中山大學

SUN YAT-SEN UNIVERSITY

数据库系统实验大作业报告

社区医院门诊管理系统的设计与实现

姓 名　韩一鸣 宋彧实 郭知雨

学 号　23366023 23366051 23366020

学 院　网络空间安全学院

专 业　网络空间安全

2026 年 1 月 14 日

摘　要

随着人民对美好生活需求的日渐增长，日常医疗健康环节成为人民生活保障不可或缺的一环，优化社区医院全流程诊疗服务成为提高医疗系统效率的关键。

本文面向“门诊全流程诊疗业务回环”构建一套社区医院门诊管理系统，采用 B/S 架构：前端基于 Vue3 与 Element Plus 实现多角色业务页面，后端基于 Flask 提供 REST API 并结合 JWT 实现身份认证与基于角色的授权控制，数据库采用 MySQL 8.0（InnoDB）并通过外键、唯一约束、检查约束与索引策略保障数据完整性与关键一致性。系统功能涵盖预约挂号、到院登记/签到、就诊状态流转、费用结算与经营统计等环节，避免了传统人工或分散式登记方式容易造成信息割裂、状态不可追溯与数据一致性风险。

系统围绕**患者在线预约（Appointment）**与**患者就诊（Visit）**两条主线设计状态机与事务边界，提供患者端预约与查询、前台端登记/签到/收费与报表、管理员端排班/人员/诊室维护及统计分析等功能。数据库设计遵循第三范式，构建科室、诊室、排班、员工、患者、账号、账单、收入记录与病历等核心实体关系，并针对手机号、身份证号、状态、时间与日期等高频筛选维度建立索引以支撑业务查询与聚合统计。系统以“数据库约束 + 应用层校验”组合实现一致性控制，并通过典型业务用例验证能够完成从预约到离院结算的全流程闭环。

关键词：门诊管理、数据库设计、B/S 架构、Flask、Vue3、JWT、MySQL

目录

1 引言

随着人民群众对医疗服务的需求日渐增加, 社区医院门诊管理逐渐由传统的小规模, 小频率业务场景转向高频、短周期、强流程的业务场景，典型业务链路由“预约/登记—分诊—就诊—缴费—离院—统计”构成。传统的社区医院信息系统设计缺乏统一的信息系统支撑，常见问题包括：预约信息与到院信息分离导致核验困难；就诊状态更新依赖人工口头传递导致不可追溯；收费记录与统计口径不一致导致对账成本高；关键数据（身份、科室、排班）缺乏约束易产生脏数据。

本文以“社区医院门诊管理系统”为对象，实现了面向多角色的一体化门诊业务闭环，并将关键一致性规则在数据库与应用层共同落地。本文的主要工作包括：梳理业务流程与角色需求；设计 B/S 三层总体架构与模块划分；完成面向实体的数据库概念模型与关系模式设计；实现基于 JWT 的认证授权与基于状态机的流程控制；通过查询与统计接口验证数据库索引与数据结构的可用性。

2 需求调研与系统分析

2.1 业务流程调研

门诊业务可划分为四个阶段：

- 预约挂号阶段：患者提交科室、预计到达时间与联系方式，形成预约记录，等待前台确认或取消。
- 到院登记与分诊阶段：患者到院后由前台核验预约或进行现场登记，并依据排班容量与科室资源分配诊室与医生。
- 就诊与缴费阶段：就诊流程按状态推进，待缴费时生成账单并完成结算。
- 离院与统计阶段：缴费完成后离院，数据进入统计口径，用于收入与就诊量分析。

