数据库课程设计

|  |  |
| --- | --- |
| 题 目： | 医院信息管理系统 |
| 专 业： | 医学信息工程 |
| 年 级： | 2022级 |
| 指导老师： | 刘慧玲，刘秀峰，何晓华 |

目录

[第一章 绪论 3](#_Toc170063194)

[1.1 选题背景 3](#_Toc170063195)

[1.2 相关技术介绍 4](#_Toc170063196)

[第二章 系统总体设计 5](#_Toc170063197)

[2.1 系统概述 5](#_Toc170063198)

[2.1.1 系统的设计目的 5](#_Toc170063199)

[2.1.2 需求分析来源 5](#_Toc170063200)

[2.1.3 系统适用对象 5](#_Toc170063201)

[2.1.4 系统设计亮点 6](#_Toc170063202)

[2.2 系统总体功能结构图 6](#_Toc170063203)

[2.3 开发环境 7](#_Toc170063204)

[2.4 系统总体结构功能图 8](#_Toc170063205)

[2.5 系统功能模块及功能描述 8](#_Toc170063206)

[第三章 数据库设计 10](#_Toc170063207)

[3.1 需求分析 10](#_Toc170063208)

[3.1.1 业务流程图 10](#_Toc170063209)

[3.1.2 数据字典 10](#_Toc170063210)

[3.1.3 数据流图 13](#_Toc170063211)

[3.2 概念结构设计 14](#_Toc170063212)

[3.3 逻辑结构设计 15](#_Toc170063213)

[3.3.1 E-R图向关系模式转换原则 15](#_Toc170063214)

[3.3.2 关系模式规范化 15](#_Toc170063215)

[3.4 物理结构设计 16](#_Toc170063216)

[3.4.1 存储结构和存储方法 16](#_Toc170063217)

[3.5 数据库建立 18](#_Toc170063218)

[3.5.1 建立数据库 18](#_Toc170063219)

[3.5.2 创建数据表和主键 18](#_Toc170063220)

[3.5.3 创建外键约束 23](#_Toc170063221)

[3.5.4 创建索引 25](#_Toc170063222)

[3.5.5 创建用户和授权 26](#_Toc170063223)

[3.5.6 触发器的实现 27](#_Toc170063224)

[3.6 数据库实施 29](#_Toc170063225)

[3.6.1 数据表截图（null） 29](#_Toc170063226)

[3.6.2 触发器实现截图 29](#_Toc170063227)

[第四章 数据库安全性和完整性设计 31](#_Toc170063228)

[4.1 安全性设计 31](#_Toc170063229)

[4.2 完整性设计 38](#_Toc170063230)

[第五章 程序开发与设计 39](#_Toc170063231)

[5.1 UI界面设计（null） 39](#_Toc170063232)

[5.2 程序开发、设计与完善 39](#_Toc170063233)

[5.2.1 程序测试—见现场演示 39](#_Toc170063234)

[5.2.2 程序完善 39](#_Toc170063235)

[第六章 总结与收获 40](#_Toc170063236)

[6.1 总结（null） 40](#_Toc170063237)

[6.2 团队成员的心得与收获（null） 40](#_Toc170063238)

[参考文献: 40](#_Toc170063239)

# 绪论

## 选题背景

我国医院信息管理系统在不同历史时期有着不同的发展阶段，从上世纪70年代开始，我国医院的信息化建设一直是以提高管理工作效率、辅助财政核算为主要目的[1]。进入现代，随着信息技术的飞速发展，医院信息管理系统已成为现代医疗行业不可或缺的一部分。这些系统通过集成医疗流程中的各个环节，提高了医疗服务的效率和质量，同时为医院管理层提供了决策支持。医院信息管理系统的设计不仅需要考虑医疗流程的复杂性，还要确保数据的安全性和系统的可靠性。从早期的单机应用到现在的全院级系统，再到区域医疗信息化的探索，医院信息系统的发展经历了多个阶段，每个阶段都反映了技术进步和医疗需求的变化。

在当前的大数据时代背景下，医院信息管理系统的设计面临着新的挑战和机遇[2]。一方面，系统需要处理和分析海量的医疗数据，如电子病历、医疗影像、检验报告等，以提高诊疗的精准度和医院运营效率；另一方面，随着医疗信息化建设的深入，不同信息系统之间的数据交换和共享成为提升医疗服务质量的关键。然而，不同系统间的数据格式和传输协议的差异导致了“信息孤岛”现象[3]，这不仅影响了信息资源的有效利用，也制约了医院信息化建设的进一步发展。因此，设计一个高效、安全且能够促进数据共享的医院信息管理系统，对于提升医疗服务水平、优化医疗资源配置、实现区域医疗信息共享具有重要意义[4]。本文将探讨医院信息管理系统的设计原则、关键技术和实现策略，以期为医疗信息化建设提供参考和指导。

## 相关技术介绍

**Microsoft SQL Server：SQL Server** 是Microsoft 公司推出的关系型数据库管理系统，具有使用方便可伸缩性好与相关软件集成程度高等优点。Microsoft SQL Server 数据库引擎为关系型数据和结构化数据提供了更安全可靠的存储功能，用于构建和管理用于业务的高可用和高性能的数据应用程序。

**Visual Studio：Visual Studio**（简称VS）是美国微软公司的开发工具包系列产品。VS是一个基本完整的开发工具集，包括了整个软件生命周期中所需要的大部分工具，如UML工具、代码管控工具、集成开发环境(IDE)等等。所写的目标代码适用于微软支持的所有平台，其中就包括了SQL Server。

**Axure RP：Axure RP**是美国Axure Software Solution公司旗舰产品，是一个专业的快速原型设计工具，让负责定义需求和规格、设计功能和界面的专家能够快速创建应用软件或Web网站的线框图、流程图、原型和规格说明文档。

**Microsoft Visio：Microsoft Visio**是Windows操作系统下运行的流程图和矢量绘图软件。本项目使用此应用制作流程图和ER图。

**亿图脑图MindMaster：MindMaster**思维导图是深圳市亿图软件有限公司推出的一款跨平台思维导图软件。软件提供了丰富的智能布局、多样性的展示模式、结合精美的设计元素和预置的主题样式，在整理工作思路，简化工作流程等方面的工作较为实用，本项目使用此应用制作ER图**。**

# 系统总体设计

## 系统概述

### 系统的设计目的

医院对于信息管理系统的需求是多方面的，主要围绕提高医疗服务效率、优化资源配置及增强患者体验等方面。本次课程设计我们通过分析整合医院的各个部门和流程，包括预约、诊断、治疗、药品管理、病历记录等，通过数据库实现对数据的集中管理和实时更新。通过高效的信息流通，医生和护士可以快速获取病人的医疗历史和当前状况，从而做出更准确的诊断和治疗决策。此外，系统还能够帮助医院管理层进行资源分配和财务分析，优化运营成本，提高医院的整体服务水平。通过这种信息化手段，医院能够更好地满足患者的需求，提升患者满意度，同时也为医院的可持续发展打下坚实的基础。

### 需求分析来源

本次课程设计，我们小组通过网络媒介搜集资料及参考文献，询问指导老师等措施完成需求分析。

### 系统适用对象

|  |  |
| --- | --- |
| 适用对象 | 主要操作 |
| 病人 | 挂号，住院登记，出院登记，药物缴费 |
| 医生 | 就诊药单、住院药单登记 |
| 管理员 | 医生、病房、药物、收银员、科室信息登记 |

### 系统设计亮点

1. 高度集成性:本系统将医院的各个部门及处理流程整合到了一个统一的平台上，实现了数据的统一性和流通性。
2. 数据保密性和安全性良好：本系统将用户分为病人、医生、管理员三类，不同用户登录系统后只能查看自己应用范围内的数据和进行规定的操作流程，不同用户之间无法看到其他用户的数据和操作，保护了用户的隐私及增强了数据的保密性和安全性。
3. 操作界面简洁易上手：本系统的登陆及操作界面简洁的同时保证了各种操作完善可用，方便不同用户上手操作，操作难度低，减少了用户操作难的问题，提升了系统使用的效率。

## 系统总体功能结构图

本系统基于.NetFrame的分层架构设计，如图2.2所示：

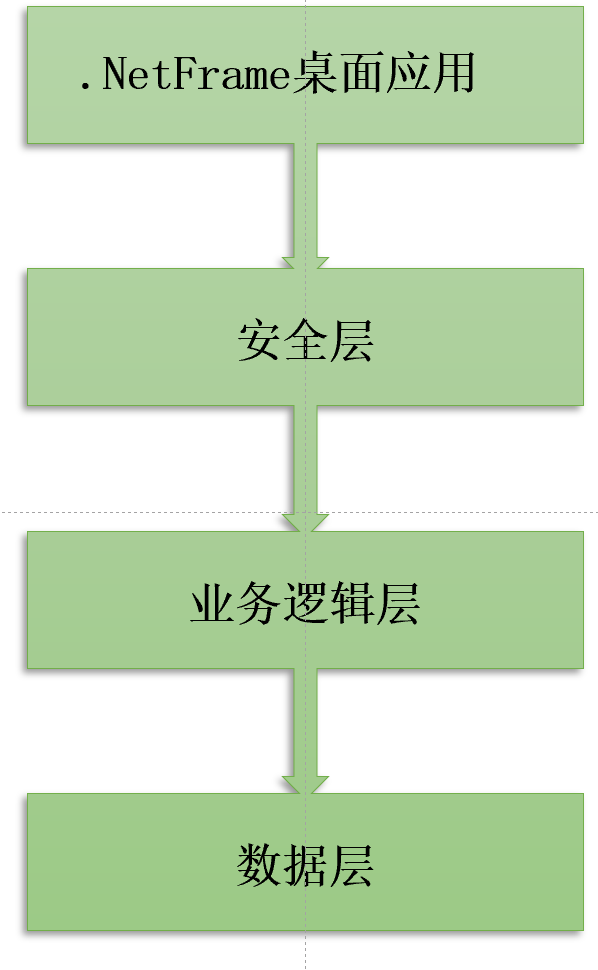


图2.2 医院信息管理系统分层架构示意图

## 开发环境

|  |  |
| --- | --- |
| 软件环境 | |
| Microsoft SQL Server2008 | **数据库设计与开发** |
| Visual Studio 2022 | **程序设计** |
| Axure Rp9 | **前端设计** |
| Microsoft Visio 2019  亿图脑图MindMaster | **流程图、ER图设计** |
| 硬件环境 | |
| 处理器 | **AMD Ryzen 7 6800H with Radeon Graphics 3.20 GH** |
| RAM | **16GB** |
| 硬盘 | **1TB** |

