


数据库PJ 说明文档

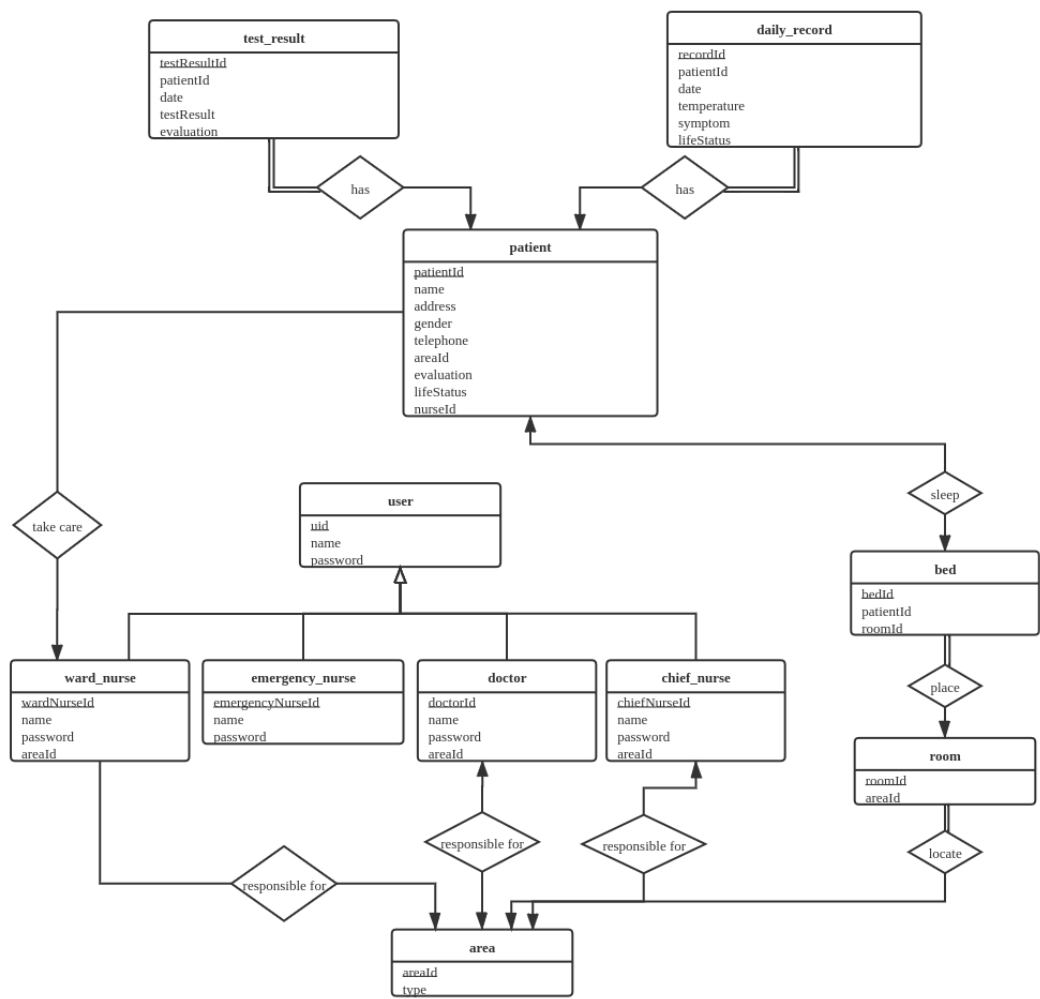
宣子涛 18302010015
戴予淳 18302010047

项目github仓库地址：<https://github.com/FLLIGHT/Database-PJ>

一、项目功能完成情况概述

1. 全部功能 

二、ER图设计



三、数据库表结构和索引定义说明

3.1 PATIENT（病人表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
patientId	病人UUID	VARCHAR	128			主键	✓
name	病人的姓名	VARCHAR	255				
address	病人的地址	VARCHAR	2047				
gender	病人的性别	VARCHAR	63				
telephone	病人的电话	VARCHAR	63				
areald	病人所属治疗区域	INT	11			外键	✓
evaluation	病人的病情评级	INT	11			外键	✓
lifeStatus	病人的生命状态	INT	11			外键	✓
nurseld	照顾病人的病床护士ID	INT	11	✓	0		✓

