


# 数据库工程作业

要求:

- 1. 完成一个小型的数据库信息管理系统（或部分功能），并填写工程作业报告；程序和报告请在规定时间之内上传。
- 2. 开发模式（B/S 或 C/S）、开发高级语言任选，后台数据库使用大型数据库管理系统（SQL Server、Oracle、MySQL 等），不要使用桌面数据库。
- 3. 报告中所列举的四种操作，每种操作举一个例子即可。
- 4. 作业成绩按照报告中的标准评分，程序只实现报告中涉及的部分即可。
- 5. 作业完成后，请将工程作业报告和程序打包提交给助教老师，并联系助教老师进行系统说明和演示，回答相关问题。

## 工程作业报告

1. 项目信息（10 分）

学号	2211989	姓名	胡文馨	专业	信息安全
项目名称	智慧医疗管家				
必备环境	Qt5.9, MySQL, Navicat				
系统主要功能简介（4 分）	<p>“智慧医疗管家”已实现的主要功能如下：</p> <ul style="list-style-type: none"><li>1. 实现不同类型用户（病人/医生）的注册，登录；</li><li>2. 若用户类型是病人，则可以选择操作：预约/查看增加看病记录/报销；</li><li>3. 预约界面实现：病人预约的查询、取消、更改、添加；</li><li>4. 查看记录界面实现：查看病人所有的看病记录，包括看病日期、诊断结果、治疗情况以及医生备注，并且可以添加新的看病记录；</li><li>5. 支付界面实现：病人支付情况的查看与更新</li></ul>				
系统主要页面截图（6 分）	<p>截 3-4 个页面即可：</p> <p>登录界面：</p>  <p>查看预约记录界面：</p>				



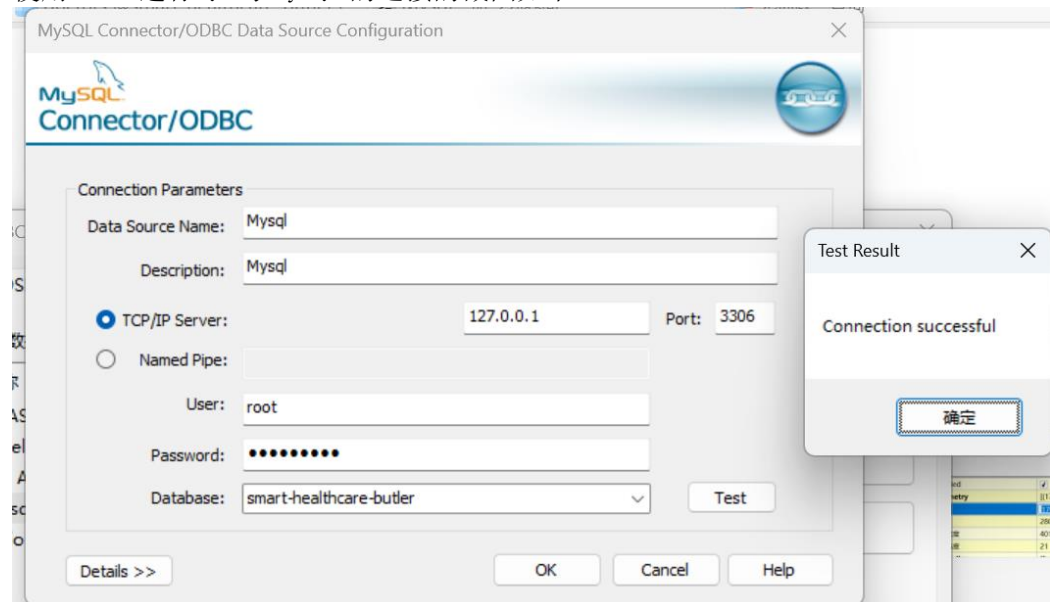
配置 步骤 2 分	DBMS	MySQL			
	高级 语言	C++			
连接串 分析 (6 分)	序号	名称	功能说明	取值	
	1	db.setHost Name	db.setHostName("127.0.0.1"); 设置了数据库服务器的 IP 地址。在这里，它被设置 为本地主机，即数据库服务器位于本地计算机上	"127.0.0.1"	
	2	db.setP ort	db.setPort(3306); 设置了数据库服务器的端口号。默认情况下，MySQL 数据库的端口号是 3306。	3306	
	3	db.setD atabase Name	db.setDatabaseName("Mysql"); 设置了要连接的数据库的名称。在这里，我设置它 为"Mysql"，这是由使用 ODBC 进行与 MySQL 连接时 输入的内容决定的。	"Mysql"	
	4	db.setU serNam e	db.setUserName("root"); 设置了连接数据库所需的用户名。通常情况下，数 据库系统会要求提供有效的用户名以验证连接。在 这里为"root"。	"root"	
	5	db.setP asswor d	db.setPassword("147258369"); 设置了连接数据库所需的密码。密码用于验证用户 身份以访问数据库。我的 MySQL 的密码为 "147258369"。	"147258369"	

连接串代码  
(截屏)  
(2 分)