图2.3 开发环境介绍图

## 系统总体结构功能图

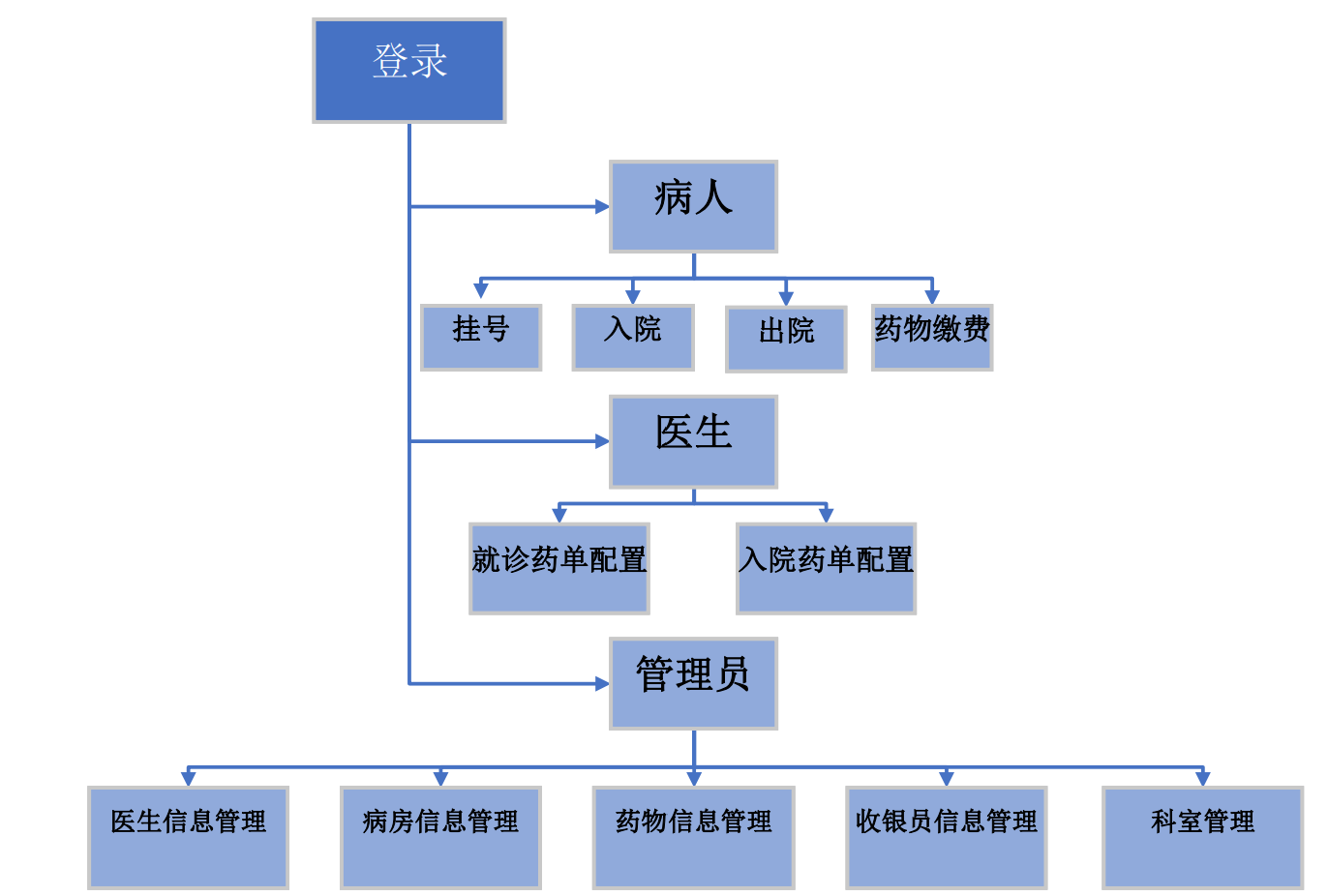


图2.4 系统总体结构功能图

## 系统功能模块及功能描述

|  |  |  |
| --- | --- | --- |
| 功能模块 | 子功能 | 功能描述 |
| 系统登录 | 系统登录 | 系统登录 |
| 账号注册 | 未注册账号的病人在登录系统时需要先创建账号 |
| 修改密码 | 用户或职工可以根据保密需要修改自己的账号密码 |
| 病人就诊 | 挂号 | 病人就诊需要挂号预约 |
| 办理住院 | 病人如有住院需求需要办理住院手续 |
| 办理出院 | 病人出院需要办理出院手续 |
| 就诊缴费 | 病人就诊后支付诊疗费和药物费用 |
| 信息管理 | 医生信息管理 | 管理员对医生信息进行管理 |
| 病房信息管理 | 管理员对病房信息进行管理 |
| 药物信息管理 | 管理员对药物信息进行管理 |
| 科室管理 | 管理员对科室信息进行管理 |
| 收银员管理 | 管理员对收银员信息进行管理 |
| 用户信息管理 | 管理员对用户信息进行管理 |
| 药单配置 | 就诊药单登记 | 医生对病人就诊时所开药单进行登记 |
| 入院药单登记 | 医生对病人入院治疗时所开药单进行登记 |

# 数据库设计

## 需求分析

### 业务流程图

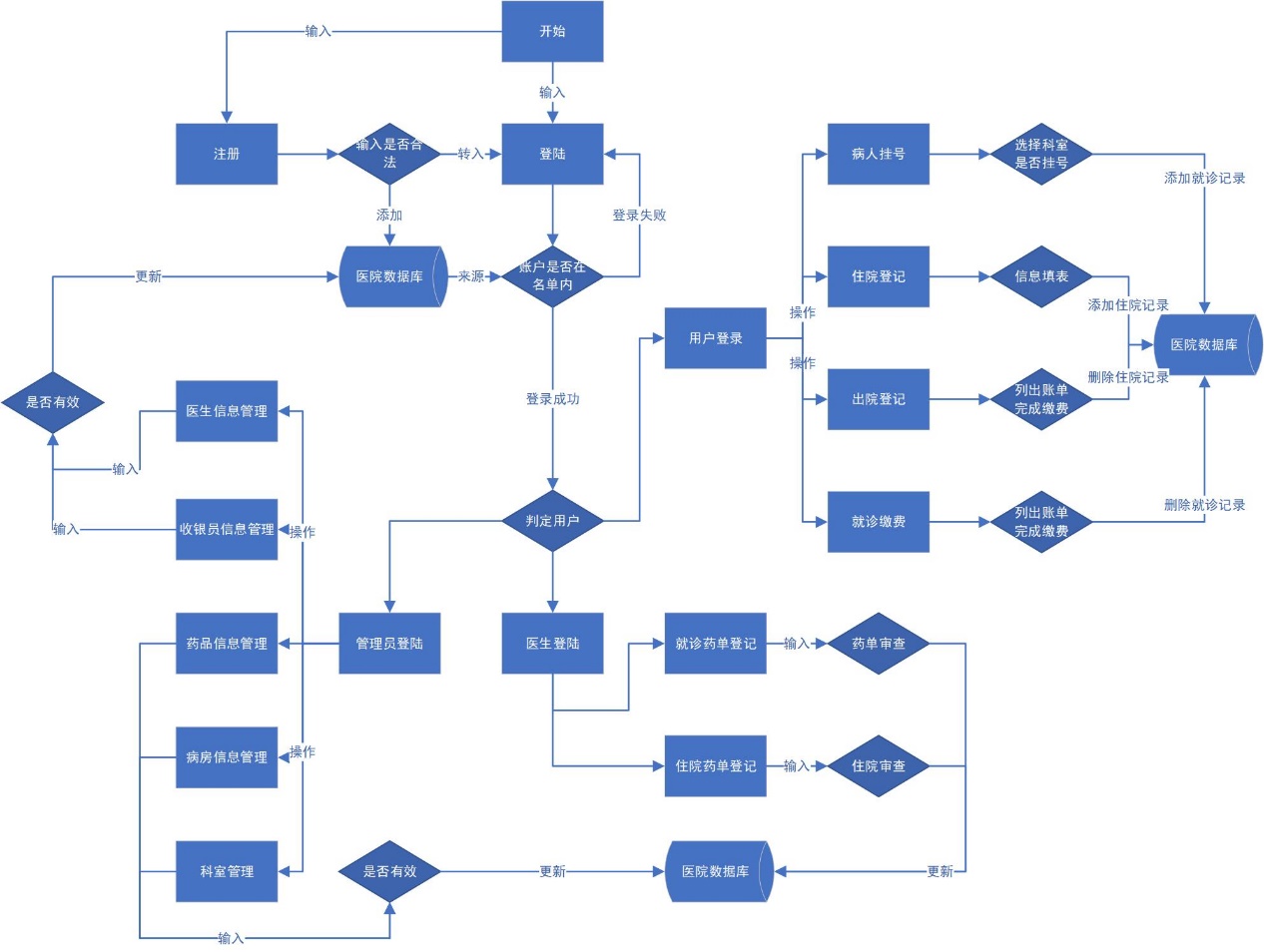


图3.1.1 数据库总体流程图

### 数据字典

表3.1.2 数据字典

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据项表 | | | | |
| 数据项编号 | 数据项号 | 别名 | 类型 | 是否允许Null |
| DI\_1 | [User] | 用户名 | varchar(20) | NOT NULL |
| DI\_2 | password | 密码 | varchar(20) | NOT NULL |
| DI\_3 | id | 住院编号 | varchar(50) | NULL |
| DI\_4 | Reid | 就诊编号 | varchar(50) | NULL |
| DI\_5 | User\_doctor | 负责医生号 | varchar(50) | NULL |
| DI\_6 | User\_bool | 用户状态 | varchar(50) | NULL |
| DI\_7 | Doctor\_id | 医生编号 | varchar(50) | NOT NULL |
| DI\_8 | Doctor\_name | 医生姓名 | nchar(10) | NOT NULL |
| DI\_9 | Doctor\_sex | 医生性别 | nchar(10) | NOT NULL |
| DI\_10 | Doctor\_age | 医生年龄 | int | NOT NULL |
| DI\_11 | Doctor\_workyears | 工龄 | int | NULL |
| DI\_12 | Doctor\_dment | 部门编号 | varchar(50) | NOT NULL |
| DI\_13 | Doctor\_phone | 电话 | varchar(50) | NULL |
| DI\_14 | Doctor\_pro | 职称 | varchar(50) | NOT NULL |
| DI\_15 | Doctor\_bool | 医生状态 | varchar(50) | NULL |
| DI\_16 | Doctor\_password | 医生密码 | varchar(50) | NULL |
| DI\_18 | Dpt\_id | 科室编号 | varchar(50) | NOT NULL |
| DI\_19 | Dpt\_name | 科室名 | nchar(10) | NOT NULL |
| DI\_20 | Dpt\_dir | 主任编号 | varchar(50) | NULL |
| DI\_21 | Dpt\_site | 科室位置 | nchar(10) | NULL |
| DI\_22 | Dpt\_bool | 科室状态 | varchar(50) | NULL |
| DI\_23 | Patient\_id | 病人编号 | varchar(50) | NOT NULL |
| DI\_24 | Patient\_name | 病人姓名 | nchar(10) | NULL |
| DI\_25 | Patient\_sex | 病人性别 | nchar(10) | NULL |
| DI\_26 | Patient\_age | 年龄 | int | NULL |
| DI\_27 | Patient\_dpment | 住院科号 | varchar(50) | NULL |
| DI\_28 | Pantient\_phone | 电话 | varchar(50) | NULL |
| DI\_29 | Patient\_intime | 医治时间 | smalldatetime | NULL |
| DI\_30 | Patient\_outtime | 出院时间 | smalldatetime | NULL |
| DI\_31 | Patient\_cashier | 收银员编号 | varchar(50) | NULL |
| DI\_32 | Patient\_sickid | 病房号 | varchar(50) | NULL |
| DI\_33 | Patient\_sicknum | 病床号 | varchar(50) | NULL |
| DI\_34 | Patient\_dor | 主治医生 | varchar(50) | NOT NULL |
| DI\_35 | Patient\_bool | 住院状态 | varchar(50) | NULL |
| DI\_36 | Sickroom\_id | 病房编号 | varchar(50) | NOT NULL |
| DI\_37 | Sickroom\_num | 床位编号 | varchar(50) | NOT NULL |
| DI\_38 | Sickroom\_dment | 科室 | varchar(50) | NOT NULL |
| DI\_39 | Sickroom\_money | 收费标准 | nchar(10) | NOT NULL |
| DI\_40 | Sickroom\_status | 目前状态 | varchar(50) | NULL |
| DI\_41 | Sickroom\_bool | 病房状态 | varchar(50) | NULL |
| DI\_42 | Cashier\_id | 收银员编号 | varchar(50) | NOT NULL |
| DI\_43 | Cashier\_name | 收银员姓名 | nchar(10) | NOT NULL |
| DI\_44 | Cashier\_bool | 收银员状态 | varchar(50) | NULL |
| DI\_45 | Drug\_id | 药品编号 | varchar(50) | NOT NULL |
| DI\_46 | Drug\_name | 药品名 | nchar(10) | NOT NULL |
| DI\_47 | Drug\_num | 库存数量 | int | NULL |
| DI\_48 | Drug\_price | 单价 | money | NOT NULL |
| DI\_49 | Drug\_bool | 药品状态 | varchar(50) | NULL |
| DI\_50 | Pstion\_id | 药单编号 | varchar(50) | NOT NULL |
| DI\_51 | Pstion\_dor | 医生编号 | varchar(50) | NOT NULL |
| DI\_52 | Pstion\_pat | 病人编号 | varchar(50) | NOT NULL |
| DI\_53 | Pstion\_bool | 药单状态 | varchar(50) | NULL |
| DI\_54 | Include\_drugid | 药品编号 | varchar(50) | NOT NULL |
| DI\_55 | Include\_psid | 药单编号 | varchar(50) | NOT NULL |
| DI\_56 | Include\_date | 日期 | date | NOT NULL |
| DI\_57 | Include\_num | 数量 | int | NULL |
| DI\_58 | Register\_id | 挂号编号 | varchar(50) | NOT NULL |
| DI\_59 | Register\_name | 姓名 | varchar(50) | NULL |
| DI\_60 | Register\_sex | 性别 | varchar(50) | NULL |
| DI\_61 | Register\_age | 年龄 | int | NULL |
| DI\_62 | Register\_dor | 医生号 | varchar(50) | NULL |
| DI\_63 | Register\_bool | 挂号状态 | varchar(50) | NULL |
| DI\_64 | Register\_time | 挂号时间 | smalldatetime | NULL |
| DI\_65 | Register\_Cash | 收银员编号 | varchar(50) | NULL |
| DI\_66 | PstionRe\_id | 药单编号 | varchar(50) | NOT NULL |
| DI\_67 | Pstion\_dor | 医生编号 | varchar(50) | NULL |
| DI\_68 | Pstion\_Re | 病人编号 | varchar(50) | NULL |
| DI\_69 | Pstion\_bool | 就诊药单状态 | varchar(50) | NULL |
| DI\_70 | IncludeRe\_drugid | 药品编号 | varchar(50) | NOT NULL |
| DI\_71 | IncludeRe\_Reid | 药单编号 | varchar(50) | NOT NULL |
| DI\_72 | IncludeRe\_date | 日期 | date | NOT NULL |
| DI\_73 | Include\_num | 数量 | int | NULL |