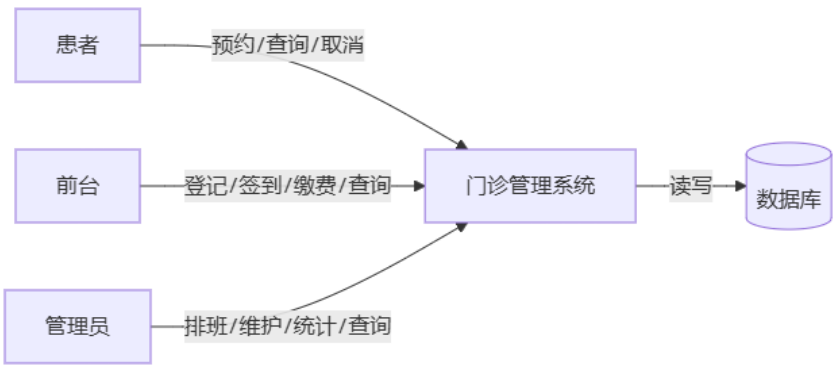

图 1: 业务流程图

2.2 用户角色与权限

系统定义三类核心角色：**患者（patient）、前台（receptionist）与管理员（admin）**。患者端侧重于预约挂号信息（包含预约科室、预约信息、患者个人信息等）的提交、查询与取消；前台端负责登记就诊、患者签到、就诊状态维护、缴费与报表信息的统计提交；管理员侧负责基础数据维护与统计分析。权限控制采用基于角色的访问控制（RBAC），后端在接口层强制校验。

在前端方面分别针对不同的用户角色设计界面，对接三方不同需求，通过账号在后端的校验，将用户导入不同的前端界面，实现功能的精准对接。

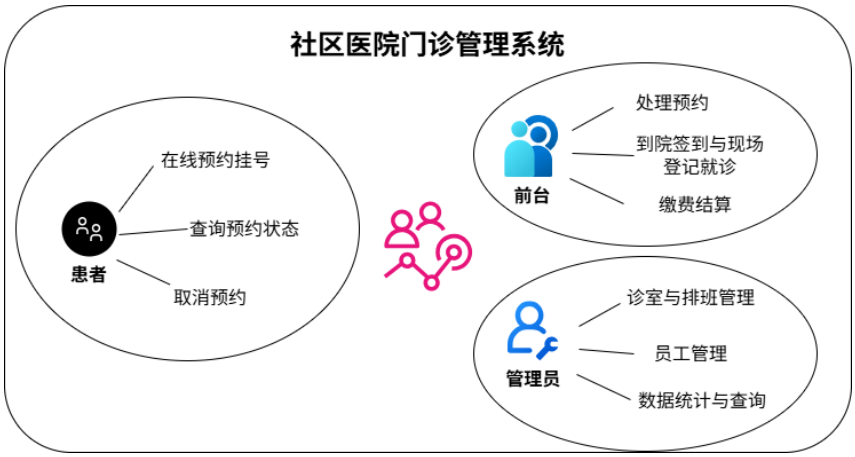

图 2: 系统用户图

2.3 核心业务状态与一致性需求

2.3.1 预约状态机

对于患者端，设定患者的预约状态机，预约状态集合为：待确认、已确认、已完成、已取消。关键的状态转移包括：待确认 → 已确认/已取消，已确认 → 已取消，到院签到后待确认/已确认 → 已完成。完成或取消后的预约不再流转。

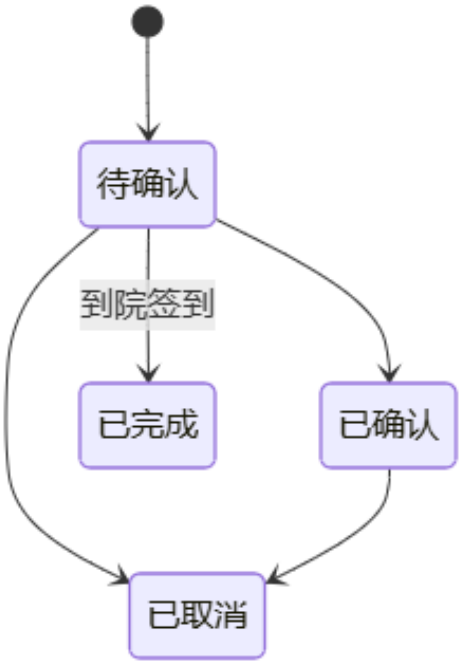

图 3: 预约状态机图示

2.3.2 就诊状态机

患者预约完成后，在前台端可以提交预约相关的信息，将对应的看病预约由预约状态转入就诊状态。就诊状态集合为：候诊中、就诊中、待缴费、已离院，并严格单向流转。本系统通过接口级校验保证不出现越级或回退状态。

图 4: 就诊状态机图示

2.3.3 排班容量与结算恒等

管理员端对医疗人员进行相关排班时，对排班的容量进行核验，排班容量需满足 $current\_patients \leq max\_patients$，并在分配就诊时进行“占位”更新以避免并发超额。缴费结算需满足：$insurance\_amount \leq total\_amount$ 以及 $total\_amount = insurance\_amount + self\_pay\_amount$，并同时在应用层校验与数据库检查约束中体现。

