# 資料庫管理（113-1）

iclinic

# 專案介紹

# 「iclinic」是一款診所與患者的多邊平臺，實質上可以視為診所的資訊系統解決方案。

| 市場痛點 | 診所整合平臺 |
| --- | --- |
| **小診所資源不足** | **多邊平臺效應** |
| 缺乏自行開發應用程式的能力或必要性 | 診所多 → 患者效益大<br>患者多 → 診所效益大 |
| **平台效益不足** | **潛在業務** |
| 患者單獨下載單一診所App 效益低 | 診所廣告、公開評價、顧問服務、交叉銷售 |

## 「iclinic」是一款診所與患者的多邊平臺，實質上可以視為診所的資訊系統解決方案。

### 平臺主要功能

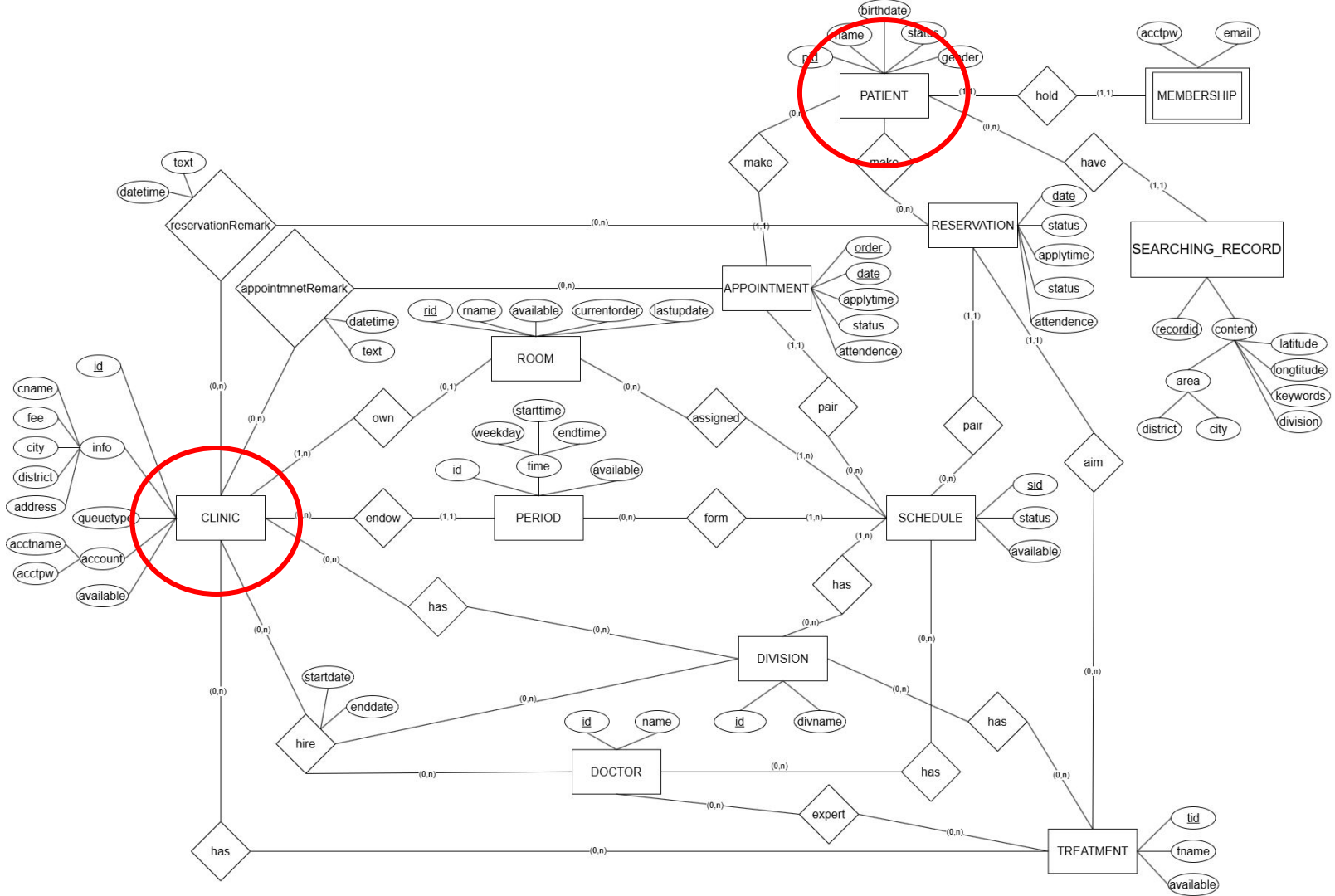
提供診所、病人掛號與預約

提供診所、病人查詢資料

提供診所管理系統，包含診間、醫生等等

# demo

# 系統設計


birthdate
name
status
pid
gender
PATIENT
hold
acctpw
email
MEMBERSHIP
make
make
have
text
datetime
reservationRemark
appointmnetRemark
datetime
text
RESERVATION
date
status
applytime
status
attendence
SEARCHING_RECORD
recordid
content
latitude
longtitude
keywords
division
area
district
city
APPOINTMENT
order
date
applytime
status
attendence
rid
rname
available
currentorder
lastupdate
ROOM
id
cname
fee
city
district
address
info
queuetype
acctname
account
acctpw
available
CLINIC
own
assigned
pair
pair
aim
starttime
weekday
endtime
id
time
available
endow
PERIOD
form
SCHEDULE
sid
status
available
has
has
DIVISION
startdate
enddate
hire
id
name
DOCTOR
id
divname
has
has
expert
has
TREATMENT
tid
tname
available