### 数据流图

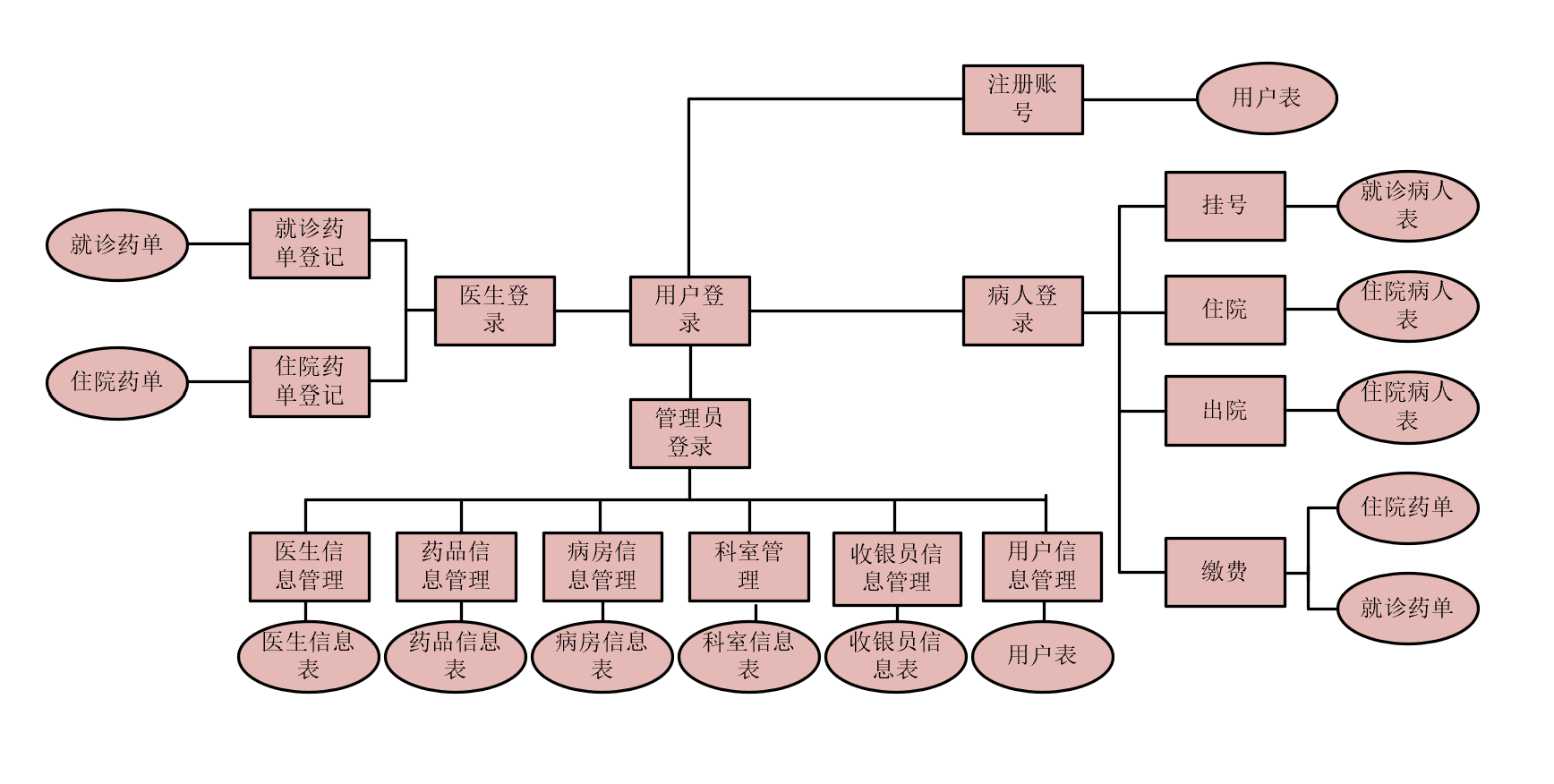


图3.1.3 数据流图

## 概念结构设计

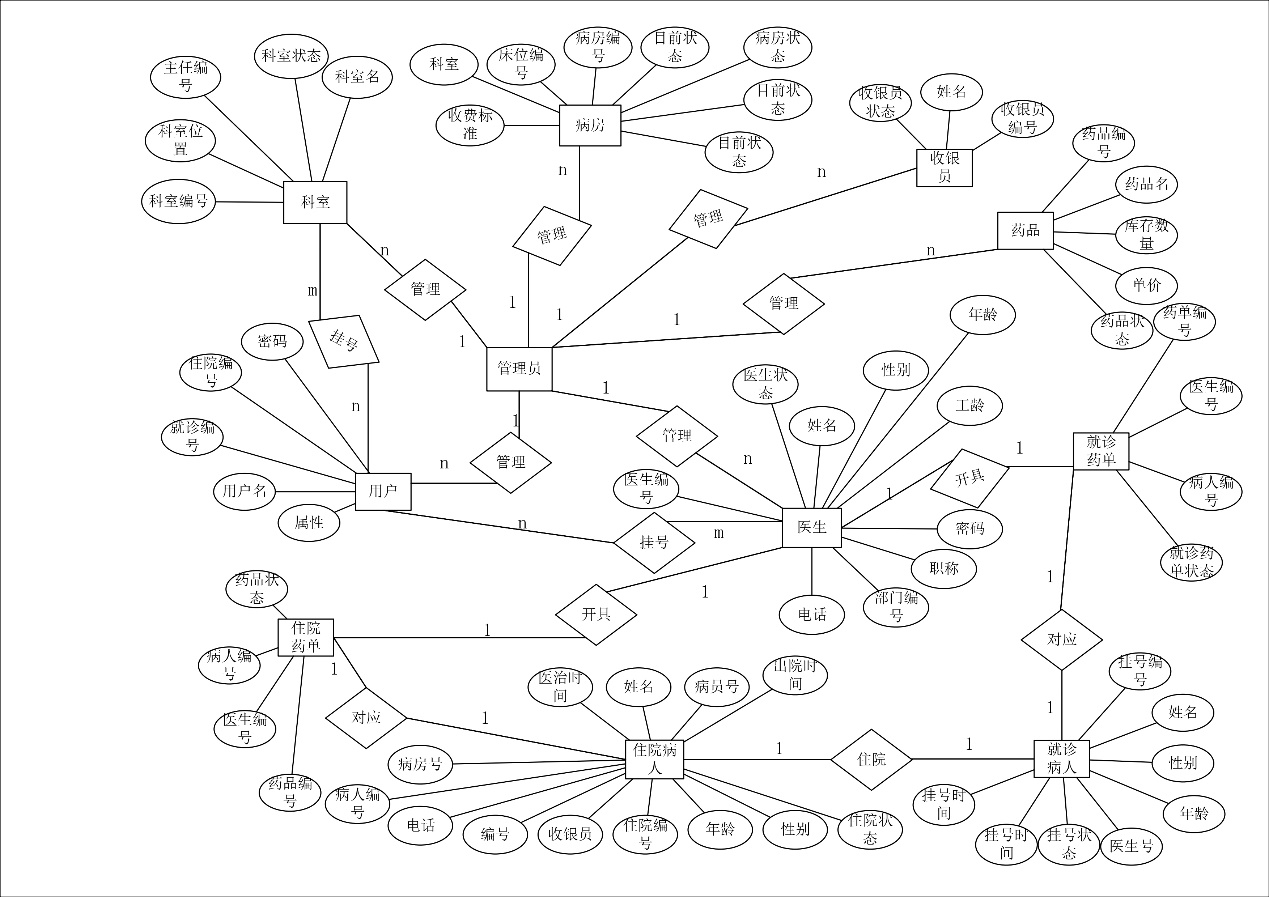


图3.2 全局E-R图

1. 就诊药单登记：完成对挂号病人开药单
2. 住院药单登记：完成对住院病人开药单
3. 医生信息管理：能够进行对医生信息的增加、删除、修改某些信息
4. 病房信息管理：能够进行对病房信息的增加、删除、修改某些信息
5. 药品信息管理：能够进行对药品信息的增加、删除、修改某些信息
6. 科室信息管理：能够进行对科室信息的增加、删除、修改某些信息
7. 收银员信息管理：能够进行对收银员信息的增加、删除、修改某些信息
8. 用户注册：用户将创建一个账号，凭借此账号进入用户界面
9. 挂号：用户能选择科室，查看医生选择是否挂号
10. 住院：用户完成信息填表，增加病人住院信息
11. 出院：列出病人消费账单，完成缴费，并删除病人住院信息
12. 就诊缴费：列出就诊消费账单，完成缴费，并删除病人就诊记录

## 逻辑结构设计

### E-R图向关系模式转换原则

1. 一个实体转化为一个表，实体的属性转化为表的列，实体的码转化为标的主键。
2. 实体间的联系根据联系的类型转化如下：

表3.3.1 关系实体联系表

|  |  |
| --- | --- |
| 1：1的联系 | 两个实体分别转化为表之后，只要在一个表中增加外键，一般在记录较少的表中增加属性，作为外键，该属性是另一个表的主键。 |
| 1：n的联系 | 在n端的实体对应的表中增加属性，该属性是1端实体对应的主键。 |
| m：n的联系 | 在通过引进一个新表来表达两个实体多对多的联系，新表的主键是由联系两端实体的主键组合而成，同时增加相关的联系属性。 |