2.4 功能需求概要

系统功能需求按角色与模块归纳如下：

- 认证与用户：登录、患者注册、获取当前用户信息。
- 患者端：科室列表、创建预约、查询预约、取消预约。
- 前台端：现场登记、预约确认/取消、到院签到（预约转就诊）、就诊状态流转、缴费结算、收费报表与收入记录查询。
- 管理员端：诊室/排班/员工维护，患者/就诊/账单/收入查询，病历维护，收入与就诊量统计。

图 5: 系统功能

2.5 其他需求概要

- 一致性：数据库外键引用设置及索引设置合理；关键状态流转合法；账单金额恒等成立；排班容量不超额；预约与就诊、就诊与账单/病历满足一对一关系约束。
- 安全性：基于 JWT 的鉴权与基于角色的授权；密码采用不可逆哈希存储；接口返回统一错误结构避免泄露敏感信息。
- 可用性：提供脚本化初始化与启动方式；接口返回结构统一，便于前端提示与定位问题。
- 性能：高频筛选字段与日期字段建立索引；统计查询尽量在数据库侧完成聚合计算。

3 系统总体设计

3.1 总体架构

系统采用典型 **B/S 三层结构**：浏览器端负责交互展示；后端 API 负责业务规则与鉴权；数据库负责持久化与一致性约束。系统技术栈为：前端 Vue3 + Element Plus，后端 Flask + SQLAlchemy，数据库 MySQL 8.0（InnoDB）。

图 6: 系统总体架构

3.2 功能模块划分

3.2.1 前端模块

前端按角色划分页面与路由：公共登录/注册页面；患者端预约页面；前台端登记、预约处理、就诊与缴费页面；管理员端诊室、排班、员工、患者/就诊/账单查询与统计页面。前端将 JWT 与用户信息存储于本地，并在路由守卫中完成登录态校验与角色访问控制。

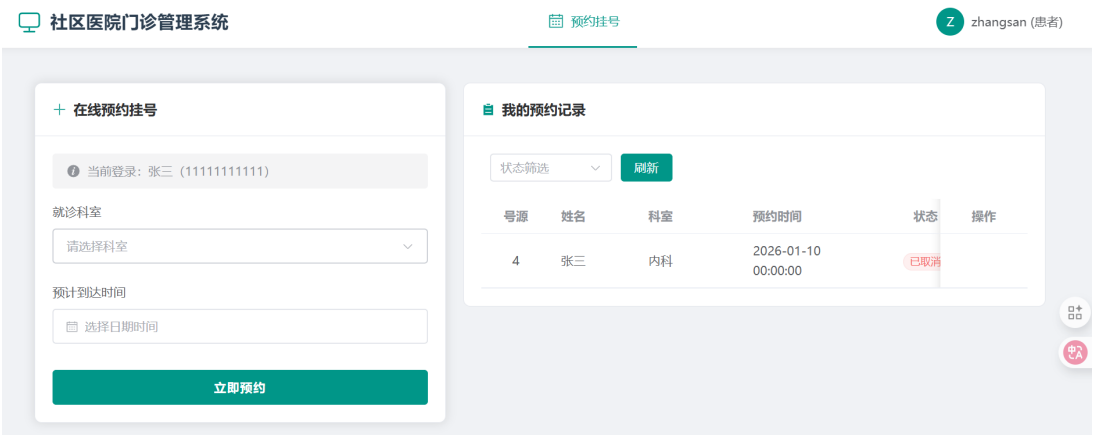

图 7: 患者端页面

图 8: 前台端页面

图 9: 管理端页面

3.2.2 后端模块

后端采用按角色划分的蓝图（Blueprint）组织接口：认证模块提供登录、注册与用户信息；患者模块提供科室查询与预约管理；前台模块提供登记、签到、状态流转、缴费与报表；管理员模块提供诊室/排班/员工维护、病历维护与统计查询。后端统一返回 JSON 结构，并对错误进行统一封装便于前端展示。

4 数据库设计

4.1 设计目标与原则

数据库设计目标包括：支撑门诊闭环业务；保障数据一致性与完整性；兼顾查询性能与统计需求。设计原则包括面向实体建模、规范化（第三范式）与可扩展性。数据库使用 utf8mb4 字符集与 InnoDB 存储引擎。