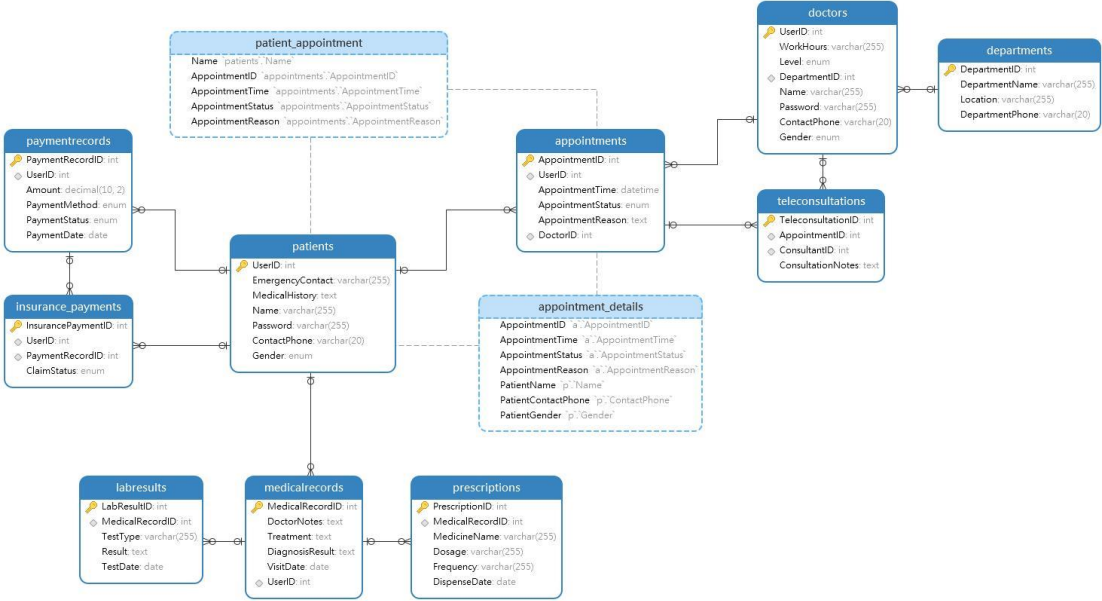
```
1  #include "database.h"
2  #include "mainwindow.h"
3  #include <QApplication>
4  #include <QMainWindow>
5  #include <QSqlDatabase>
6  #include <QSqlQuery>
7  #include <QMessageBox>
8  #include <QDebug>
9  QSqlDatabase Database::db = QSqlDatabase::addDatabase ("QODBC");    // 创建静态数据库实例
10 QSqlQuery* Database::sql = NULL;
11
12 Database::Database()
13 {
14     qDebug()<<"数据库实例已创建";
15 }
16 bool Database::getConnection()
17 {
18     // 数据库连接
19     db.setHostName("127.0.0.1");//本地主机
20     db.setPort(3306);//端口
21     db.setDatabaseName("Mysql");//数据库名
22     db.setUserName("root");// 绑定数据库系统用户名
23     db.setPassword("147258369");//密码
24
25     if(!db.open ())
26     {
27         QMessageBox::information(nullptr, "infor", "Sorry! link failed");
28         qDebug()<<"DataBase Error";
29         qDebug()<<db.lastError().text();
30         return false;
31     }
32     else
33     {
34         QMessageBox::information(nullptr, "infor", "success! ");
35         qDebug()<<"connect succeeded!";
36         return true;
37     }
38 }
39
40 void Database::quitConnection()
41 {
42     db.close();
43 }
44
```

备注

使用 ODBC 进行 Qt 与 MySQL 的连接的截图如下:



### 3. 数据库设计 (14 分)

说明	<p>(10 分) 按照数据表的创建顺序, 依次给出所涉及数据表的信息, 其中参照字段以“(字段 1, 字段 2, ……, 字段 n)”的形式给出, 被参照字段以“表名(字段 1, 字段 2, ……, 字段 n)”的形式给出;</p> <p>(4 分) 一般 DBMS 都可以为数据库生成关系图, 请将该图片截屏并粘贴到表格中。</p>				
数据表 (10)	创建顺序	数据表名称	主键	参照属性	被参照表及属性
	1	departments	DepartmentID		
	2	doctors	UserID	DepartmentID	departments(DepartmentID)
	3	patients	UserID		
	4	medicalrecords	MedicalRecordID	UserID	patients(UserID)
	5	paymentrecords	PaymentRecordID	UserID	patients(UserID)
	6	appointments	AppointmentID	UserID DoctorID	patients(UserID) doctors(UserID)
	7	teleconsultations	Teleconsultations	UserID AppointmentID	doctors(UserID) appointments(AppointmentID)
	8	labresults	LabResultID	MedicalRecordID	medicalrecords(MedicalRecordID)
	9	prescriptions	PrescriptionID	MedicalRecordID	medicalrecords(MedicalRecordID)
	10	insurance_payments	InsurancePaymentID	UserID PaymentRecordID	patients(UserID) paymentrecords(PaymentRecordID)
关系图 (4)	<p>(截屏)</p> <p>在 Navicat 中生成的关系图如下:</p> 				
备注					

#### 4. 含有事务应用的删除操作 (13 分)

说明	<p>(1 分) 简要说明该操作所要完成的功能;</p> <p>(2 分) 该操作会涉及的表 (必须含有两张或两张以上的关系表, 同时以“表名”的形式给出)</p> <p>(1 分) 表连接涉及字段描述 (描述方式为“表 1. 属性=表 2. 属性”)</p>
----	--