### 关系模式规范化

表3.3.2 关系模式范式表

|  |  |  |  |
| --- | --- | --- | --- |
| 实体 | 表名 | 组成 | 范式 |
| 用户 | Users | （[User]，password，id，**Reid**，**User\_doctor**，User\_bool） | 3NF |
| 医生 | Doctor | （Doctor\_id，Doctor\_name，Doctor\_sex，Doctor\_age，Doctor\_workyears，Doctor\_dment，Doctor\_phone，Doctor\_pro，Doctor\_bool，Doctor\_password） | 3NF |
| 科室 | Dpt | （Dpt\_id，Dpt\_name，**Dpt\_dir**，Dpt\_site，Dpt\_bool） | 3NF |
| 住院病人 | Patient | （Patient\_id ，Patient\_name ，Patient\_sex ，Patient\_age，**Patient\_dpment**，Patient\_phone，Patient\_intime，Patient\_outtime，**Patient\_cashier**，**Patient\_sickid**，Patient\_sicknum，Patient\_dor，Patient\_bool） | 3NF |
| 病房 | Sickroom | （Sickroom\_id，Sickroom\_num，**Sickroom\_dment**，Sickroom\_money，Sickroom\_status，Sickroom\_bool） | 3NF |
| 收银员 | Cashier | （Cashier\_id，Cashier\_name，Cashier\_bool） | 3NF |
| 药品 | Drug | （Drug\_id，Drug\_name，Drug\_num，Drug\_price，Drug\_bool） | 3NF |
| 住院药单 | Pstion | （Pstion\_id，**Pstion\_dor**，**Pstion\_pat**，Pstion\_bool） | 3NF |
| 住院包括 | Include | （**Include\_drugid**，**Include\_psid**，Include\_date，Include\_num） | 3NF |
| 就诊病人 | Register | （Register\_id，Register\_name，Register\_sex，Register\_age，**Register\_dor**，Register\_bool，Register\_time，Register\_Cash） | 3NF |
| 就诊药单 | PstionRe | （PstionRe\_id，**PstionRe\_dor**，**PstionRe\_Re**，PstionRe\_bool） | 3NF |
| 就诊包括 | IncludeRe | （**IncludeRe\_drugid**，**IncludeRe\_Reid**，IncludeRe\_date，IncludeRe\_num） | 3NF |
| 备注 | 有下划线的属性或者属性组为候选码，标红的属性为主码，加粗的属性为外码 | | |

以上各表均满足第三范式：表中的所有数据元素不但要能唯一地被关键字所标识，而且它们之间相互独立，不存在其他的函数关系（传递函数依赖等）。

## 物理结构设计

### 存储结构和存储方法

1. **存储结构的确定**

本系统采用 SQL Server 2008 数据库，该数据库采用关系型模型，将数据组织成多个二维表，每个表包含若干行和列。每个表对应系统中的一个实体，例如用户、医生、科室等。表中的每一行代表一个实体的实例，每一列代表实体的一个属性。例如，Users 表包含用户编号、密码、姓名、性别等属性，每一行代表一个用户的详细信息。表与表之间通过外键关联，形成复杂的数据库结构。

1. **存放位置**

数据库文件将存放在服务器端，确保数据的安全性和可靠性。服务器将配备专业的数据库服务器软件，例如 SQL Server 2008，以提供高效的数据库管理和查询服务。数据库文件可以存放在服务器上的硬盘或固态硬盘，根据数据量和访问频率选择合适的存储介质。

1. **系统位置**
2. **备份与恢复**: 数据库将设置备份和恢复机制，定期进行数据备份，以防数据丢失或损坏。可以设置自动备份计划，将数据库文件备份到不同的存储位置，例如本地硬盘、网络存储或云存储。同时，也需要设置恢复机制，以便在数据丢失或损坏时能够快速恢复数据。
3. **访问控制**: 数据库将设置访问控制机制，确保只有授权用户才能访问数据库，保障数据安全。可以设置用户账户和密码，并根据用户角色分配不同的权限，例如只读权限、读写权限等。
4. **性能优化**: 数据库将进行性能优化，以提高数据查询效率。可以采用索引、分区、查询优化等技术，提高数据库的运行效率。
5. **存取方法的选择**

本系统采用 SQL 语言进行数据库操作，包括数据的增删改查等。SQL 语言是一种标准的关系数据库查询语言，功能强大，易于学习和使用。系统还将提供图形化界面，方便用户进行数据库操作。例如，用户可以通过界面查询患者的就诊记录、医生的排班信息等。

此外，C# 窗口应用程序将作为前端界面，负责与用户交互，并将用户输入的数据提交给数据库进行处理。C# 语言可以方便地连接 SQL Server 数据库，并执行 SQL 语句进行数据操作。

## 数据库建立

### 建立数据库

-- 创建数据库

CREATE DATABASE HOSPITAL;

### 创建数据表和主键

创建User表：

-- 创建Users 表

CREATE TABLE Users (

[User] varchar(20) NOT NULL,

password varchar(20) NOT NULL,

id varchar(50) NULL,

Reid varchar(50) NULL,

User\_doctor varchar(50) NULL,

User\_bool varchar(50) NULL,

CONSTRAINT PK\_Users PRIMARY KEY ([User])

);

创建Doctor 表：

-- 创建Doctor 表

CREATE TABLE Doctor (

Doctor\_id varchar(50) NOT NULL,

Doctor\_name nchar(10) NOT NULL,

Doctor\_sex nchar(10) NOT NULL,

Doctor\_age int NOT NULL,

Doctor\_workyears int NULL,

Doctor\_dment varchar(50) NOT NULL,

Doctor\_phone varchar(50) NULL,

Doctor\_pro varchar(50) NOT NULL,

Doctor\_bool varchar(50) NULL,

Doctor\_password varchar(50) NULL,

CONSTRAINT PK\_Doctor PRIMARY KEY (Doctor\_id)

);

创建Dpt 表：

-- 创建Dpt 表

CREATE TABLE Dpt (

Dpt\_id varchar(50) NOT NULL,

Dpt\_name nchar(10) NOT NULL,

Dpt\_dir varchar(50) NULL,

Dpt\_site nchar(10) NULL,

Dpt\_bool varchar(50) NULL,

CONSTRAINT PK\_Dpt PRIMARY KEY (Dpt\_id)

);

创建Patient 表：

-- 创建Patient 表

CREATE TABLE Patient (

Patient\_id varchar(50) NOT NULL,

Patient\_name nchar(10) NULL,

Patient\_sex nchar(10) NULL,

Patient\_age int NULL,

Patient\_dpment varchar(50) NULL,

Patient\_phone varchar(50) NULL,

Patient\_intime smalldatetime NULL,

Patient\_outtime smalldatetime NULL,

Patient\_cashier varchar(50) NULL,

Patient\_sickid varchar(50) NULL,

Patient\_sicknum varchar(50) NULL,

Patient\_dor varchar(50) NOT NULL,

Patient\_bool varchar(50) NULL,

CONSTRAINT PK\_Patient PRIMARY KEY (Patient\_id)

);

创建Sickroom 表：

-- 创建Sickroom 表

CREATE TABLE Sickroom (

Sickroom\_id varchar(50) NOT NULL,

Sickroom\_num varchar(50) NOT NULL,

Sickroom\_dment varchar(50) NOT NULL,

Sickroom\_money nchar(10) NOT NULL,

Sickroom\_status varchar(50) NULL,

Sickroom\_bool varchar(50) NULL,

CONSTRAINT PK\_Sickroom PRIMARY KEY (Sickroom\_id)

);

创建Cashier 表：

-- 创建Cashier 表

CREATE TABLE Cashier (

Cashier\_id varchar(50) NOT NULL,

Cashier\_name nchar(10) NOT NULL,

Cashier\_bool varchar(50) NULL,

CONSTRAINT PK\_Cashier PRIMARY KEY (Cashier\_id)

);

创建Drug 表：

-- 创建Drug 表

CREATE TABLE Drug (

Drug\_id varchar(50) NOT NULL,

Drug\_name nchar(10) NOT NULL,

Drug\_num int NULL,

Drug\_price money NOT NULL,

Drug\_bool varchar(50) NULL,

CONSTRAINT PK\_Drug PRIMARY KEY (Drug\_id)

);

创建Pstion 表：

-- 创建Pstion 表

CREATE TABLE Pstion (

Pstion\_id varchar(50) NOT NULL,

Pstion\_dor varchar(50) NOT NULL,

Pstion\_pat varchar(50) NOT NULL,

Pstion\_bool varchar(50) NULL,

CONSTRAINT PK\_Pstion PRIMARY KEY (Pstion\_id)

);

创建Include 表：

-- 创建Include 表

CREATE TABLE Include (

Include\_drugid varchar(50) NOT NULL,

Include\_psid varchar(50) NOT NULL,

Include\_date date NOT NULL,

Include\_num int NULL,

CONSTRAINT PK\_Include PRIMARY KEY (Include\_drugid, Include\_psid, Include\_date)

);

创建Register 表：

-- 创建Register 表

CREATE TABLE Register (

Register\_id varchar(50) NOT NULL,

Register\_name varchar(50) NULL,

Register\_sex varchar(50) NULL,

Register\_age int NULL,

Register\_dor varchar(50) NULL,

Register\_bool varchar(50) NULL,

Register\_time smalldatetime NULL,

Register\_Cash varchar(50) NULL,

CONSTRAINT PK\_Register PRIMARY KEY (Register\_id)

);

创建PstionRe 表：

-- 创建PstionRe 表

CREATE TABLE PstionRe (

PstionRe\_id varchar(50) NOT NULL,

PstionRe\_dor varchar(50) NULL,

PstionRe\_Re varchar(50) NULL,

PstionRe\_bool varchar(50) NULL,

CONSTRAINT PK\_PstionRe PRIMARY KEY (PstionRe\_id)

);

创建IncludeRe 表：

-- 创建IncludeRe 表

CREATE TABLE IncludeRe (

IncludeRe\_drugid varchar(50) NOT NULL,

IncludeRe\_Reid varchar(50) NOT NULL,

IncludeRe\_date date NOT NULL,

IncludeRe\_num int NULL,

CONSTRAINT PK\_IncludeRe PRIMARY KEY (IncludeRe\_drugid, IncludeRe\_Reid, IncludeRe\_date)

);

### 创建外键约束

-- Users 表外键约束

ALTER TABLE Users

ADD CONSTRAINT FK\_Users\_Doctor

FOREIGN KEY (User\_doctor) REFERENCES Doctor (Doctor\_id);

-- Doctor 表外键约束

ALTER TABLE Doctor

ADD CONSTRAINT FK\_Doctor\_Dpt

FOREIGN KEY (Doctor\_dment) REFERENCES Dpt (Dpt\_id);

-- Patient 表外键约束

ALTER TABLE Patient

ADD CONSTRAINT FK\_Patient\_Doctor

FOREIGN KEY (Patient\_dor) REFERENCES Doctor (Doctor\_id),

CONSTRAINT FK\_Patient\_Sickroom

FOREIGN KEY (Patient\_sickid, Patient\_sicknum) REFERENCES Sickroom (Sickroom\_id, Sickroom\_num);

-- Sickroom 表外键约束

ALTER TABLE Sickroom

ADD CONSTRAINT FK\_Sickroom\_Dpt

FOREIGN KEY (Sickroom\_dment) REFERENCES Dpt (Dpt\_id);

-- Pstion 表外键约束

ALTER TABLE Pstion

ADD CONSTRAINT FK\_Pstion\_Doctor

FOREIGN KEY (Pstion\_dor) REFERENCES Doctor (Doctor\_id),

CONSTRAINT FK\_Pstion\_Patient

FOREIGN KEY (Pstion\_pat) REFERENCES Patient (Patient\_id);

-- Include 表外键约束

ALTER TABLE Include

ADD CONSTRAINT FK\_Include\_Drug

FOREIGN KEY (Include\_drugid) REFERENCES Drug (Drug\_id),

CONSTRAINT FK\_Include\_Pstion

FOREIGN KEY (Include\_psid) REFERENCES Pstion (Pstion\_id);

-- Register 表外键约束

ALTER TABLE Register

ADD CONSTRAINT FK\_Register\_Doctor

FOREIGN KEY (Register\_dor) REFERENCES Doctor (Doctor\_id),

CONSTRAINT FK\_Register\_Cashier

FOREIGN KEY (Register\_Cash) REFERENCES Cashier (Cashier\_id);

-- PstionRe 表外键约束

ALTER TABLE PstionRe

ADD CONSTRAINT FK\_PstionRe\_Register

FOREIGN KEY (PstionRe\_Re) REFERENCES Register (Register\_id);