4.2 概念结构设计（E-R）

系统核心实体包含：科室（Department）、诊室（Room）、排班（Schedule）、员工（Employee）、系统账号（SysUser）、患者（Patient）、预约（Appointment）、就诊（Visit）、账单（Bill）、收入记录（IncomeRecord）、病历（MedicalRecord）与患者账号映射（PatientUser）。其主要关系包括：科室与诊室/员工（1:N），诊室与排班（1:N），预约到院签到后生成就诊（0/1:1），就诊与账单/病历（1:1），账单与收入记录（可扩展为 1:N）。

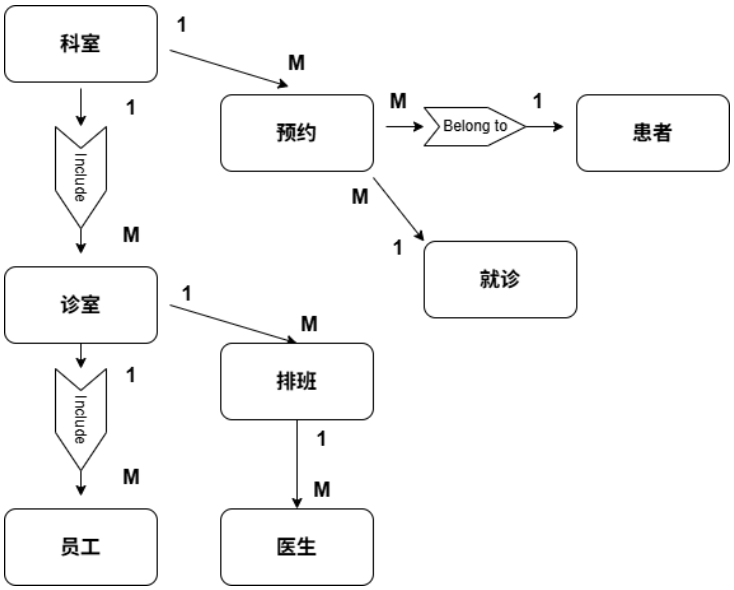

图 10: 系统 E-R 图

4.3 逻辑结构设计

本小节给出关键数据表部分字段与约束的节选说明（以实际建表脚本为准）。为便于阅读，表格中“约束”字段使用简写：**PK 主键、FK 外键、AI 自增、NN 非空、UQ 唯一、CK 检查约束**。

字段	类型	约束	说明
dept_id	INT	PK, AI	科室编号
dept_name	VARCHAR(50)	NN, UQ	科室名称
description	TEXT		科室描述
created_at	TIMESTAMP	NN	创建时间

表 1: department（科室表）

字段	类型	约束	说明
room_id	INT	PK, AI	诊室 ID
room_number	VARCHAR(20)	NN, UQ	诊室编号
dept_id	INT	NN, FK	所属科室
status	ENUM	NN	启用状态
created_at	TIMESTAMP	NN	创建时间

表 2: room（诊室表）

字段	类型	约束	说明
schedule_id	INT	PK, AI	排班 ID
room_id	INT	NN, FK	诊室
doctor_id	VARCHAR(20)	NN, FK	医生工号
work_date	DATE	NN	日期（索引）
time_slot	ENUM	NN	时间段（上午/下午/全天）
max_patients	INT	NN, CK	最大接诊数
current_patients	INT	NN, CK	已占用数

表 3: schedule（排班表）

字段	类型	约束	说明
emp_id	VARCHAR(20)	PK	工号
name	VARCHAR(50)	NN	姓名
gender	ENUM	NN	性别
phone	VARCHAR(20)		电话
position	ENUM	NN	岗位（医生/护士/前台/管理员）
dept_id	INT	FK	所属科室（可空）
status	ENUM	NN	在职状态
created_at	TIMESTAMP	NN	创建时间

表 4: employee（员工表）

字段	类型	约束	说明
patient_id	INT	PK, AI	患者 ID
name	VARCHAR(50)	NN	姓名
gender	ENUM		性别
id_card	VARCHAR(18)	UQ	身份证号（可空但唯一）
phone	VARCHAR(20)	NN	手机号
created_at	TIMESTAMP	NN	创建时间

表 5: patient（患者表）

字段	类型	约束	说明
appt_id	INT	PK, AI	预约 ID
patient_name	VARCHAR(50)	NN	预约姓名（快照）
phone	VARCHAR(20)	NN	联系电话
dept_id	INT	NN, FK	预约科室
expected_time	DATETIME	NN	预计到达时间
status	ENUM	NN	预约状态
patient_id	INT	FK	关联患者（可空）
created_at	TIMESTAMP	NN	创建时间

