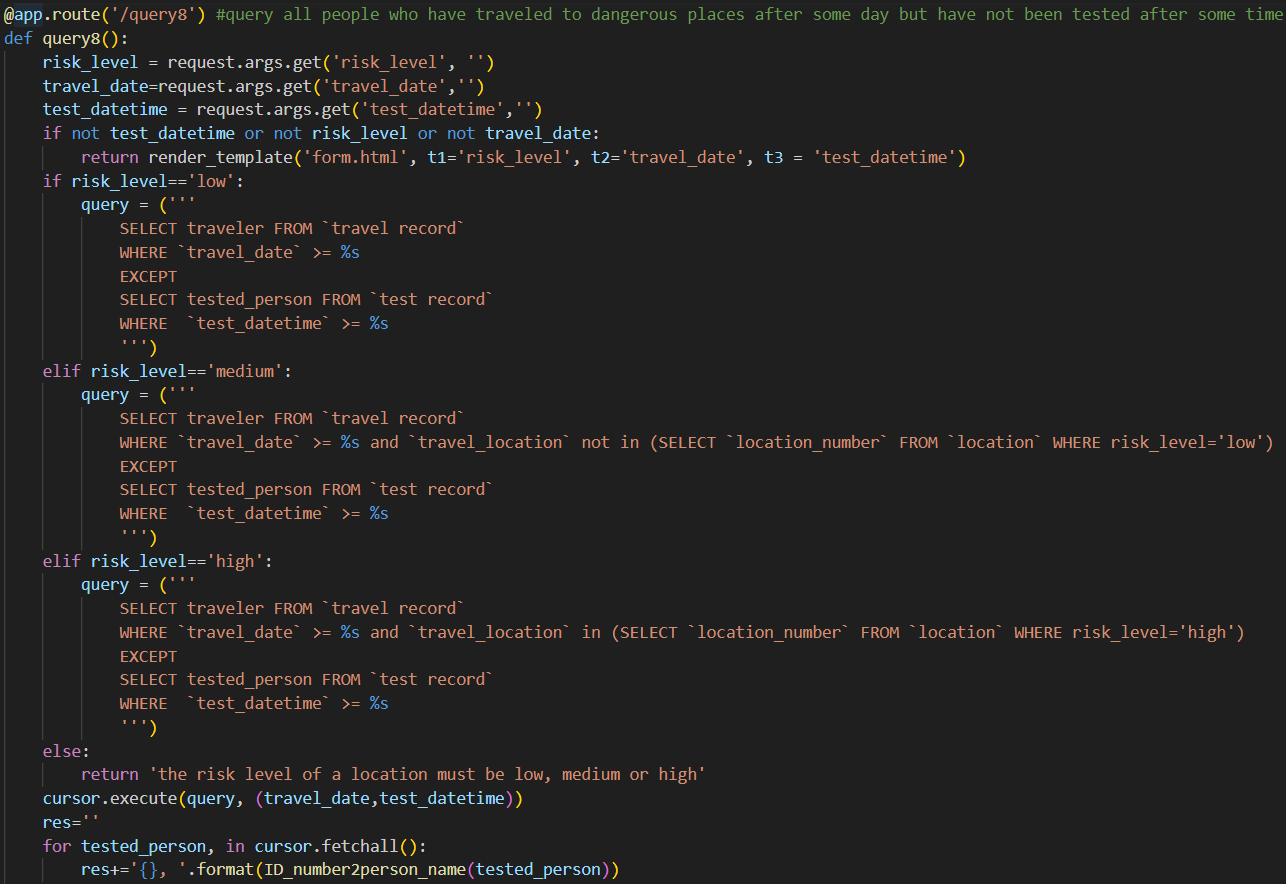


可以看到我们先对risk\_level与location\_name进行了格式要求的判断，然后不管该location\_name是否存在都将其对应的条目删除，最后重新插入新的条目。