-- IncludeRe 表外键约束

ALTER TABLE IncludeRe

ADD CONSTRAINT FK\_IncludeRe\_Drug

FOREIGN KEY (IncludeRe\_drugid) REFERENCES Drug (Drug\_id),

CONSTRAINT FK\_IncludeRe\_PstionRe

FOREIGN KEY (IncludeRe\_Reid) REFERENCES PstionRe (PstionRe\_id);

### 创建索引

-- 创建Users 表索引

CREATE INDEX idx\_Users\_User ON Users ([User]);

-- 创建Doctor 表索引

CREATE INDEX idx\_Doctor\_Doctor\_id ON Doctor (Doctor\_id);

CREATE INDEX idx\_Doctor\_Doctor\_name ON Doctor (Doctor\_name);

-- 创建Dpt 表索引

CREATE INDEX idx\_Dpt\_Dpt\_id ON Dpt (Dpt\_id);

CREATE INDEX idx\_Dpt\_Dpt\_name ON Dpt (Dpt\_name);

-- 创建Patient 表索引

CREATE INDEX idx\_Patient\_Patient\_id ON Patient (Patient\_id);

CREATE INDEX idx\_Patient\_Patient\_name ON Patient (Patient\_name);

-- 创建Sickroom 表索引

CREATE INDEX idx\_Sickroom\_Sickroom\_id ON Sickroom (Sickroom\_id);

CREATE INDEX idx\_Sickroom\_Sickroom\_num ON Sickroom (Sickroom\_num);

-- 创建Cashier 表索引

CREATE INDEX idx\_Cashier\_Cashier\_id ON Cashier (Cashier\_id);

-- 创建Drug 表索引

CREATE INDEX idx\_Drug\_Drug\_id ON Drug (Drug\_id);

CREATE INDEX idx\_Drug\_Drug\_name ON Drug (Drug\_name);

-- 创建Pstion 表索引

CREATE INDEX idx\_Pstion\_Pstion\_id ON Pstion (Pstion\_id);

-- 创建Include 表索引

CREATE INDEX idx\_Include\_Include\_drugid ON Include (Include\_drugid);

CREATE INDEX idx\_Include\_Include\_psid ON Include (Include\_psid);

-- 创建Register 表索引

CREATE INDEX idx\_Register\_Register\_id ON Register (Register\_id);

-- 创建PstionRe 表索引

CREATE INDEX idx\_PstionRe\_PstionRe\_id ON PstionRe (PstionRe\_id);

-- 创建IncludeRe 表索引

CREATE INDEX idx\_IncludeRe\_IncludeRe\_drugid ON IncludeRe (IncludeRe\_drugid);

CREATE INDEX idx\_IncludeRe\_IncludeRe\_Reid ON IncludeRe (IncludeRe\_Reid);

### 创建用户和授权

-- 创建用户

CREATE LOGIN admin WITH PASSWORD = '123456';

CREATE USER admin FOR LOGIN admin;

-- 授予所有权限

GRANT ALL PRIVILEGES TO admin;

### 触发器的实现

1. 为Users表创建一个触发器，当插入新用户时检查密码长度。

*-- 创建Users表触发器*

CREATE TRIGGER tr\_Users\_Insert

ON Users

AFTER INSERT

AS

BEGIN

*-- 检查新插入的用户的密码长度*

IF EXISTS (SELECT 1 FROM inserted WHERE LEN(password) < 6)

BEGIN

*-- 密码长度不满足要求，抛出异常*

RAISERROR('密码长度不能少于6个字符。', 16, 1)

ROLLBACK TRANSACTION

END

END;

1. 为Doctor表创建一个触发器，当插入新医生信息时检查性别字段。

*-- 创建Doctor表触发器*

CREATE TRIGGER tr\_Doctor\_Insert

ON Doctor

AFTER INSERT

AS

BEGIN

*-- 检查新插入的医生的性别是否有效（例如，只允许'男'或'女'）*

IF EXISTS (SELECT 1 FROM inserted WHERE Doctor\_sex NOT IN ('男', '女'))

BEGIN

*-- 性别无效，抛出异常*

RAISERROR('医生的性别必须是“男”或“女”。', 16, 1)

ROLLBACK TRANSACTION

END

END;

1. 为Dpt表创建一个触发器，当插入新的科室信息时检查科室名称是否已存在。

*-- 创建Dpt表触发器*

CREATE TRIGGER tr\_Dpt\_Insert

ON Dpt

AFTER INSERT

AS

BEGIN

*-- 检查新插入的科室名称是否已存在*

IF EXISTS (SELECT 1 FROM Dpt WHERE Dpt\_name = (SELECT Dpt\_name FROM inserted))

BEGIN

*-- 科室名称已存在，抛出异常*

RAISERROR('科室名称已经存在。', 16, 1)

ROLLBACK TRANSACTION

END

END;

1. 为Patient表创建一个触发器，当插入新的患者信息时检查患者是否已经存在。

*-- 创建Patient表触发器*

CREATE TRIGGER tr\_Patient\_Insert

ON Patient

AFTER INSERT

AS

BEGIN

*-- 检查新插入的患者ID是否已存在*

IF EXISTS (SELECT 1 FROM Patient WHERE Patient\_id = (SELECT Patient\_id FROM inserted))

BEGIN

*-- 患者ID已存在，抛出异常*

RAISERROR('患者ID已经存在。', 16, 1)

ROLLBACK TRANSACTION

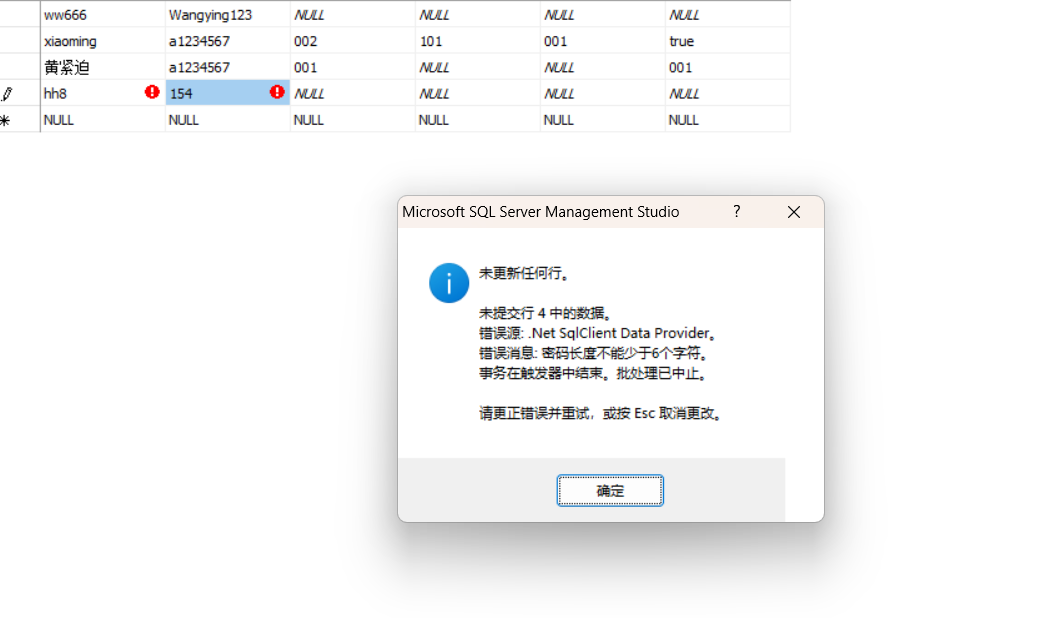
END

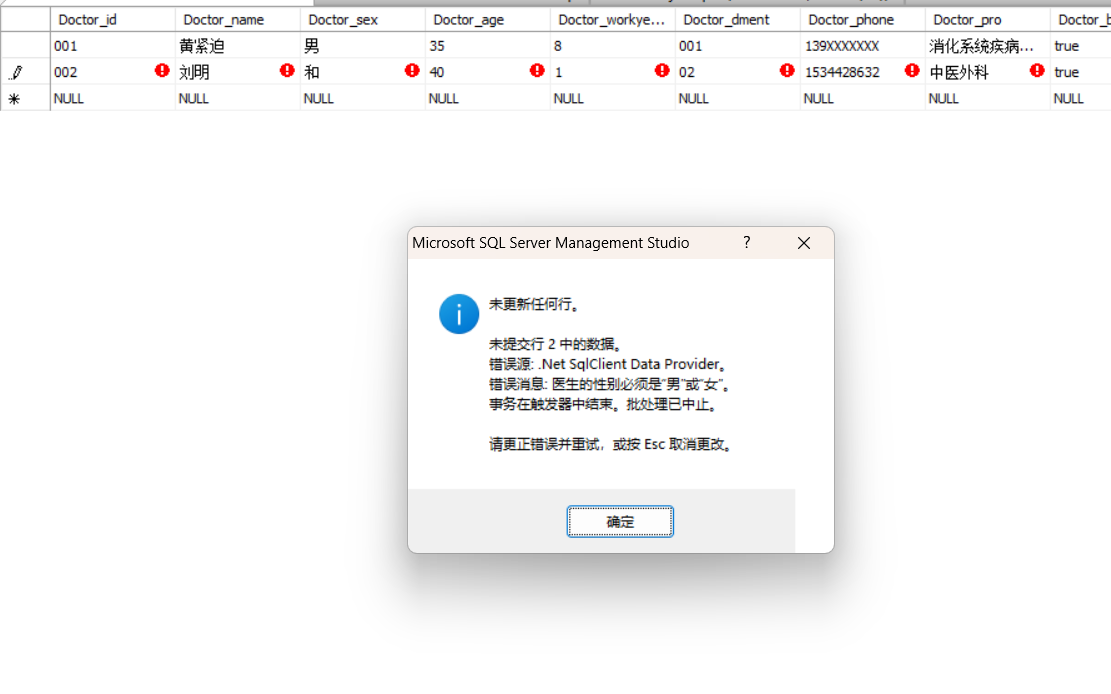
END;