	(1 分) 删除条件涉及的字段描述(以“表名. 属性=? ”形式给出) (4 分) 实现该操作的关键代码(高级语言、SQL), 截图即可;(其中如果删除语句中不包含任何形式的事务应用将扣除 3 分) (4 分) 如何执行该操作, 按所述方法能够正常演示程序则给分。																																							
功能描述 (1 分)	病人看病需要预约, 但由于各种原因如突发情况等, 用户不能履约, 这时就需要病人及时取消预约。在预约界面, 可以选择删除预约, 到达删除预约界面, 病人在此界面可以看到本人所有的预约记录, 然后选择填写要删除的预约记录, 点击“一键删除”按钮即可删除该记录, 更新后的病人的记录会同步更新在当前界面。																																							
涉及的表 (2 分)	appointments、patients、doctors																																							
表连接涉及字段 (1 分)	UserID、DoctorID (通过外键连接到了 patients 表和 doctors 表)																																							
	下面为 MySQL 语句的截图: 																																							
	Navicat 中: <table><tr><th>字段</th><th>索引</th><th>外键</th><th>检查</th><th>触发器</th><th>选项</th><th>注释</th><th>SQL 预览</th></tr><tr><td>名</td><td>字段</td><td></td><td></td><td></td><td>被引用的模式</td><td>被引用的表 (父)</td><td>被引用的字段</td><td>删除时</td><td>更新时</td></tr><tr><td>appointments_UserID</td><td></td><td></td><td></td><td></td><td>smart-healthcare-butlpatients</td><td></td><td>UserID</td><td>CASCADE</td><td>CASCADE</td></tr><tr><td>appointments_DoctorID</td><td></td><td></td><td></td><td></td><td>smart-healthcare-butl doctors</td><td></td><td>UserID</td><td>CASCADE</td><td>CASCADE</td></tr></table>		字段	索引	外键	检查	触发器	选项	注释	SQL 预览	名	字段				被引用的模式	被引用的表 (父)	被引用的字段	删除时	更新时	appointments_UserID					smart-healthcare-butlpatients		UserID	CASCADE	CASCADE	appointments_DoctorID					smart-healthcare-butl doctors		UserID	CASCADE	CASCADE
字段	索引	外键	检查	触发器	选项	注释	SQL 预览																																	
名	字段				被引用的模式	被引用的表 (父)	被引用的字段	删除时	更新时																															
appointments_UserID					smart-healthcare-butlpatients		UserID	CASCADE	CASCADE																															
appointments_DoctorID					smart-healthcare-butl doctors		UserID	CASCADE	CASCADE																															
删除条件 字段描述 (1 分)	字段	规则																																						
	AppointmentID	要删除的是 AppointmentID 等于用户输入值的记录, 同时要保证用户输入的值不为空。																																						
代码	(截屏)																																							

(4分)

```
void deleteapp::onDeleteButtonClicked() {
    // 从 LineEdit 读取用户输入的 AppointmentID
    QString appointmentIdStr = ui->appointmentIDLineEdit->text();
    if (appointmentIdStr.isEmpty()) {
        QMessageBox::warning(this, "输入错误", "请输入要删除的预约ID。");
        return;
    }
    int appointmentId = appointmentIdStr.toInt();

    QSqlDatabase db = QSqlDatabase::database();
    if (!db.isOpen()) {
        QMessageBox::critical(this, "错误", "数据库连接失败。");
        return;
    }

    qDebug() << "Starting transaction";
    if (!db.transaction()) {
        qDebug() << "Failed to start transaction:" << db.lastError().text();
        QMessageBox::critical(this, "错误", "无法开始事务: " + db.lastError().text());
        return;
    }

    QSqlQuery query;
    query.prepare("DELETE FROM appointments WHERE AppointmentID = ?");
    query.addBindValue(appointmentId);

    bool ok = query.exec();
    qDebug() << "Executing query: DELETE FROM appointments WHERE appointment_id =" << appointmentId << " Result: " << ok;
    if (ok) {
        if (db.commit()) {
            qDebug() << "Transaction committed successfully";
            model->select(); // 更新视图
            QMessageBox::information(this, "删除成功", "预约记录已删除。");
        } else {
            qDebug() << "Failed to commit transaction:" << db.lastError().text();
            db.rollback();
            QMessageBox::critical(this, "事务提交失败", "请重试或检查数据库状态: " + db.lastError().text());
        }
    } else {
        qDebug() << "Failed to execute query:" << query.lastError().text();
        db.rollback();
        QMessageBox::critical(this, "删除失败", "数据库错误: " + query.lastError().text());
    }
}
```

程序演示 (4分)

病人用户点击我要预约后进入预约界面:



点击删除预约按钮, 跳转到删除预约界面:

此时用户可以输入想要删除的预约记录, 并点击一键删除按钮。



点击一键删除按钮后，就可以正确执行含有事务应用的删除操作，此时会跳出窗口，提示用户：预约记录已删除。



如果未删除成功，就会进行回滚，并且程序报错，弹出窗口提示用户删除失败。

（下图删除失败是因为我把代码

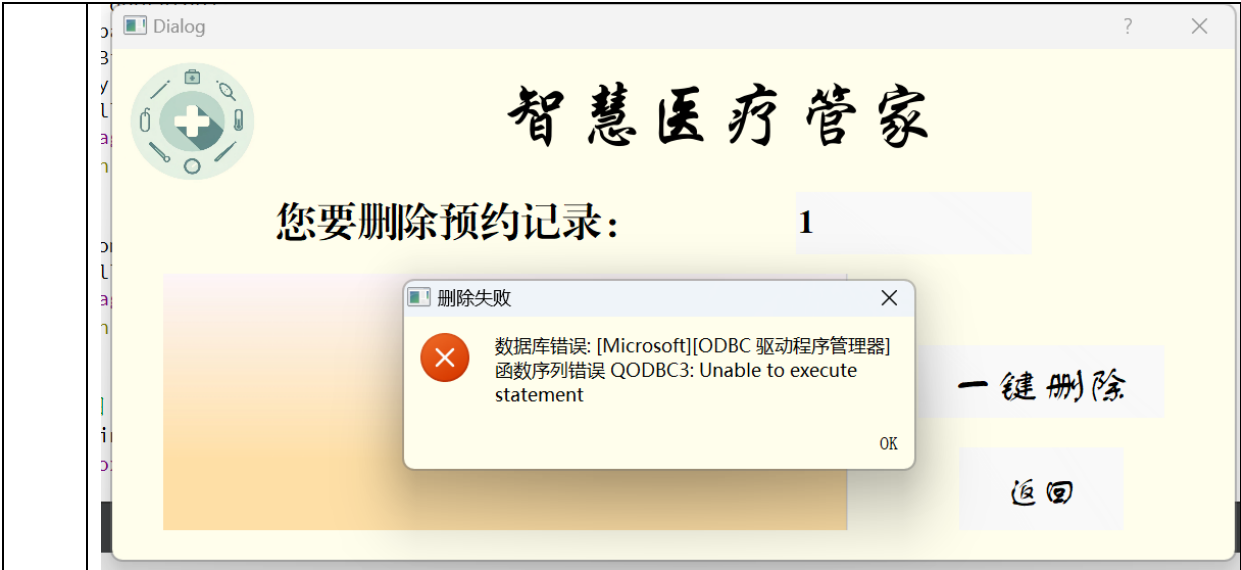
```
query.prepare("DELETE FROM appointments WHERE AppointmentID = ?");
```