表 6: appointment（预约表）

字段	类型	约束	说明
visit_id	INT	PK, AI	就诊 ID
appt_id	INT	UQ, FK	关联预约（可空，且唯一）
patient_id	INT	NN, FK	患者
dept_id	INT	NN, FK	科室
schedule_id	INT	FK	分配排班（可空）
status	ENUM	NN	就诊状态
check_in_time	TIMESTAMP		到院/登记时间
created_at	TIMESTAMP	NN	创建时间

表 7: visit（就诊记录表）

字段	类型	约束	说明
bill_id	INT	PK, AI	账单 ID
visit_id	INT	NN, UQ, FK	关联就诊（1:1）
total_amount	DECIMAL(10,2)	NN, CK	总费用
insurance_amount	DECIMAL(10,2)	NN, CK	医保金额
self_pay_amount	DECIMAL(10,2)	NN, CK	自费金额
pay_status	ENUM	NN	支付状态
pay_time	TIMESTAMP		支付时间
created_at	TIMESTAMP	NN	创建时间

表 8: bill（账单表，节选）

4.4 一致性约束与索引策略

一致性约束包括：外键约束保证引用完整；唯一性约束防止关键字段重复（如账号名、科室名、身份证号、预约生成就诊的一对一关系等）；检查约束覆盖排班容量与金额恒等。索引策略优先覆盖高频检索条件与报表聚合维度，如手机号、身份证号、状态、到院时间、排班日期与收入日期。

4.5 账号与统计相关表

下列给出部分账号信息与收入信息统计相关数据表的部分字段：

字段	类型	约束	说明
user_id	INT	PK, AI	账号 ID
username	VARCHAR(50)	NN, UQ	用户名
password_hash	VARCHAR(255)	NN	密码哈希
role	ENUM	NN	角色（patient/receptionist/admin）
emp_id	VARCHAR(20)	FK	绑定员工（可空）
is_active	TINYINT	NN	是否启用
created_at	TIMESTAMP	NN	创建时间

表 9: sys_user（系统账号表）

字段	类型	约束	说明
user_id	INT	PK, FK	患者账号
patient_id	INT	NN, UQ, FK	绑定患者（唯一）
created_at	TIMESTAMP	NN	创建时间

表 10: patient_user（患者账号映射表）

字段	类型	约束	说明
record_id	INT	PK, AI	收入记录 ID
bill_id	INT	NN, FK	关联账单
dept_id	INT	NN, FK	科室
doctor_id	VARCHAR(20)	FK	医生（可空）
amount	DECIMAL(10,2)	NN	金额
record_date	DATE	NN	记账日期（索引）
created_at	TIMESTAMP	NN	创建时间

表 11: income_record（收入记录表）

字段	类型	约束	说明
record_id	INT	PK, AI	病历 ID
visit_id	INT	NN, UQ, FK	关联就诊（1:1）
diagnosis	TEXT		诊断
treatment	TEXT		处置/治疗
prescription	TEXT		处方
note	TEXT		备注
updated_at	TIMESTAMP		更新时间

表 12: medical_record（病历表）

5 系统实现

5.1 工程结构

系统工程由前后端与数据库脚本组成。重要目录结构及文件概览如下:

```
backend/      # Flask后端（API、模型、工具）
frontend/     # Vue3前端（页面、路由、API封装）
database/     # MySQL建表与初始化数据脚本
figures/      # 报告与文档用图（架构图、用例图、ER图）
docs/         # 项目说明文档
start.ps1/    # 项目启动脚本
```

5.2 前端实现要点

前端采用组件化开发与基于路由的页面组织方式，核心实现要点包括：统一 HTTP 请求封装（携带 JWT 并处理过期）；路由守卫根据用户角色进行访问控制与重定向；不同角色的业务页面分别覆盖预约、登记、收费与统计等场景。部分关键代码实现:

Http 请求封装

```
http.interceptors.request.use((config) => {
  const token = getToken()
  if (token) {
    config.headers = config.headers || {}
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})

http.interceptors.response.use(
  (resp) => resp,
  (err) => {
    const status = err?.response?.status
    const body = err?.response?.data

    const message = body?.error?.message || body?.msg || err?.message || '
        请求失败'
    const e = new Error(message)
    e.code = body?.error?.code || (status ? `http_${status}` : '
        network_error')
    e.status = status
    e.details = body?.error?.details

    if (status === 401) clearAuth()
    throw e
```

```
  },
)
```