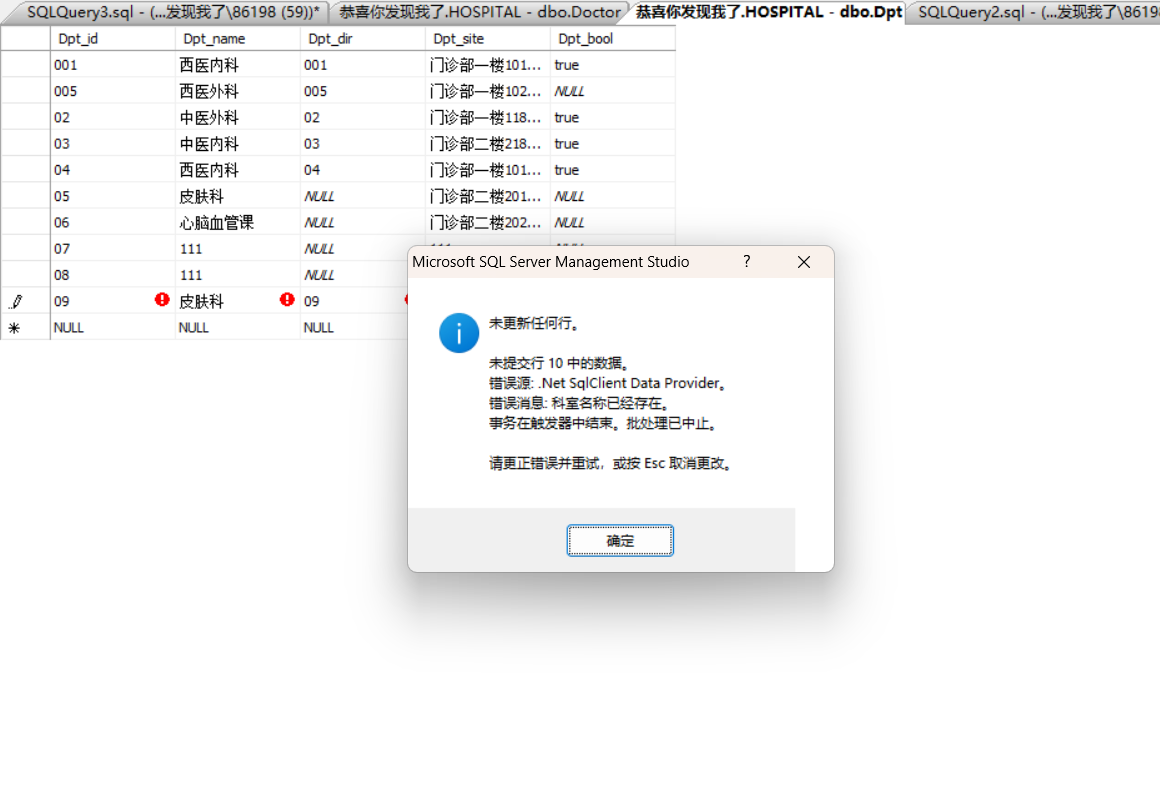
## 数据库实施

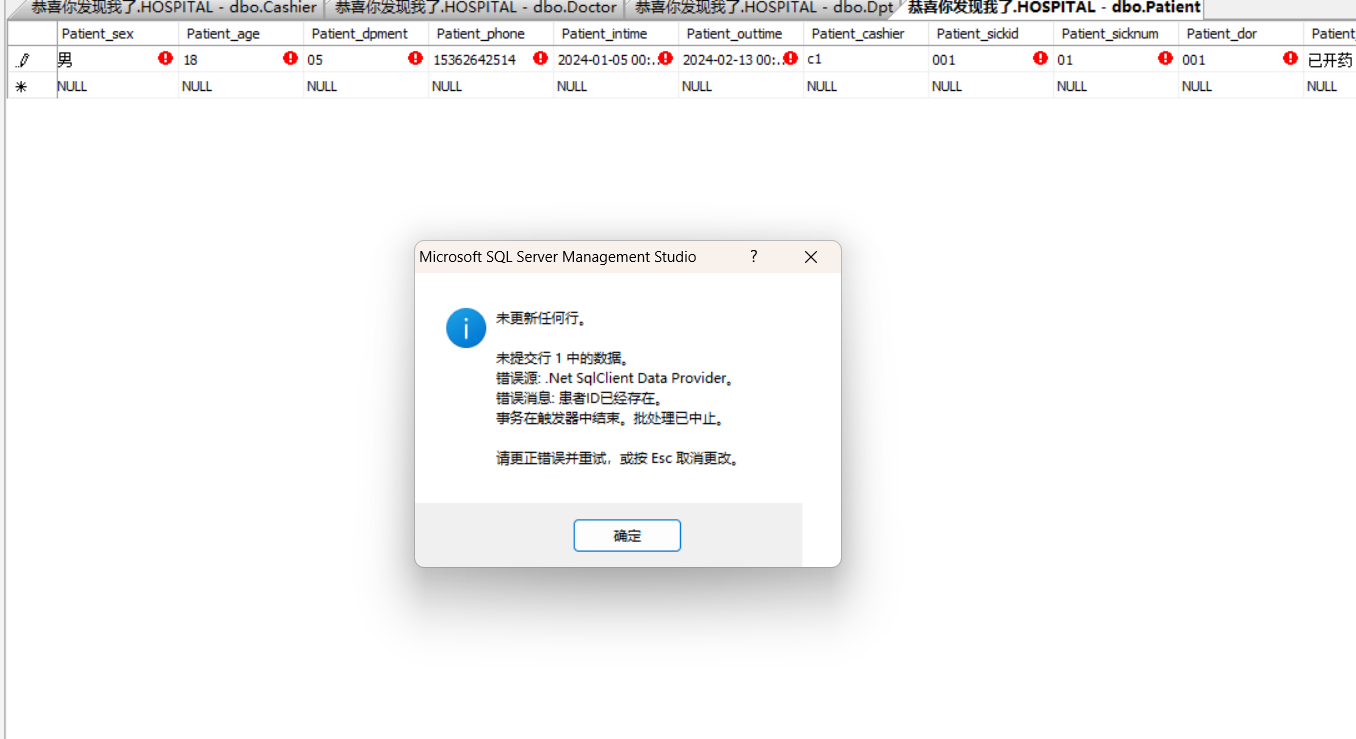
### 数据表截图（null）

### 触发器实现截图









# 数据库安全性和完整性设计

## 安全性设计

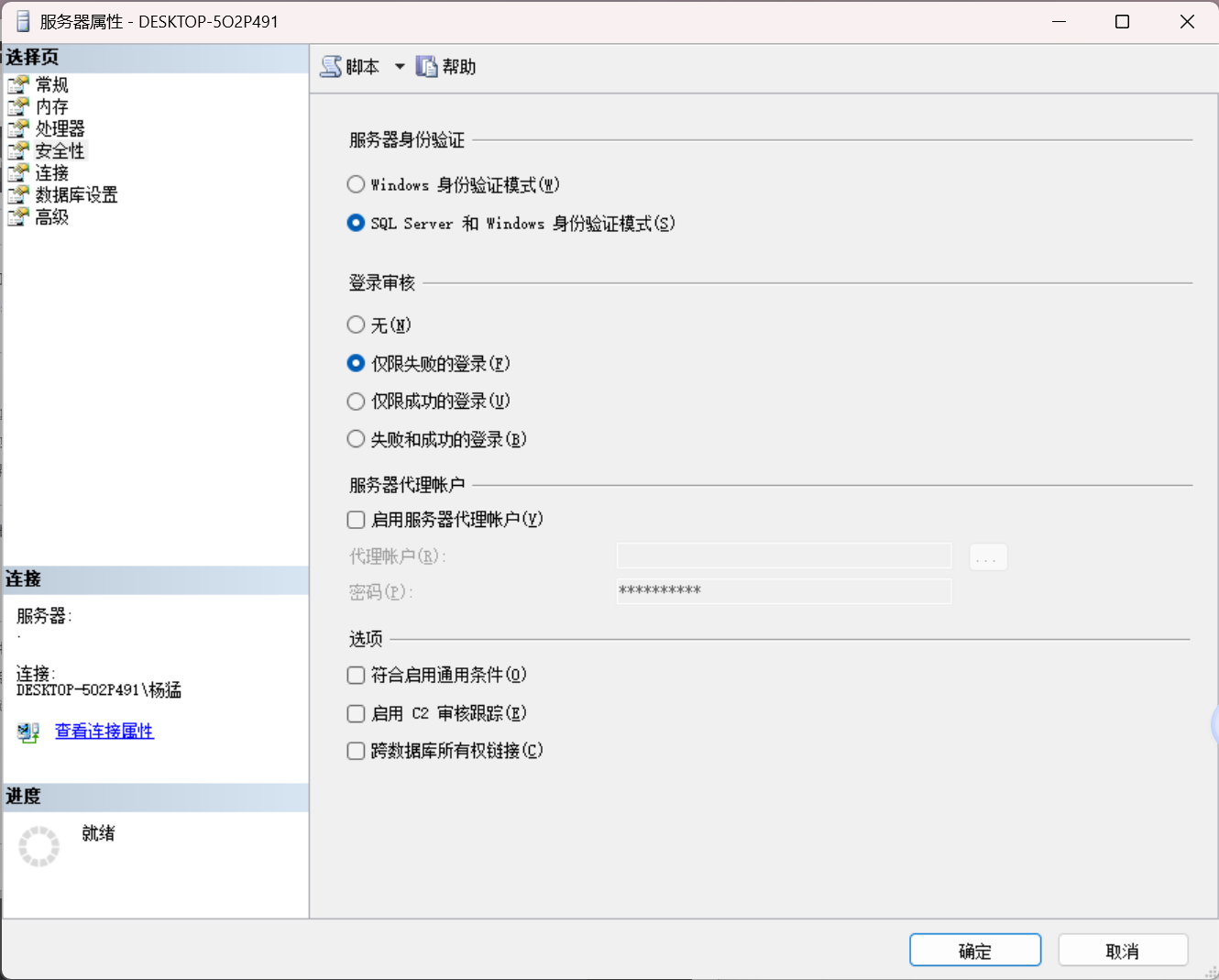
数据库的安全性是指保护数据库以防止未授权访问、数据泄露、更改或破坏。在医疗管理系统中，数据安全性尤为重要，因为其中包含大量敏感信息，如患者病历、个人隐私等。为了确保数据安全，本系统采取了以下措施：

1. **用户身份验证与访问控制**

本系统实施了多层次的用户身份验证机制，确保只有经过授权的用户才能访问数据库资源。采用 SQL Server 身份验证和 Windows 身份验证的混合模式，以提供灵活且安全的用户访问策略。此外，通过角色管理与权限分配机制，实施了细粒度的访问控制，确保用户仅能访问其工作职责所必需的数据资源。遵循最小权限原则，有效降低安全风险。

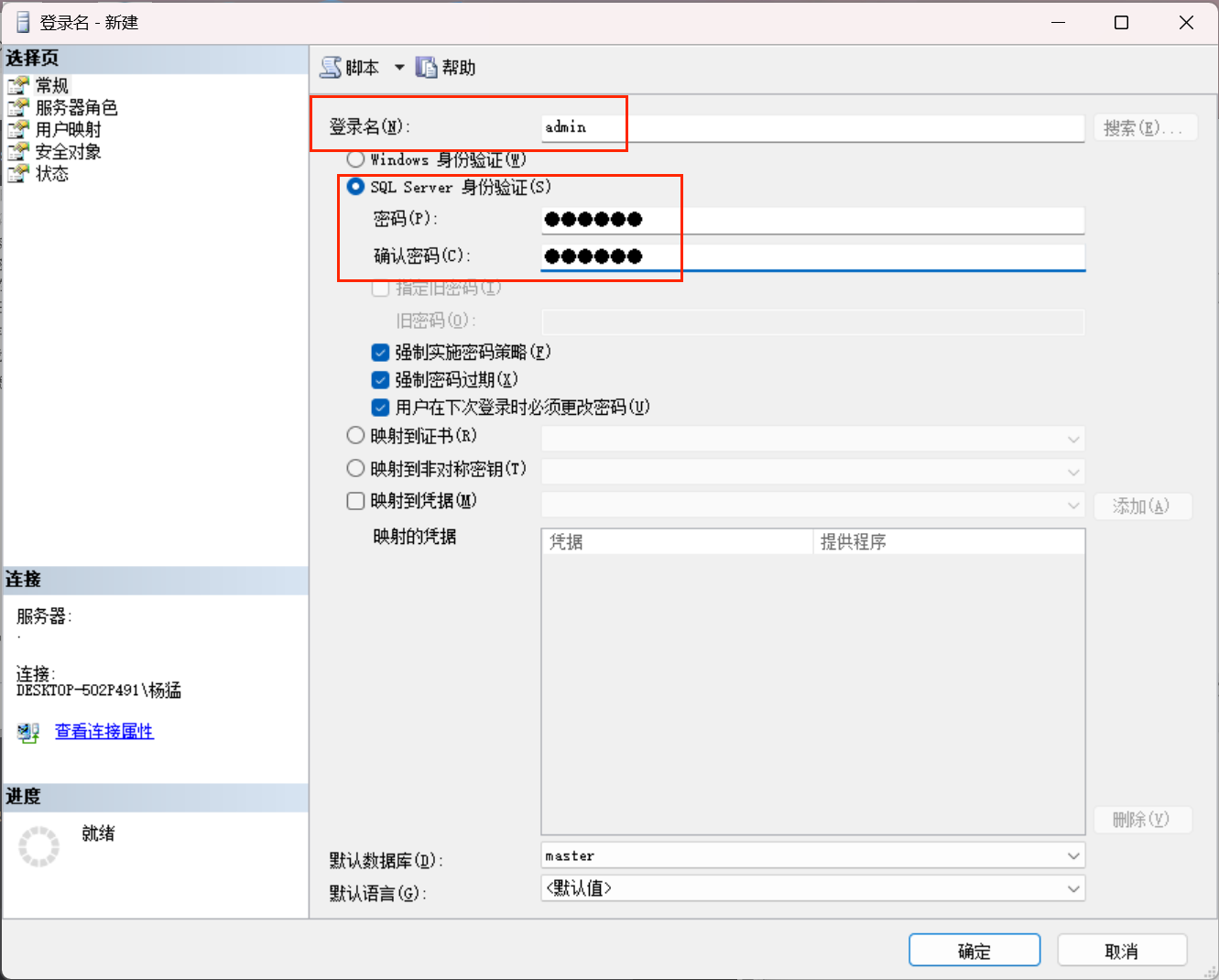
先更改数据库登录用户的验证方式为SQL server和Windows身份验证模式：