# 資料蒐集與生成

**CLINIC（診所）**

- 資料來源為健保特約醫事機構的公開資料集。
- 使用 `pandas` 模組讀取資料，並透過自定義函式`parse_address_simple` 將地址解析為「縣市」、「鄉鎮市區」及「詳細地址」。
- 其他欄位（如診所代碼、名稱、排隊模式）使用隨機生成器補足。

**DIVISION（科別）**

- 為靜態資料，根據醫療機構標準分類定義。
- 包含 44 種科別（如內科、外科、骨科等）。
- 為每個科別分配唯一編號（如D01 至 D44）及名稱後插入資料庫。

**DOCTOR（醫生）與 HIRE（聘用）**

- 使用 `Faker` 模組生成隨機姓名。
- DOCTOR 表與現有的診所和科別表建立關聯。
- HIRE 表記錄醫生的診所與科別，以及隨機生成的起始日期和其他欄位。

# 資料蒐集與生成

**PATIENT（病患）**

- 完全使用 Faker 模組生成，包括病患編號、姓名、性別及生日。
- 資料生成過程保證唯一性，並透過資料庫會話（Session）插入資料庫。

**PERIOD（診療時段）與 ROOM（診間）**

- 基於診所資料動態生成。
- PERIOD 包含診所代碼、工作日及診療的開始與結束時間。
- ROOM 為每間診所分配若干診療室，生成診間代碼與名稱後與診所表建立關聯。

**生成與插入的方式**

- 所有資料表的生成均使用pandas 和 Faker 等工具輔助，結合自定義的資料處理函式。
- 生成的資料通過 SQLAlchemy 的資料庫會話統一插入，確保資料完整性和一致性。

# 搜尋掛號記錄效能優化 -> Indexing

```
SELECT *
FROM APPOINTMENT
WHERE 1=1
    AND (pid = 'PATIENT_ID' OR 'PATIENT_ID' IS NULL)
    AND (sid = 'SCHEDULE_ID' OR 'SCHEDULE_ID' IS NULL)
    AND (date = '2024-12-20' OR '2024-12-20' IS NULL)
    AND (appointment.order = 1 OR 1 IS NULL)
    AND (applytime = '2024-12-19 12:00:00' OR '2024-12-19 12:00:00' IS NULL)
    AND (status = 'CONFIRMED' OR 'CONFIRMED' IS NULL)
    AND (attendance = 'YES' OR 'YES' IS NULL);
  return go(f, seed, [])
}
```