登录状态存储

```
export function setAuth(token, user) {
  localStorage.setItem(TOKEN_KEY, token)
  localStorage.setItem(USER_KEY, JSON.stringify(user))
  window.dispatchEvent(new Event('auth-changed'))
}

export function clearAuth() {
  localStorage.removeItem(TOKEN_KEY)
  localStorage.removeItem(USER_KEY)
  window.dispatchEvent(new Event('auth-changed'))
}
```

路由守卫跳转

```
router.beforeEach((to) => {
  const token = getToken()
  const user = getUser()

  if (to.path === '/') {
    if (!token || !user) return { path: '/login' }
    return { path: defaultPathForUser(user) }
  }

  if (to.meta?.requiresAuth && !token) {
    clearAuth()
    return { path: '/login', query: { redirect: to.fullPath } }
  }

  if (to.meta?.roles?.length) {
    if (!user || !to.meta.roles.includes(user.role)) {
      clearAuth()
      return { path: '/login', query: { redirect: to.fullPath } }
    }
  }
})
```

5.3 后端实现要点

后端使用 Flask 提供 REST API，并以蓝图方式按角色组织路由。数据层使用 SQLAlchemy 映射实体模型，并在关键接口中实现事务语义以保证跨表写入一致。错误处理与响应返回统一封装，便于前端对成功与失败进行一致处理。后端部分代码实现：

Flask 初始化与错误封装

```
def register_blueprints(app: Flask) -> None:
    app.register_blueprint(auth_bp)
    app.register_blueprint(patient_bp)
    app.register_blueprint(receptionist_bp)
    app.register_blueprint(admin_bp)

@app.errorhandler(APIError)
def _api_error(err: APIError):
    db.session.rollback()
    return error(err.message, code=err.code, status=err.status, details=err
        .details)
```

5.4 认证与授权实现

系统采用 JWT 进行认证。用户登录成功后由后端签发令牌，前端在后续请求中携带令牌访问受限资源。后端在统一鉴权组件中完成：令牌解析、用户有效性校验与角色授权校验，从而保证越权访问被拒绝。前端在路由守卫中根据角色元信息进行页面访问控制与默认工作台跳转，降低误操作概率。

JWT 认证 (基于用户角色授权)

```
token = create_access_token(identity=str(user.user_id), additional_claims={
    "role": user.role})

def roles_required(*roles: str):
    def decorator(fn):
        def wrapper(*args, **kwargs):
            verify_jwt_in_request()
            if current_user.role not in roles:
                raise APIError("Forbidden", code="forbidden", status=403)
            return fn(*args, **kwargs)
        return wrapper
    return decorator
```

5.5 预约管理实现

患者可创建预约并查询个人预约列表，未完成预约允许取消。前台可对预约执行确认或取消，并在患者到院后通过签到流程将预约转化为就诊记录。该过程属于跨表写入操作，后端将“预约状态更新、排班占位与就诊创建”放入同一事务语义中，保证任一步失败时整体回滚。

预约状态机设置

```
_APPOINTMENT_TRANSITIONS = {
    "待确认": {"已确认", "已取消"},
    "已确认": {"已取消"},
    "已完成": set(),
    "已取消": set(),
}
```

5.6 登记、就诊与收费实现

前台支持两条路径创建就诊：预约签到转就诊与现场登记直接创建就诊。就诊状态严格按照状态机单向推进。待缴费阶段由前台录入金额信息，后端进行恒等校验并生成账单，同时写入收入记录以支撑后续报表统计。部分代码如下：

预约转就诊及事务语义

```
schedule = _reserve_schedule(dept_id=appt.dept_id, target_dt=appt.
    expected_time)

visit = Visit(
    patient_id=patient.patient_id,
    room_id=schedule.room_id,
    doctor_id=schedule.doctor_id,
    appt_id=appt.appt_id,
    status="候诊中",
)
db.session.add(visit)
appt.status = "已完成"

db.session.flush()
db.session.commit()

# 排班占位
updated = (
    Schedule.query.filter(Schedule.schedule_id == schedule_id)
    .filter(Schedule.current_patients < Schedule.max_patients)
    .update({Schedule.current_patients: Schedule.current_patients + 1},
        synchronize_session=False)
)

# 现场登记
schedule = _reserve_schedule(dept_id=int(dept_id), target_dt=target_dt)
patient = _get_or_create_patient(name=name, phone=phone, gender=gender,
    id_card=id_card)
visit = Visit(patient_id=patient.patient_id, room_id=schedule.room_id,
```

```
    doctor_id=schedule.doctor_id, appt_id=None, status="候诊中")
```