图4.1.1 数据库安全性设置1



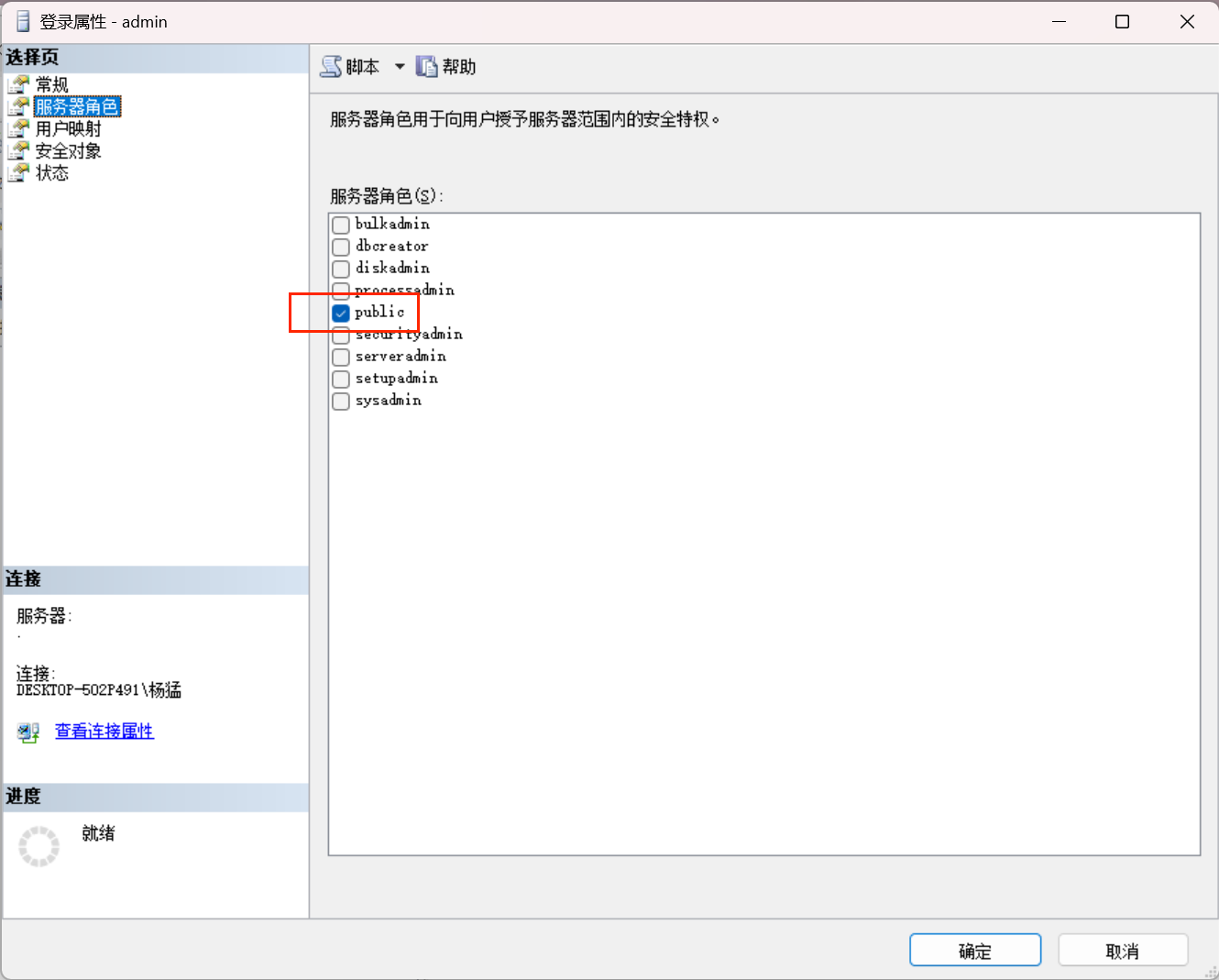
再创建新的用户admin，密码为123456，当admin用户使用账号和密码就可以登录到指定的数据库：

图4.1.2 数据库安全性设置2



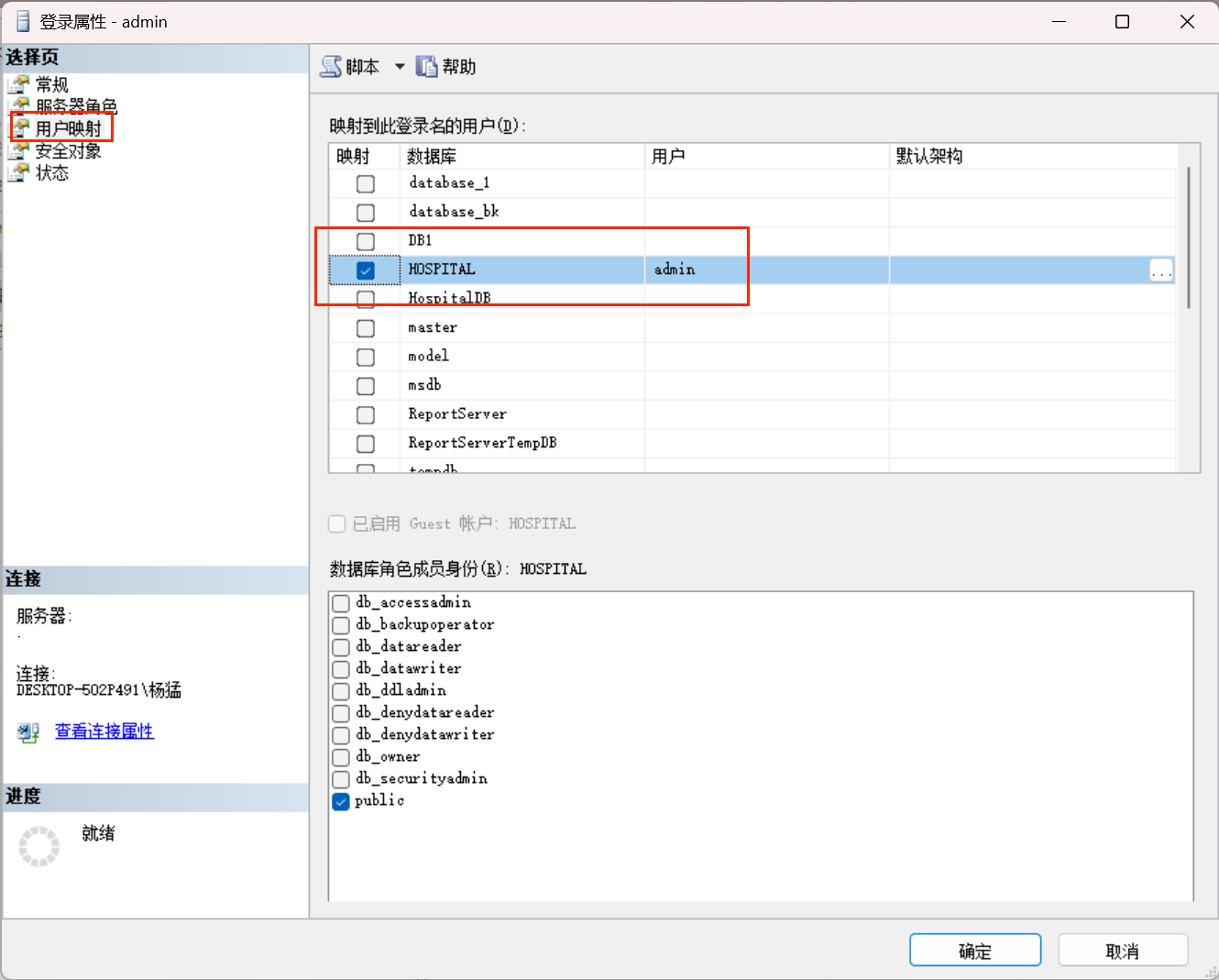
再然后对服务器角色向用户授权服务器范围内安全特权方式，在这里我们为admin选中的是public特权，这是SQLserver中默认的一类角色，若想让角色拥有服务器管理的最高权限可以选择sysadmin：

图4.13 数据库安全性设置3



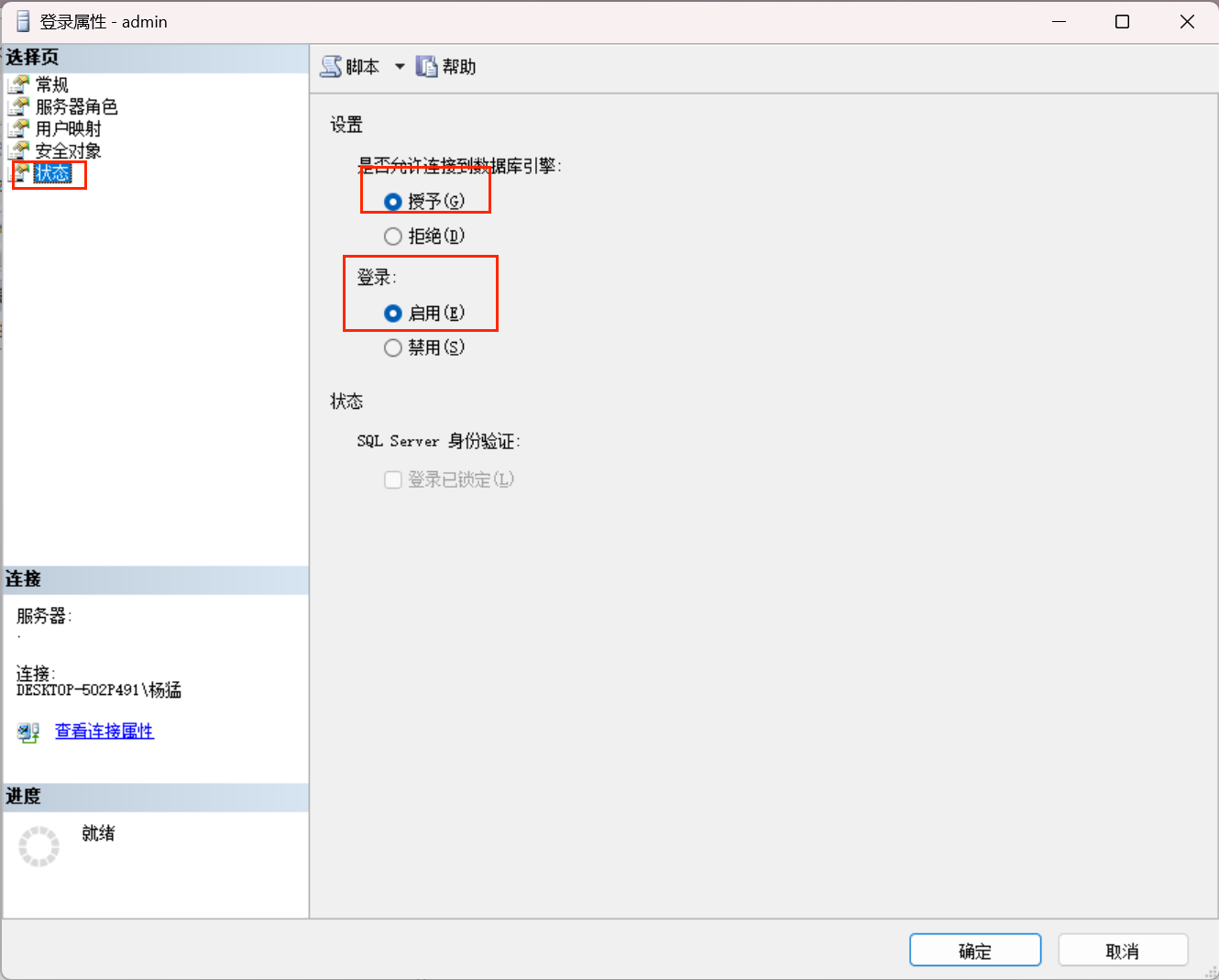
再然后在用户映射里面选中此账号能操作的数据库和登陆者的角色身份，在这里我们为admin设置的数据库是HOSPITAL，角色身份是public：