写成了

```
query.prepare("DELETE FROM appointments WHERE appointment_id = ?");
```

，就会发生数据库错误，进行了回滚。）





在 Navicat 中也可以看到删除前后的变化:

删除前:

AppointmentID	UserID	AppointmentTime	AppointmentStatus	AppointmentReason	DoctorID
1	2	2024-05-15 10:47:30	scheduled	常规体检	1
2	4	2024-05-16 11:48:20	scheduled	过敏测试	1
3	6	2024-05-17 09:49:49	completed	糖尿病复查	1
4	8	2024-05-18 14:50:23	cancelled	高血压随访	1
5	10	2024-05-19 15:50:52	scheduled	常规体检	3
6	2	2024-05-20 10:30:12	completed	头痛检查	1
7	4	2024-05-21 11:52:42	cancelled	胃痛检查	1
8	6	2024-05-22 09:22:24	cancelled	皮疹复查	9
9	8	2024-05-23 14:53:39	completed	心电图检查	1
10	10	2024-05-24 15:15:03	scheduled	牙痛检查	7

删除后 (正确删除了 AppointmentID 为 1 的记录):

AppointmentID	UserID	AppointmentTime	AppointmentStatus	AppointmentReason	DoctorID
2	4	2024-05-16 11:48:20	scheduled	过敏测试	1
3	6	2024-05-17 09:49:49	completed	糖尿病复查	1
4	8	2024-05-18 14:50:23	cancelled	高血压随访	1
5	10	2024-05-19 15:50:52	scheduled	常规体检	3
6	2	2024-05-20 10:30:12	completed	头痛检查	1
7	4	2024-05-21 11:52:42	cancelled	胃痛检查	1
8	6	2024-05-22 09:22:24	cancelled	皮疹复查	9
9	8	2024-05-23 14:53:39	completed	心电图检查	1
10	10	2024-05-24 15:15:03	scheduled	牙痛检查	7

备注 由于 Navicat 中的 SQL 查询编辑器不支持直接从用户那里动态接收输入，所以需要通过外部程序来管理用户输入，所以我这里直接在 Qt 中实现了有关含有事务应用的删除操作。

## 5. 触发器控制下的添加操作（20 分）

说明	(1 分) 简要说明该操作所要完成的功能; (2 分) 简要说明该触发器所要完成的功能 (1 分) 该操作会涉及的表 (以“表名”的形式给出)。 (2 分) 该操作输入数据以及输入数据应该满足的条件, 如: 数值范围、是否为空; (6 分) 实现该操作的关键代码 (高级语言、SQL), 截图即可; (8 分) 如何执行该操作, 按所述方法能够正常演示程序则给分。	
功能描述 (1 分)	看病记录帮助病人跟踪自己的身体状况, 当产生新的看病记录时, 病人可以手动添加记录。在看病记录界面, 病人可以看到本人所有的预约记录, 并且可以选择“添加新的记录”按钮, 到达增加记录界面, 填写相关内容后, 点击“提交”按钮即可新增记录。	
触发器描述 (2 分)	触发器 1: 在向 medicalrecords 表中插入一条记录后, 通过触发器, 会向 prescriptions 表插入一条自动生成的带有默认值的新记录后。 触发器 2: 在执行插入操作前检查 patients 表中是否存在与新记录的 UserID 相对应的用户。如果 UserID 不存在, 则触发一个错误信号并输出错误消息, 以确保只有存在于 patients 表中的用户才能被引用。 触发器 3: 在执行插入前检查新记录的 VisitDate 字段。第一, 它检查 VisitDate 是否为空, 如果为空则触发错误并返回消息“VisitDate 不能为空”。第二, 它检查 VisitDate 的格式是否符合 YYYY-MM-DD 的标准, 如果格式不正确则触发错误并返回消息“VisitDate 格式无效, 应为 YYYY-MM-DD”, 以确保输入的数据格式正确。	
涉及的表 (1 分)	medicalrecords、prescriptions	
输入数据 (2 分)	字段	规则
	MedicalRecordsID	输入一条看病记录: 记录包含 MedicalRecordID
	DoctorNotes	输入一条看病记录: 记录包含 DoctorNotes
	Treatment	输入一条看病记录: 记录包含 Treatment
	DiagnosisResult	输入一条看病记录: 记录包含 DiagnosisResult
	VisitDate	输入一条看病记录: 记录包含 VisitDate
	UserID	输入一条看病记录: 记录包含 UserID
插入操作源码 (3 分)	(截屏)	

```
void addrec::on_pushButton_clicked()
{
    // 获取用户输入的值
    int userId = ui->userIdLineEdit->text().toInt();
    QString doctorNotes = ui->doctorNotesLineEdit->text();
    QString treatment = ui->treatmentLineEdit->text();
    QString diagnosisResult = ui->diagnosisResultLineEdit->text();
    QString visitDate = ui->visitDateLineEdit->text();

    QSqlQuery query;
    query.prepare("INSERT INTO medicalrecords (DoctorNotes, Treatment, DiagnosisResult, VisitDate, UserID) "
        "VALUES (:doctorNotes, :treatment, :diagnosisResult, :visitDate, :userId)");
    query.bindValue(":doctorNotes", doctorNotes);
    query.bindValue(":treatment", treatment);
    query.bindValue(":diagnosisResult", diagnosisResult);
    query.bindValue(":visitDate", visitDate);
    query.bindValue(":userId", userId);

    if (query.exec()) {
        QMessageBox::information(this, "Success", "Record added successfully");
    } else {
        QMessageBox::critical(this, "Error", query.lastError().text());
    }
}
```