5.7 统计查询实现

管理员与前台可按条件查询账单与收入记录，并支持按日期、科室或医生维度进行聚合统计。该模块的关键在于数据库索引与字段设计：通过在预约时间、到院时间、收入日期等字段建立索引，提高筛选与聚合计算效率。

收入就诊统计查询

```
rows = (
    q.with_entities(IncomeRecord.record_date, func.sum(IncomeRecord.amount)
        , func.count(IncomeRecord.record_id))
    .group_by(IncomeRecord.record_date)
    .order_by(IncomeRecord.record_date.asc())
    .all()
)
```

5.8 统计查询示例

统计模块常见查询包括按日期聚合收入、按科室统计就诊量等。以下给出一个按日期汇总收入的 SQL 示例：

```
SELECT
  record_date,
  SUM(amount) AS total_income
FROM income_record
WHERE record_date BETWEEN '2025-12-01' AND '2025-12-31'
GROUP BY record_date
ORDER BY record_date;
```

5.9 管理员端数据维护

通过管理员端对诊室、员工排班、员工信息进行基本维护，并通过管理端查询对应的信息。

基本数据维护

```
room_id = payload.get("room_id")
doctor_id = (payload.get("doctor_id") or "").strip()
work_date = parse_date(payload.get("work_date") or "")
time_slot = (payload.get("time_slot") or "").strip()

if not room_id or not doctor_id or not time_slot:
```

```
    raise APIError("room_id, doctor_id, work_date, time_slot are required",
        code="validation_error", status=400)
if time_slot not in ("上午", "下午", "全天"):
    raise APIError("Invalid time_slot", code="validation_error", status
        =400)
if Room.query.get(room_id) is None:
    raise APIError("Invalid room_id", code="validation_error", status=400)
if Employee.query.get(doctor_id) is None:
    raise APIError("Invalid doctor_id", code="validation_error", status
        =400)
```

管理端数据查询

```
    # 关键查询限定约束
    if start_time:
    q = q.filter(Visit.check_in_time >= parse_datetime(start_time))
elif start_date:
    start = parse_date(start_date)
    q = q.filter(Visit.check_in_time >= datetime.combine(start, time.min))

if end_time:
    q = q.filter(Visit.check_in_time <= parse_datetime(end_time))
elif end_date:
    end = parse_date(end_date)
    q = q.filter(Visit.check_in_time <= datetime.combine(end, time.max))

    # 报表数据
    items = q.offset(offset).limit(limit).all()
return ok(
    {
        "total": total,
        "limit": limit,
        "offset": offset,
        "items": [{"bill": b.to_dict(), "visit": b.visit.to_dict() if b.
            visit else None} for b in items],
    }
)
```

数据库部分关键索引

```
    CONSTRAINT ck_schedule_capacity CHECK (current_patients <= max_patients
        ),
    INDEX idx_schedule_work_date (work_date);

    INDEX idx_appt_phone (phone),
    INDEX idx_appt_expected_time (expected_time);
```

```
    INDEX idx_visit_status (status),
    INDEX idx_visit_check_in_time (check_in_time);

    CONSTRAINT ck_bill_amount CHECK (total_amount = insurance_amount +
        self_pay_amount);

    INDEX idx_income_record_date (record_date);
```

6 接口设计

系统接口按角色划分 URL 前缀:/api/auth、/api/patient、/api/receptionist 与/api/admin。表13至表16给出主要接口的节选列表。

方法	路径	说明
POST	/api/auth/login	登录并返回 JWT
POST	/api/auth/register	患者注册（创建患者并绑定账号）
GET	/api/auth/profile	获取当前用户信息
POST	/api/auth/logout	登出（前端清理令牌）

表 13: 认证接口

方法	路径	说明
GET	/api/patient/departments	科室列表
POST	/api/patient/appointments	创建预约
GET	/api/patient/appointments/query	查询我的预约
DELETE	/api/patient/appointments/{appt_id}	取消预约