图4.1.4 数据库安全性设置4



最后在状态页面授权允许连接到数据库引擎和已启用的登录：

图4.1.5 数据库安全性设置5



1. 数据库的备份与恢复

为防止数据丢失和确保业务连续性，本系统执行定期的数据备份计划，并采用多种备份技术，如完整备份、差异备份和事务日志备份。同时，制定了详尽的意外恢复方案，包括数据库镜像和日志传送等高可用性技术，确保在意外发生时能够迅速恢复数据服务。

图4.1.6 备份数据库

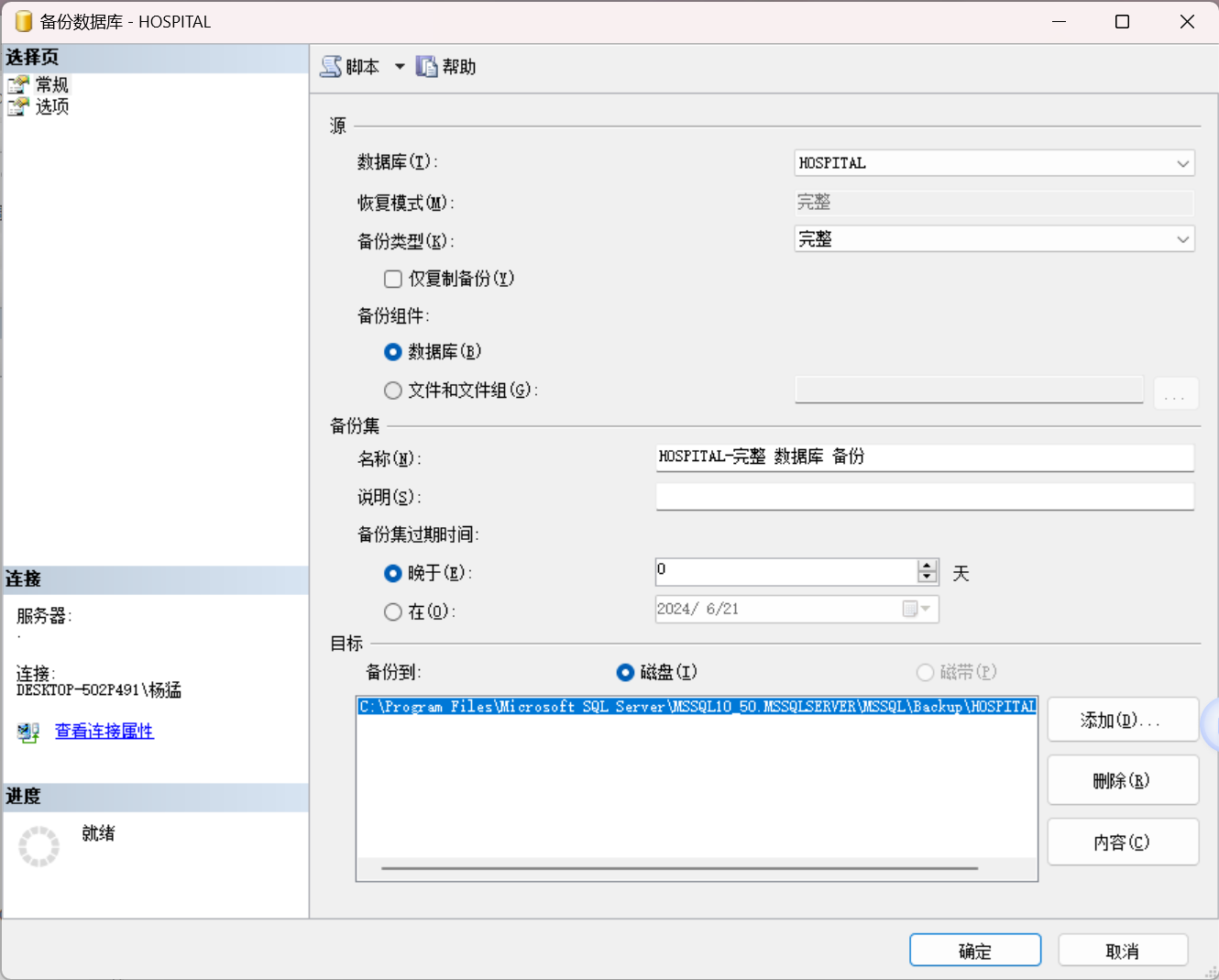
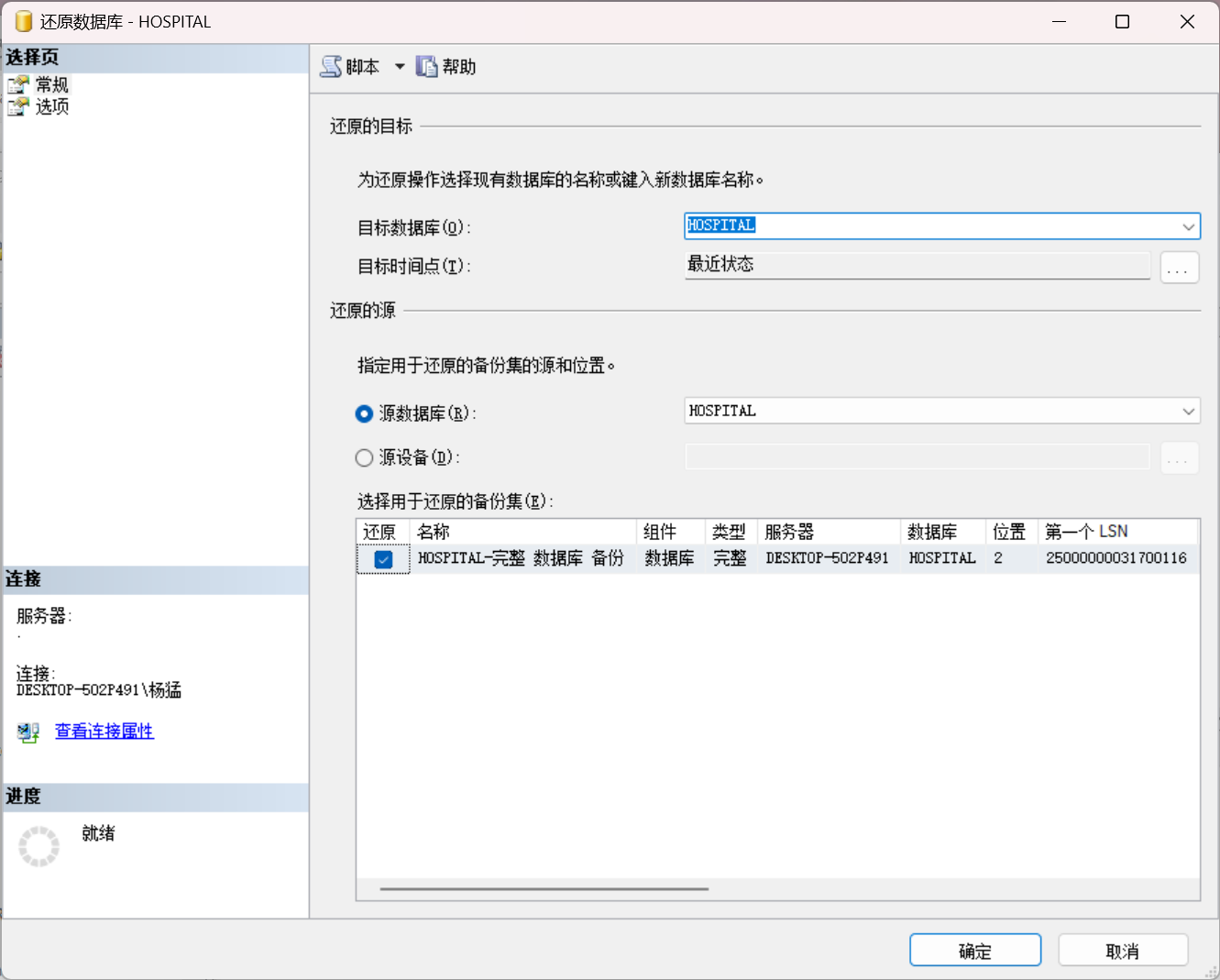


图4.1.7 还原数据库



## 完整性设计

数据库完整性是指数据库数据的正确性和相容性。

1. **实体完整性**

实体完整性是指关系中的每个元组都是可识别的和唯一的，指的是关系数据库中所有的表都必须有主键，而且表中不允许出现无主键值的记录和与主键值相同的记录。以该系统举例为在病人表中病人编号作为主键，确保每个患者都有一个唯一的标识，且不允许为空。或是在医生表中医生编号名作为主键，确保每个医生都有一个唯一的标识，且不允许为空。

1. **参照完整性**

**参照完整性也称引用完整性，**现实世界中的实体之间往往存在着某种联系，在关系模型中，实体以及实体之间的联系都是用关系来表示的，这样就自然存在着关系与关系之间的引用，参照完整性就是描述实体之间的联系的、一般是指多个实体或关系之间的关联关系。以本系统举例为药品表中包含患者ID和医生ID作为外键，分别引用患者表和医生表的主键。在插入或更新处方时，必须确保处方中的患者ID和医生ID在相应的表中存在。就诊药单中包含患者ID和医生ID作为外键，同样需要确保引用的有效性。

1. **用户自定义完整性**

用户自定完整性是指某一具体应用涉及的数据必须满足的语义要求，是针对某一具体应用领域定义的数据约束条件。实际上就是指明关系中属性的取值范围，防止属性的值与应用语义矛盾。在设计关系模型时应提供定义和检验这类完整性的机制，以便用统一的系统方法处理它们，而不要由应用程序承担这一功能。

1. **域完整性**

域是数据库中的一个重要概念。一般地，域是一组具有相同数据类型的值的集合。在设计数据库时，可以用CREATE DOMAIN语句建立一个域并指明域应该满足的完整性约束，然后就可以用域来定义属性。

1. **索引设计**

为了提高查询性能，可以在经常用于搜索、排序和连接的字段上创建索引。例如，在患者表和医生表的姓名字段上创建索引，以便快速查找特定姓名的患者或医生。

# 程序开发与设计

## UI界面设计（null）

## 程序开发、设计与完善

### 程序测试—见现场演示

### 程序完善

1. **功能性完善**

一个程序功能性完善的体现首先在于程序能够精确地执行其设计的任务，满足用户的核心需求，其中包括程序的主要功能点都能按照预期工作，没有遗漏或错误。其次，一个功能完善的程序应该具备一定的灵活性和适应性，能够处理各种输入和情况，即使在非标准或边缘情况下也能给出合理的响应。

1. **性能优秀**

一个设计优秀的程序应该具有高效的性能，包括快速的响应时间和较低的资源消耗，体现在对系统资源的占用合理，能够快速响应用户的操作。

1. **稳定性完善**

一个程序的稳定性完善体现在各种条件下都能稳定运行，不容易出现崩溃或数据丢失。一个稳定性完善的程序具有较好的鲁棒性和兼容性。

# 总结与收获

## 总结（null）

## 团队成员的心得与收获（null）

参考文献:

[1]. 廖先珍与唐续国, 国内医院信息管理系统的应用现状及发展趋势. 医学信息, 2005(08): 第873-875页.

[2]. 袁翰超, 大数据背景下医院信息管理系统应用策略——评《信息分析方法及医学应用》. 科技管理研究, 2023. 43(19): 第264页.

[3]. 胡瑞熙, 探究医院管理中医疗管理信息化建设的作用. 中国管理信息化, 2023. 26(19): 第150-153页.

[4]. 李包罗与许燕, 医院信息系统简介. 中国护理管理, 2009. 9(01): 第77-79页.