# 搜尋掛號記錄效能優化 -> Indexing

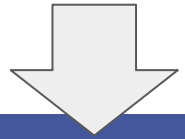
| Before Indexing-pid, sid | | After Indexing-pid, sid |
| --- | --- | --- |
| Avg: 5.14s |  | Avg: 5.14s |

| Before Indexing-date | | Before Indexing-date |
| --- | --- | --- |
| Avg: 17.80s |  | Avg: 14.46s |

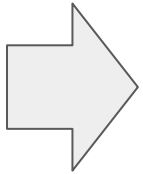
# 診所僱新醫生 ->交易管理

醫生可以在很多診所就職

不是每間診所都會加入平臺



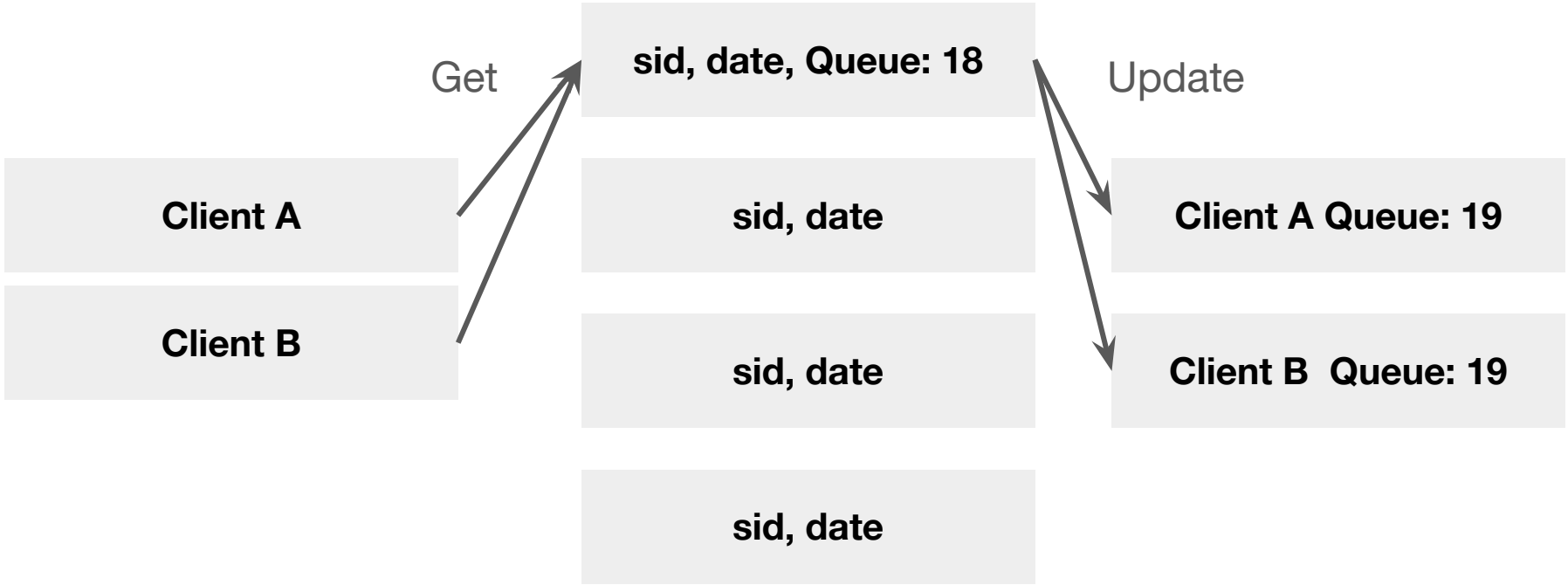
診所不一定知道新醫生有沒有在資料庫 內



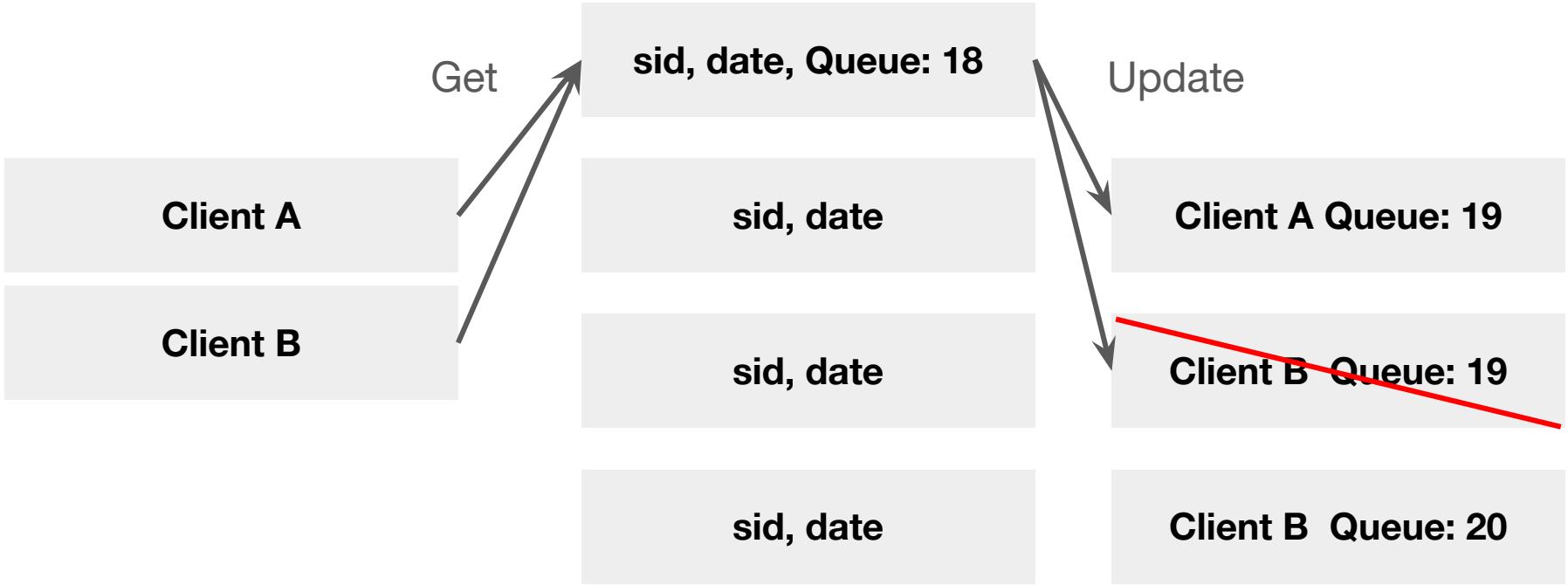
有可能需要同時創建醫生
以及僱用記錄

```python
def create_or_update_hire(db: Session, data: DoctorAndHireCreate):
    if not id_validator(data.docid):
        raise HTTPException(status_code=400, detail="Invalid doctor id")

    try:
        # 开启事务
        with db.begin():
            # Step 1: Ensure the doctor exists
            doctor = get_doctor(db, docid=data.docid)
            if not doctor:
                if not data.docname:
                    raise HTTPException(status_code=400, detail="Doctor name is required")
                create_doctor(db, DoctorCreate(docid=data.docid, docname=data.docname))

            # Step 2: Check if cid and divid exist in Clinicdivision
            clinicdivision = db.query(Clinicdivision).filter_by(cid=data.cid, divid=data.divid).first()
            if not clinicdivision:
                raise HTTPException(status_code=404, detail="Clinicdivision not found")

            # Step 3: Try updating the hire if it exists
            existing_hire = db.query(Hire).filter_by(docid=data.docid, cid=data.cid, divid=data.divid).first()
            if existing_hire:
                return update_hire(
                    db,
                    docid=data.docid,
                    cid=data.cid,
                    divid=data.divid,
                    hire_update=HireUpdate(startdate=data.startdate, enddate=data.enddate)
                )

            # Step 4: Create a new hire if no existing hire
            return create_hire(
                db,
                HireCreate(docid=data.docid, cid=data.cid, divid=data.divid, startdate=data.startdate, enddate=data.enddate)
            )

    except HTTPException as e:
        # 如果是 HTTPException，直接抛出
        raise e
    except SQLAlchemyError as e:
        # 捕获 SQLAlchemy 异常并回滚事务
        db.rollback()
        raise HTTPException(status_code=500, detail=f"Database error: {str(e)}")
    except Exception as e:
        # 捕获其他异常
        db.rollback()
        raise HTTPException(status_code=500, detail=f"Unexpected error: {str(e)}")
```