(截屏)

以下图片均为从 Navicat 中截图：

一共有三个触发器：

名	触发	插入	更新	删除
add_prescription_on_medical_record_insert	AFTER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
mmedical_record_insert_integrity_check	BEFORE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
visittime	BEFORE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

触发器 1: add\_prescription\_on\_medical\_record\_insert

定义

```
1 BEGIN
2     DECLARE prescription_id INT;
3
4     -- 插入到处方表中
5     INSERT INTO prescriptions (MedicalRecordID, MedicineName, Dosage, Frequency, DispenseDate)
6     VALUES (NEW.MedicalRecordID, '默认药品', '默认剂量', '默认频率', CURDATE());
7
8     -- 获取自动生成的PrescriptionID
9     SELECT LAST_INSERT_ID() INTO prescription_id;
10
11    -- 使用自动生成的PrescriptionID更新处方记录
12    UPDATE prescriptions SET MedicineName = CONCAT('默认药品_', prescription_id),
13                           Dosage = CONCAT('默认剂量_', prescription_id),
14                           Frequency = CONCAT('默认频率_', prescription_id)
15    WHERE PrescriptionID = prescription_id;
16 END
```

触发器 2: mmedical\_record\_insert\_integrity\_check

定义

```
1 BEGIN
2     DECLARE user_exists INT;
3
4     -- 检查 UserID 是否存在于 patients 表中
5     SELECT COUNT(*) INTO user_exists FROM patients WHERE UserID = NEW.UserID;
6
7     IF user_exists = 0 THEN
8         SIGNAL SQLSTATE '45000'
9         SET MESSAGE_TEXT = '插入的 UserID 在 patients 表中不存在';
10    END IF;
11
12 END
```

触发器源码  
(3分)

触发器 3: visittime

定义

```
1 BEGIN
2     -- 检查 VisitDate 是否为空
3     IF NEW.VisitDate IS NULL THEN
4         SIGNAL SQLSTATE '45000'
5         SET MESSAGE_TEXT = 'VisitDate 不能为空';
6     END IF;
7
8     -- 检查 VisitDate 格式是否正确
9     IF NEW.VisitDate NOT REGEXP '^[0-9]{4}-[0-9]{2}-[0-9]{2}$' THEN
10        SIGNAL SQLSTATE '45000'
11        SET MESSAGE_TEXT = 'VisitDate 格式无效, 应为 YYYY-MM-DD';
12    END IF;
13
14 END
```

说明：不违背触发器能够执行插入操作。  
在查看记录界面点击“添加新的记录”后即可到达新增记录界面：

程序演示  
(4分)

Dialog

智慧医疗管家

用户名

请输入用户名

记录产生日期

请输入记录产生日期, 例: 2024-05-01

医疗记录ID

请输入医疗记录ID

治疗

请输入治疗方式

诊断结果

请输入诊断结果

医生备注

请输入医生备注

提交

取消

填写相关内容:

Dialog

智慧医疗管家

用户名

11

记录产生日期

2024-05-30

医疗记录ID

11

治疗

输葡萄糖

诊断结果

低血糖

医生备注

按时吃饭

提交

取消

点击“提交”后，若输入内容均符号要求，会弹出提示窗口，显示“Record added successfully.”说明提交成功。



PrescriptionID	MedicalRecordID	MedicineName	Dosage	Frequency	DispenseDate
1		1 阿莫西林	500mg	每天三次	2024-05-01
2		2 布洛芬	200mg	每天两次	2024-05-02
3		3 氯雷他定	10mg	每天一次	2024-05-03
4		4 阿司匹林	100mg	每天一次	2024-05-04
5		5 甲硝唑	250mg	每天两次	2024-05-05
6		6 青霉素	1g	每天一次	2024-05-06
7		7 对乙酰氨基酚	500mg	每天一次	2024-05-07
8		8 头孢克洛	500mg	每天两次	2024-05-08
9		9 地塞米松	0.75mg	每天一次	2024-05-09
10		10 利福平	500mg	每天一次	2024-05-10
11		16 默认药品_11	默认剂量_11	默认频率_11	2024-05-30

说明：违背触发器要求，不能够执行插入操作，系统报错。

当输入了不存在的用户名时，触发器 2 发挥作用，程序报错弹出报错窗口。

Dialog

## 智慧医疗管家

用户名 **18**

记录产生日期 2024-01-

医疗记录ID **16**

治疗 输葡萄糖

诊断结果 轻微低血糖

医生备注 及时吃早饭

提交 取消

Error

[MySQL][ODBC 8.0(a) Driver][mysqld-8.0.36]  
错误: 无法执行语句  
QODBC3: Unable to execute statement

程序  
演示  
(4  
分)