表 14: 患者接口

方法	路径	说明
GET	/api/receptionist/appointments	预约列表
PUT	/api/receptionist/appointments/{appt_id}/status	更新预约状态
POST	/api/receptionist/checkin/{appt_id}	到院签到（预约转就诊）
POST	/api/receptionist/register	现场登记就诊
GET	/api/receptionist/visits	就诊列表
PUT	/api/receptionist/visits/{visit_id}/status	就诊状态流转
POST	/api/receptionist/payment/{visit_id}	缴费结算（账单 + 收入记录）

表 15: 前台接口

方法	路径	说明
GET/POST	`/api/admin/rooms`	诊室查询/新增
GET/POST	`/api/admin/schedules`	排班查询/新增
GET/POST	`/api/admin/employees`	员工查询/新增
GET	`/api/admin/statistics/income`	收入统计
GET	`/api/admin/statistics/visits`	就诊统计
GET/PUT	`/api/admin/visits/{visit_id}/medical-record`	病历查询/更新

表 16: 管理员接口（节选）

7 测试与评估

7.1 测试目标与方法

测试目标包括：核心业务闭环正确性；权限控制正确性；一致性约束有效性；典型查询与统计接口可用性。测试方法采用业务用例驱动：对预约、签到、就诊状态流转、缴费结算与报表查询等关键流程执行端到端验证，并对异常输入与越权访问进行负向测试。

7.2 部分测试用例

用例	操作步骤（概要）	期望结果
预约创建	患者选择科室与到达时间提交预约	生成预约，状态为待确认
预约确认	前台对预约执行确认	预约状态更新为已确认
到院签到	前台核验后执行签到	创建就诊并置预约为已完成
状态流转	前台按顺序更新就诊状态	候诊中 → 就诊中 → 待缴费 → 已离院
缴费结算	录入金额并提交结算	账单与收入记录写入且金额恒等成立

表 17: 测试用例

8 部署与运行

系统支持使用脚本进行数据库初始化与前后端启动。数据库初始化使用建表脚本与基础数据脚本写入科室、诊室、员工与默认账号等；后端通过环境变量配置数据库连接与 JWT 密钥；前端通过开发代理转发/api 请求至后端服务。演示模式下可启用自动建表与种子数据以降低联调门槛；按作业要求提交与验收时建议关闭自动建表并使用 SQL 脚本进行严格初始化。

8.1 启动步骤（示例）

分布启动

```
1) 初始化MySQL并导入database/schema.sql与database/init_data.sql
2) 配置后端环境变量（DATABASE_URL、JWT_SECRET_KEY等）
3) 启动后端：python backend/run.py
4) 启动前端：cd frontend && npm install && npm run dev
```

命令行一行启动

```
.\start.ps1 -MySqlUser root -MySqlHost 127.0.0.1 -MySqlPort 3306
```

其余具体实现操作可以参照项目内文档。

9 总结与展望

本系统完成了从社区医院门诊管理系统的需求分析、系统总体设计、数据库设计与系统具体实现，提出了面向多角色的全流程诊疗业务闭环方案。系统在数据层通过外键、唯一与检查约束保证实体关系与关键恒等式，在应用层通过状态机校验与事务边界降低跨表写入不一致风险；在功能层实现预约、登记、就诊、缴费与统计查询等核心能力。同时系统功能可进一步扩展：细化排班维度（按小时粒度）；增加更完整的收费项目与处方药品管理；引入审计日志与操作追踪；在并发场景下进一步强化排班占位与结算流程的并发控制策略。

10 致谢

项目团队成员分工：

- 韩一鸣 后端开发及数据库结构设计
- 宋彧实 前端设计及后端优化
- 郭知雨 前后端测试

项目 Github 地址： https://github.com/yiming-qing/Community-Hospital-Outpatient-Management-System

欢迎各位用户体验并提出改进建议！

参考文献

[1] Pallets Projects. Flask Documentation. https://flask.palletsprojects.com/.

[2] SQLAlchemy. SQLAlchemy Documentation. https://docs.sqlalchemy.org/.

[3] Vue.js. Vue 3 Documentation. https://vuejs.org/.

[4] Oracle. MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/.

[5] IETF. JSON Web Token (JWT), RFC 7519. https://www.rfc-editor.org/rfc/rfc7519.