# 併行控制：掛號預約


Get
sid, date, Queue: 18
Update
Client A
sid, date
Client A Queue: 19
Client B
sid, date
Client B Queue: 19
sid, date

# 併行控制：掛號預約


Get
sid, date, Queue: 18
Update
Client A
sid, date
Client A Queue: 19
Client B
sid, date
Client B Queue: 19
sid, date
Client B Queue: 20

```
    # 解析 command
    try:
        command = json.loads(command)
    except json.JSONDecodeError:
        return {"message": "Invalid JSON format. Please provide valid input."}
    if command['date'] < datetime.datetime.now().strftime('%Y-%m-%d'):
        return {"message": "Invalid date. Please enter a date in the future."}
    # 查询 Schedule
    schedule = db.query(Schedule).filter(Schedule.sid == command['sid']).first()
    if not schedule:
        return {"message": "No such schedule found. Please try again with a valid schedule ID."}

    # 查询 Period
    period = db.query(Period).filter(Period.perid == schedule.perid).first()
    if not period:
        return {"message": "No such period found. Please check the schedule details."}

    patient = db.query(Appointment).filter(Appointment.pid == session['user_id'], Appointment.date == command['date'],
    Appointment.sid == command['sid'], Appointment.status == 'P').first()
    if patient:
        return {"message": "You have a pending appointment on this date."}
    # 解析日期并匹配
    try:
        command['date'] = datetime.datetime.strptime(command['date'], '%Y-%m-%d')
    except ValueError:
        return {"message": "Invalid date format. Please use 'YYYY-MM-DD'."}

    if (period.weekday - 1) != command['date'].weekday():
        return {"message": "The date does not match the schedule's weekday."}

    # 准备数据
    input_data = {
        "pid": session['user_id'],
        "sid": command['sid'],
        "date": command['date'],
        "status": "P",
        "attendance": 0,
        "applytime": datetime.datetime.now()
    }

    # 数据库事务
    try:
        appointments = db.query(Appointment).filter(
            Appointment.date == input_data['date'],
            Appointment.sid == input_data['sid']
        ).with_for_update().all()
        input_data['order'] = len(appointments) + 1

        new_appointment = Appointment(**input_data)
        db.add(new_appointment)
        db.commit()
        db.refresh(new_appointment)
        return {"message": "Appointment created successfully. Your Queue number is {}.".format(input_data['order']), "appointment":
new_appointment}
    except SQLAlchemyError as e:
        db.rollback()
        return {"message": f"Database error: {str(e)}"}
    except Exception as e:
        return {"message": f"Unexpected error: {str(e)}"}
```

# 未來展望

# 我們的資料庫還可以做這些功能

**給病人的功能**

- 使用行政區、經緯度查詢
- 預約看診、療程（指定時間）
- 給予該次看診的診所回饋

**給診所的功能**

- 視覺化管理資訊

**admin**

- 各類SQL Query

# 心得感想

# 共同心得

- 前期定義要先做好！
- 從前端開始似乎比後端更有條理？
- 資料庫設計永遠有想不到的問題！