当输入的记录产生日期为空时，触发器 3 发挥作用，程序报错弹出报错窗口。

Dialog

## 智慧医疗管家

用户名 **4**

记录产生日期 请输入记录产生日期，例：2024-05-01

医疗记录ID **14**

治疗 输葡萄糖

诊断结果 轻微低血糖


医生备注 及时吃早饭

提交 取消

Error

[MySQL][ODBC 8.0(a) Driver]  
[mysqld-8.0.36]Incorrect date value: '' for  
column 'VisitDate' at row 1 QODBC3: Unable  
to execute statement

当记录产生日期输入不规范时，没有按照特定的格式输入时，触发器 3 发挥作用，程序报错弹出报错窗口。

	<div><div>Dialog</div><div><div></div><div><h2>智慧医疗管家</h2></div></div><div><div>用户名</div><div>6</div></div><div><div>记录产生日期</div><div>1月9号</div></div><div><div>医疗记录ID</div><div>19</div></div><div><div>治疗</div><div>吃甘草片</div></div><div><div>诊断结果</div><div>上火咳嗽</div></div><div><div>医生备注</div><div>不要熬夜</div></div><div><div>提交</div><div>取消</div></div></div>
备注	<p>在 MySQL 中，SQLSTATE '45000' 表示一个通用的用户定义异常。它的作用是让触发器在检测到某种特定条件时主动抛出错误，并返回给调用程序（即我的 Qt 应用程序）</p> <p>程序报错我使用了：SIGNAL SQLSTATE '45000'，它在 SQL 触发器中用来抛出用户定义的错误。</p>



## 6. 存储过程控制下的更新操作（18 分）

说明	<p>（1 分）简要说明该操作所要完成的功能；</p> <p>（1 分）简要说明该存储过程所要完成的功能；</p> <p>（2 分）说明该操作涉及操作的表（必须包含两张或两张以上的关系表，以“表名形式”描述）</p> <p>（1 分）表连接涉及字段描述（描述方式为“表 1. 属性=表 2. 属性”）</p> <p>（2 分）该操作会修改字段（以“表名. 字段名”的形式给出），以及修改规则，如新数值的计算方法、在何种条件下予以修改等；</p> <p>（6 分）实现该操作的关键代码（高级语言、SQL），截图即可；</p> <p>（5 分）如何执行该操作，按所述方法能够正常演示程序则给分。</p>	
功能描述 （1 分）	<p>病人看病花钱需要报销，需要及时更新报销状态以确定看病的某项记录的钱是否报销。在支付界面，病人可以通过查询某条特定的支付记录，看到该记录的完整信息，通过修改 insurance_payments 表中的报销状态，可同步更新 payments 表中的是否结清报销完状态。</p>	
存储过程功能描述 （1 分）	<p>存储过程 UpdatePaymentStatus 用于更新支付记录的状态。</p> <p>它首先会验证传入的新状态（newClaimStatus）是否合法（只能是 pending, approved, rejected 中的一种），如果不合法则触发错误。</p> <p>它会根据病人用户输入的值更新 insurance_payments 表中对应 paymentRecordID 的 ClaimStatus 字段。根据新的 ClaimStatus 值更新 paymentrecords 表中的 PaymentStatus 字段：如果新状态是 approved，则将 PaymentStatus 设置为 paid，否则设置为 unpaid。</p>	
涉及的关系表 （2 分）	paymentrecords、insurance_payments	
表连接涉及字段 （1 分）	PaymentRecordID	
更改字段 （2 分）	字段	规则
	newClaimStatus	更改 insurance_payments 表中对应特定用户输入的 paymentRecordID 的 ClaimStatus 字段，若 ClaimStatus 为 approved，则将 PaymentStatus 设置为 paid；若 ClaimStatus 为 pending/rejected，则将 PaymentStatus 设置为 unpaid。
	PaymentStatus	同上，修改 newClaimStatus 的同时也会修改 PaymentStatus。
更新代码 （3 分）	<p>（截屏）</p> <p>更新操作代码：</p>	