3.2 TEST_RESULT（核酸检测结果表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
testResultId	核酸检测单ID	INT	11			主键	✓
patientId	病人UUID	VARCHAR	128			外键	✓
date	核酸检测的时间	DATE TIME	/				
testResult	核酸检测结果	VARCHAR	255				
evaluation	病情评级	INT	11			外键	✓

3.3 DAILY_RECORD（每日状态记录）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
recordId	记录的ID	INT	11			主键	✓
patientId	病人UUID	VARCHAR	511			外键	✓
temperature	病人的体温	FLOAT	/				
symptom	病人的症状	VARCHAR	255	✓	NULL		

lifeStatus	病人的生命状态	INT	11			外键	✓
date	记录的时间	DATETIME	/				

3.4 DOCTOR（医生表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
doctorId	医生的ID	INT	11			主键	✓
name	医生的名字	VARCHAR	255				
password	医生的密码	VARCHAR	255				
areald	医生管理的区域	INT	11			外键	✓

3.5 CHIEF_NURSE（护士长表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
chiefNurseId	护士长的ID	INT	11			主键	✓
name	护士长的名字	VARCHAR	255				
password	护士长的密码	VARCHAR	255				
areald	护士长管理的区域	INT	11			外键	✓

3.6 WARD_NURSE（病房护士表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
wardNurseId	病房护士的ID	INT	11			主键	✓
name	病房护士的名字	VARCHAR	255				
password	病房护士的密码	VARCHAR	255				
areald	病房护士所属的区域	INT	11			外键	✓

3.7 EMERGENCY_NURSE（急诊护士表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
emergencyNurseId	急诊护士的ID	INT	11			主键	✓
name	急诊护士的名字	VARCHAR	255				
password	急诊护士的密码	VARCHAR	255				

3.8 MESSAGE（站内信表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
messageId	消息的ID	INT	11			主键	✓
told	接受消息的用户的ID	INT	11				✓
toType	接受消息的用户的类型	INT	11				✓
messageContent	消息的内容	VARCHAR	1024				
alreadyRead	消息的读取状态	INT	11				

3.9 BED（病床表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
bedId	病床的ID	INT	11			主键	✓
patientId	病床上的病人ID	VARCHAR	128	✓	空字符串		
roomId	病床所属病房的ID	INT	11			外键	✓

3.10 ROOM（病房表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
roomId	病房的ID	INT	11			主键	✓
areald	治疗区域的ID	INT	11			外键	✓

3.11 AREA（治疗区域表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
areald	区域的ID	INT	11			主键	✓
type	区域的类型 (5种)	VARCHAR	256				

3.12 EVALUATION（病情评级表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
evaluationId	病情评级的ID	INT	11			主键	✓
evaluation	病情评级的类型 (3种)	VARCHAR	255				

3.13 LIFE_STATUS（生命状态表）

字段名称	字段描述	字段类型	长度	允许空	缺省值	键	索引
lifeStatusId	生命状态的ID	INT	11			主键	✓
lifeStatus	生命状态的类型 (3种)	VARCHAR	255				

三、数据库表设计说明

3.1 病人的ID使用UUID（UUID VS 自增主键）

一般而言，主键可以分为自然键（存储的内容本身可以唯一确定一条记录，如：PERSON的数据表中需要记录身份证号，就可以用身份证号作为主键）和代理键（如ID）。而代理键有两种常见的实现方式：自增主键和UUID。其中，自增主键由DBMS内部实现，当将新记录插入表中时，DBMS都会依次自动生成主键，并且保证该键对表是唯一的。UUID也是唯一的，但这个唯一性是“实际性唯一”(practically unique)而不是“保证性唯一”(guaranteed unique)。