而之后的Query操作则相对复杂。具体来说：

1. 查询地点的风险级别：直接在location表中SELECT查询
2. 查询一个人在日期A和日期B之间去过的地点：在travel record中查询人员、日期符合条件的所有记录所对应的地点，加以DISTINCT关键字去重
3. 查询所有去过特定地点的人员和日期：在travel record中查询地点符合要求的记录所对应的人员、日期
4. 查询一个人的所有测试记录：在test record中查询人员符合要求的所有记录
5. 查询一个人最近的阳性记录：在test record中查询人员以及检测结果符合要求的记录，并用类似 >= all的代码通过datetime类型的比较查找到最近的一条
6. 查询某个测试地点的所有阳性记录：在test record中查询地点、检测结果符合要求的所有记录
7. 查询某个时间之后的所有阳性记录：在test record中查询地点、时间符合要求的所有记录
8. 查询在某一天之后去过危险地点但未在某段时间内进行过测试的所有人员：详见下文中给出的例子
9. 查询某个辅导员所属的前十个在某一天进行了测试的学生：在test record中查询日期符合要求的、人员属于“在person表中SELECT出的advisor是该辅导员的学生的集合”的所有记录并ORDER BY具体时间，输出查询出的前十条记录

下图展示的是一个Query操作的例子，是Query8的具体实现（省去了最后具体输出格式的两行）。



该操作中实际上存在三个参数，risk\_level, travel\_date, test\_datetime，分别代表将风险度大于等于risk\_level的地区视作危险地区，要求出行至危险地区日期不早于travel\_date而在test\_datetime之后未在做过测试的所有人员。

简单来说，就是在travel record中选出“在某天后去过危险地方”的人员EXCEPT掉test record中“在某时刻后做过测试”的人员。

事实上，若将risk\_level改写成数字形式则可以进行直接的比较。但在本设计中，risk\_level只有low, medium, high三种且风险等级的比较仅存在于此处，故其简单地使用了字符串类型存储，用if判断每种情况下所需的sql代码。

值得一提的是，在需要扫描多条记录且过程中伴有编号-姓名的转换（因此需要额外的sql操作）时，需要使用fetchall来提前将所有查询到的记录从缓冲区中提取出来。

# 总结

我们选择开发防疫系统这个题目，因为一个稳定、高效的防疫系统在应对疫情方面具有重要作用。它能够监测疫情动态、跟踪患者和接触者、识别风险区域，并采取必要措施来减少病毒传播。通过提供即时信息和数据分析，防疫系统能帮助相关部门迅速作出反应。我们的系统实现了多项功能，包括地点管理、人员管理、记录管理以及查询功能。在这个项目中，我们使用了一些相关技术来完成任务。首先是使用Python进行Web开发。我们选择了Python的Web框架Flask来构建我们的系统。Flask非常灵活易用，我们只需为每个网页页面编写对应的函数，函数返回的内容即为网页的显示内容（以HTML格式）。我们使用了MySQL Connector/Python来方便地在Python代码中处理数据库操作。在系统中，初始页面为Index，用户可以选择进入Update或Query页面。Update页面提供了六个选项，分别对应不同的数据库更改操作，如添加/更新地点及风险级别、添加新的测试地点、删除测试地点、添加人员、添加出行记录和添加测试记录。用户可以选择操作并输入相应参数进行数据库修改。Query页面与Update页面类似，提供了九个选项，用户可以选择查询操作并输入参数来获取所需的信息。通过这个项目，我们提升了数据库应用开发的技能，加深了对安全性和用户体验的重视。

References:

[1]*MySQL :: MySQL Connector/Python Developer Guide*. 取读于 2023年5月27日, 从https://dev.mysql.com/doc/connector-python/en/

[2]*Welcome to Flask—Flask Documentation (2.3.x)*. 取读于 2023年5月27日, 从 https://flask.palletsprojects.com/en/2.3.x/