	<pre>1 CREATE DEFINER='root'@'localhost' PROCEDURE `UpdatePaymentStatus` (IN paymentRecordID INT, IN newClaimStatus ENUM('pending','approved','rejected')) 2 BEGIN 3     -- 验证新状态是否合法 4     IF newClaimStatus NOT IN ('pending', 'approved', 'rejected') THEN 5         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid claim status'; 6     END IF; 7 8     -- 更新 insurance_payments 表中的 ClaimStatus 9     UPDATE insurance_payments 10    SET ClaimStatus = newClaimStatus 11    WHERE PaymentRecordID = paymentRecordID; 12 13    -- 根据 ClaimStatus 更新 paymentrecords 表中的 PaymentStatus 14    IF newClaimStatus = 'approved' THEN 15        UPDATE paymentrecords 16        SET PaymentStatus = 'paid' 17        WHERE PaymentRecordID = paymentRecordID; 18    ELSE 19        UPDATE paymentrecords 20        SET PaymentStatus = 'unpaid' 21        WHERE PaymentRecordID = paymentRecordID; 22    END IF; 23 END</pre> <p>显示更新后数据代码:</p> <pre>void pay::updateClaimStatus() {     qDebug() &lt;&lt; "Update Status Button Clicked"; // 调试输出     QString paymentRecordId = ui-&gt;paymentRecordIdEdit-&gt;text();     QString newClaimStatus = ui-&gt;statusEdit-&gt;text(); // 获取用户填写的新状态      // 调用存储过程     QSqlQuery query;     query.prepare("CALL UpdatePaymentStatus(:paymentRecordId, :newClaimStatus)");     query.bindValue(":paymentRecordId", paymentRecordId.toInt());     query.bindValue(":newClaimStatus", newClaimStatus);      if (!query.exec()) {         QMessageBox::critical(this, "Stored Procedure Execution Failed", "Stored procedure execution failed: " + query.lastError().text());         return;     }      // 刷新表格，显示更新后的数据     QSqlQuery refreshQuery;     refreshQuery.prepare("SELECT * FROM paymentrecords WHERE PaymentRecordID = :paymentRecordId");     refreshQuery.bindValue(":paymentRecordId", paymentRecordId);      if (!refreshQuery.exec()) {         QMessageBox::critical(this, "Query Failed", "Query failed: " + refreshQuery.lastError().text());         return;     }      QSqlQueryModel *model = new QSqlQueryModel;     model-&gt;setQuery(refreshQuery);     ui-&gt;tableView-&gt;setModel(model);      model-&gt;setHeaderData(0, Qt::Horizontal, tr("支付序号"));     model-&gt;setHeaderData(1, Qt::Horizontal, tr("用户名"));     model-&gt;setHeaderData(2, Qt::Horizontal, tr("支付数额"));     model-&gt;setHeaderData(3, Qt::Horizontal, tr("支付方式"));     model-&gt;setHeaderData(4, Qt::Horizontal, tr("支付状态"));     model-&gt;setHeaderData(5, Qt::Horizontal, tr("支付记录产生日期")); }</pre>
创建 存储 过程 源码 (3 分)	<p>(截屏)</p> <pre>定义 SQL 预览 1 CREATE DEFINER='root'@'localhost' PROCEDURE `UpdatePaymentStatus` (IN paymentRecordID INT, IN newClaimStatus ENUM('pending','approved','rejected')) 2 BEGIN 3     -- 验证新状态是否合法 4     IF newClaimStatus NOT IN ('pending', 'approved', 'rejected') THEN 5         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid claim status'; 6     END IF; 7 8     -- 更新 insurance_payments 表中的 ClaimStatus 9     UPDATE insurance_payments 10    SET ClaimStatus = newClaimStatus 11    WHERE PaymentRecordID = paymentRecordID; 12 13    -- 根据 ClaimStatus 更新 paymentrecords 表中的 PaymentStatus 14    IF newClaimStatus = 'approved' THEN 15        UPDATE paymentrecords 16        SET PaymentStatus = 'paid' 17        WHERE PaymentRecordID = paymentRecordID; 18    ELSE 19        UPDATE paymentrecords 20        SET PaymentStatus = 'unpaid' 21        WHERE PaymentRecordID = paymentRecordID; 22    END IF; 23 END</pre>
存储 过程 执行 源码 (1 分)	<p>(截屏)</p>

```
void pay::updateClaimStatus()
{
    qDebug() << "Update Status Button Clicked"; // 调试输出
    QString paymentRecordId = ui->paymentRecordIdEdit->text();
    QString newClaimStatus = ui->statusEdit->text(); // 获取用户填写的新状态

    // 调用存储过程
    QSqlQuery query;
    query.prepare("CALL UpdatePaymentStatus(:paymentRecordId, :newClaimStatus)");
    query.bindValue(":paymentRecordId", paymentRecordId.toInt());
    query.bindValue(":newClaimStatus", newClaimStatus);

    if (!query.exec()) {
        QMessageBox::critical(this, "Stored Procedure Execution Failed", "Stored procedure execution failed: " + query.lastError().text());
        return;
    }

    // 刷新表格，显示更新后的数据
    QSqlQuery refreshQuery;
    refreshQuery.prepare("SELECT * FROM paymentrecords WHERE PaymentRecordID = :paymentRecordId");
    refreshQuery.bindValue(":paymentRecordId", paymentRecordId);

    if (!refreshQuery.exec()) {
        QMessageBox::critical(this, "Query Failed", "Query failed: " + refreshQuery.lastError().text());
        return;
    }

    QSqlQueryModel *model = new QSqlQueryModel;
    model->setQuery(refreshQuery);
    ui->tableView->setModel(model);

    model->setHeaderData(0, Qt::Horizontal, tr("支付序号"));
    model->setHeaderData(1, Qt::Horizontal, tr("用户名"));
    model->setHeaderData(2, Qt::Horizontal, tr("支付数额"));
    model->setHeaderData(3, Qt::Horizontal, tr("支付方式"));
    model->setHeaderData(4, Qt::Horizontal, tr("支付状态"));
    model->setHeaderData(5, Qt::Horizontal, tr("支付记录产生日期"));
}
```