UUID的明显缺点是所需的存储空间明显大于自增主键，而且无法用于排序，也会使调试变得更加困难。而自增主键的缺点是其唯一性是绑定环境/上下文的，在分布式系统中表现不佳。

对于一个简单的不需要用到分布式系统的课程项目来说，自增主键在通常情况下会是更好的选择——它的实现简单（只需在建表时进行设置，之后交给DBMS即可），调试方便，还可以排序。因此，在本项目中，除了病人表以外的所有表（如核酸检测单表，医生表等）的主键都使用的自增主键。而对于病人表的主键使用UUID作为主键是出于这样一种考量：当急诊护士导入病人信息且其病情评级所对应的治疗区域有空闲的床位和护士时，病人会被立刻接纳至对应的治疗区域，并在数据库存入病人的信息。在存入病人信息后，还需要将病人的ID存放在其对应的病床的记录中（病人和病床是一一对应关系，理论上也能通过在病人处处存放病床

ID来体现对应关系，但是考虑到要求中需要查询病床是否空闲，将病人ID存放在病床处可以避免一次跨表查询，且从逻辑上来说，病床是属于医院的财产，而病人则是会离开医院的，将会离开的病人存放在固定财产的病床上是更加符合现实逻辑的选择），这时，如果使用自增主键，无法立刻知道刚刚插入的病人的主键是什么，至少需要再通过一次查询（如SELECT LAST_INSERT_ID()）才可能获得刚刚插入数据库的病人的ID，然后再把这个ID存入病床表中；如果使用UUID的话，后端生成UUID，即可直接将其存入病人表和病床表中，提高效率。

3.2 用户的特化实现

系统共有四种用户，分别是主治医生，护士长，病房护士和急诊护士。在ER图设计中，他们都由user特化而来。

将包含概化/特化的ER图转化为关系模式时，有两种不同的方法。一：为高层实体集（user）创建一个模式。为每个低层实体集（doctor, chief nurse等）创建一个模式，模式中的属性包括对应于低层实体集的每个特别属性，以及对应于高层实体集主键的每个属性。其中，高层实体集的主键属性变成既是高层实体集的主键属性，也是所有低层实体集的主键属性。总而言之，就是在高层实体集中添加通用属性，在低层实体集中添加特化属性，并通过主键进行连接。二：不为高层实体集创建任何模式，只为每个低层实体集创建一个模式，模式中的属性包括实体集的所有属性。

在本次项目中，从通用用户到四种类型的用户的特化是不相交且完全的，即：用户一定是四种类型中的一种，且不可能同时担任两种角色，这样的话，使用第二种方案进行特化就不会造成数据冗余。同时由于目前项目中的用户的通用信息本身较少（只有用户名和密码），特化的信息也较少（只有对应的区域或是没有），使用第一种方案需要多表查询，比较繁琐，因此使用第二种方案。

另外需要说明的是，ER图设计中，护士长、急诊护士和病床护士与医生一样，直接由user特化而来。从语言逻辑上来说，一个符合语言逻辑的实现方案是护士和医生有user特化而来，而后护士长、急诊护士和病床护士由护士特化而来。然而，就本项目而言，从功能的角度来说，每种护士之间所涉及的/能使用的功能都非常不同；从数据的角度说，医生、护士长和病房护士都有对应的治疗区域，而急诊护士没有，但从这方面来说，医生、护士长和病房护士这三者反而更像一些。因此，在ER图设计中，没有将护士长、急诊护士和病床护士由护士特化而来，而是将他们和医生放为一级，直接从user特化而来。

3.3 病情评级、治疗区域和生命状态的记录

项目需求文档中指定病情评级有3种：轻症、重症和危重症。类似地，治疗区域和生命状态也有固定的种类。对于这种指定的种类，大体上有3中存储方案：一、直接在数据库用varchar存对应的类型名；二、使用DBMS提供的枚举类型enum；三、在数据库中通过表存储对应的类型名，再在需要这些类型的表中通过外键引用这些类型名。