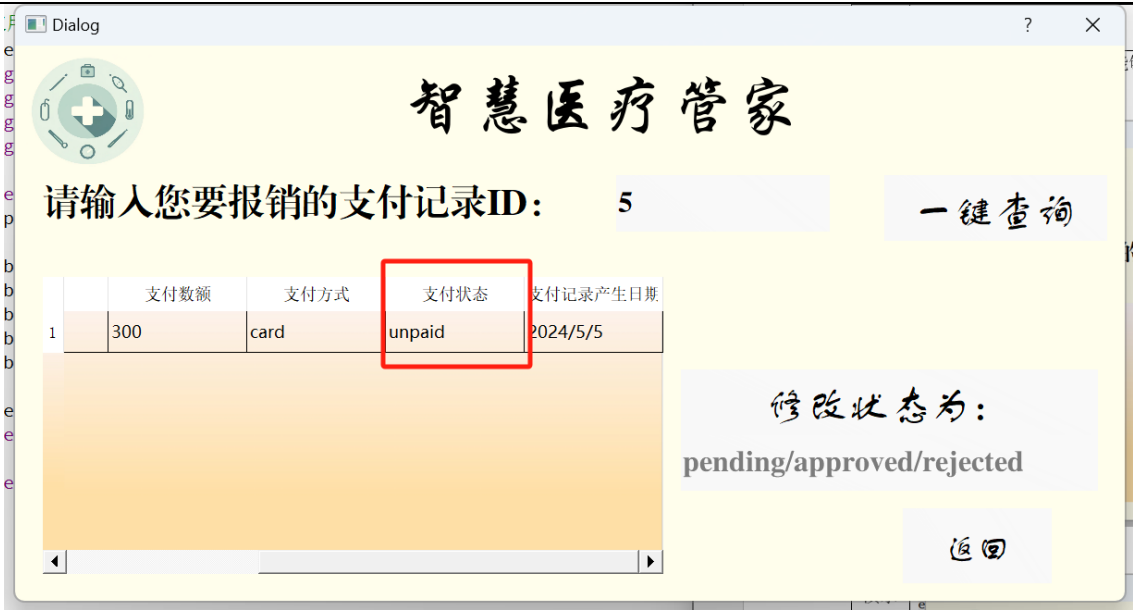
说明：不违背存储过程，能够执行更新操作

用户打开报销支付界面：



用户输入要修改的支付记录 ID，点击“一键查询”按钮后，即可看到该记录的详情信息。

程序  
演示  
(2  
分)



之后用户输入 `insurance_payments` 中要修改成的报销状态，在点击“修改状态为:”按钮，就可以看到在当前界面看到更新后的记录详情信息。



程序  
演示  
(2  
分)

说明：违背存储过程，系统报错；  
若病人用户没有按规定要求填写修改状态，即未填写 `pending/approved/rejected` 这三个值中的一个，而填写了其他非法的值，程序就会报错。



## 7. 含有视图的查询操作（15 分）

说明	<p>(1 分) 简要说明该操作所要完成的功能；</p> <p>(1 分) 简要说明建立的该视图的功能；</p> <p>(2 分) 简要说明该操作涉及的关系数据表（以“表名”的形式给出）</p> <p>(1 分) 简要说明表连接涉及的字段（以“表 1. 属性=表 2. 属性”）</p> <p>(6 分) 实现该操作的关键代码（高级语言、SQL），截图即可；</p> <p>(4 分) 如何执行该操作，按所述方法能够正常演示程序则给分。</p>
操作功能描述（1 分）	病人有时需要查看自己的看病记录，但是病人并不需要查看表中记录的所有字段，这时就需要引入视图，让病人只看到需要查看的字段信息即可。在查看预约记录界面，病人可以通过查询某条特定的预约记录，看到该记录的部分信息。
视图功能描述（1 分）	patient_appointment 视图把 patients 表和 appointments 表连接（join）到了一起，展示了 patients 表中的部分信息：Name，展示了 appointments 表中的部分信息：AppointmentID、AppointmentTime、AppointmentStatus 以及 AppointmentReason，有效呈现了用户需要的字段信息。
涉及的关系表（2 分）	patients、appointments
表连接字段（1 分）	<p>表连接涉及到的字段为： patients 表的 UserID 和 appointments 表的 UserID</p> <pre> 1  SELECT 2      `patients`.`Name` AS `Name`, 3      `appointments`.`AppointmentID` AS `AppointmentID`, 4      `appointments`.`AppointmentTime` AS `AppointmentTime`, 5      `appointments`.`AppointmentStatus` AS `AppointmentStatus`, 6      `appointments`.`AppointmentReason` AS `AppointmentReason` 7  FROM 8      ( 9      `appointments` 10     JOIN `patients` ON (( 11         `appointments`.`UserID` = `patients`.`UserID` 12     ))) </pre>
创	（截屏）

建视图代码（3分）	<div><div>定义</div><div>高级</div><div>SQL 预览</div></div> <pre>1 SELECT 2     `patients`.`Name` AS `Name`, 3     `appointments`.`AppointmentID` AS `AppointmentID`, 4     `appointments`.`AppointmentTime` AS `AppointmentTime`, 5     `appointments`.`AppointmentStatus` AS `AppointmentStatus`, 6     `appointments`.`AppointmentReason` AS `AppointmentReason` 7 FROM 8     ( 9         `appointments` 10        JOIN `patients` ON (( 11            `appointments`.`UserID` = `patients`.`UserID` 12        )))</pre>
查询代码（3分）	<p>（截屏）</p> <pre>void details::searchAppointmentByID() {     // 获取输入的 AppointmentID     QString appointmentID = ui-&gt;appointmentIDLineEdit-&gt;text();      // 构建查询字符串     QString queryStr = QString("SELECT * FROM patient_appointment WHERE AppointmentID = %1").arg(appointmentID);      // 设置查询并执行     model-&gt;setQuery(queryStr);      // 检查查询是否有结果     if (model-&gt;rowCount() == 0) {         QMessageBox::information(this, "提示", "没有找到相关的预约记录。", QMessageBox::Ok);     } else {         // 设置表头         model-&gt;setHeaderData(0, Qt::Horizontal, tr("姓名"));         model-&gt;setHeaderData(1, Qt::Horizontal, tr("预约号"));         model-&gt;setHeaderData(2, Qt::Horizontal, tr("预约时间"));         model-&gt;setHeaderData(3, Qt::Horizontal, tr("预约状态"));         model-&gt;setHeaderData(4, Qt::Horizontal, tr("预约原因"));         // 更新视图的模型         ui-&gt;appointmentsTableView-&gt;setModel(model);     } }</pre>
程序演示（4分）	<p>（截屏）</p> <p>在预约界面内，点击“查询预约”按钮到达查询界面。</p> <div></div>

在查询界面，用户输入要查询的特定的预约记录 ID，点击“一键查询”按钮，即可在该界面显示出 patient\_appointment 视图，只保留了一些用户需要的字段。

Dialog

智慧医疗管家

请输入您要查询的预约记录ID:

6

一键查询

这条预约记录如右图:

	姓名	预约号	预约时间	预约状态	子
1	李四	6	2024/5/20 10:...	completed	头痛

返回

备注