本项目设计中采用了第三种方案，即通过三个表存储了病情评级、治疗区域和生命状态的几个种类，而后在需要这些类型的表中通过外键引用这些类型。比如PATIENT表中需要记录病人的病情评级，那么只需在PATIENT表中存放数据库中记录的对应评级的主键（在PATIENT表中以外键的形式进行记录）即可。

比较这三种方案。其中，第一种方案是最简便的，DBMS不需要做任何额外的操作或检查，只需将后端调用者提供的字符串准确无误地存入数据库即可。然而，这种方案并不能真正保证准确无误。由于类型的种类是固定的（比如病情评级就是轻症、重症和危重症），但是字符串的输入可以是任意的，用户可能在输入时不小心把critical打成crutical，这样一来，如果后端在存入数据库前不加额外的检查，就会导致数据异常，在之后通过critical进行筛选时就会找不到该项。而如果通过在后端进行检查，就需要在所有可能插入该项的位置处进行检查，难免会有遗忘。而第二种方案中所使用的MySQL的enum类型被广泛地受到抨击，其原因主要是它的实现违反规范化规则、开销极大、难拓展、难关联、优化有限、无法复用和可移植性问

题，详细可见：<https://web.archive.org/web/20180324095351/http://komlenic.com/244/8-reasons-why-mysqls-enum-data-type-is-evil/>。本项目中使用的第三种方案，即 reference table 方案是一种被广泛采用的模式。额外的表看似会造成性能开销和存储浪费，然而，实际上，任何一个像样的 DBMS/数据库都将始终保持这种频繁连接的小表的高速缓存，因此不会造成浪费。最重要的是，采用第三种方案可以在数据库层面通过外键约束的形式保证数据的正确性，如果不是指定类型的数据不能被插入（如刚刚提到的 critical），从而避免在后端的重复检查。

3.4 病情评级、生命状态重复记录的合理性

在病人的属性中，记录了病情评级和生命状态。此外，在病人的核酸检测单中也记录了病人的病情评级，在病人的每日检测记录中也存放了病人的生命状态。这样以来，看似是存在数据冗余，然而实际上，他们的意义是不同的。在病人中存放的病情评级和生命状态信息是实时的，而在核酸检测单和每日记录中存放的这两个信息则是历史的。可以把核酸检测单和每日记录看作是日志，因此他们中的数据并不是冗余数据，只是提供了历史记录。比如主治医生能够更改病人的病情评级和生命状态，那他修改的一定是存放在病人中的当前的病情评级和生命状态，而不是病人最近的核酸检测单中的病情评级或是病人最近的记录中的生命状态。后者是无意义的。因此，在病人的属性中记录病情评级和生命状态不是冗余的。

四、核心功能的SQL语句和存储过程说明

4.1 收治病人

收治病人时，首先从前端（急诊护士）获取病人的基本信息（姓名、地址等）和病人的病情评级，后端首先根据病人的病情评级检查对应的治疗区域是否还有空的床位和病房护士：

```
1 //查询空床位（先查房间）
2 String sql = "SELECT bedId, patientId, roomId FROM bed WHERE roomId in
  (SELECT roomId FROM room WHERE areaId = ?) AND patientId = ' '";
3 //查询空护士
4 //首先查对应区域的所有护士
5 String sql = "SELECT wardNurseId, name, password, areaId FROM ward_nurse
  WHERE areaId = ?";
6 //再遍历所有护士，查他们管理的病人
7 String sql = "SELECT * FROM patient WHERE nurseId = ?";
8 //最后判断数量是否小于对应区域的护士最多能够管理的病人数
```

如果对应治疗区域的床位和病房护士均有空余，则将病人加入该区域，设置护士和床位，并向对应区域的护士长发送提醒消息；如果对应治疗区域的床位和病房护士至少有一个没有空余，则将病人加入隔离区。

```
1 //存病人
2 String sql = "INSERT INTO patient(patientId, name, address, gender,
  telephone, areaId, evaluation, lifeStatus, nurseId)
  VALUES(?,?,?,?,?,?,?,?,?,?)";
3 //更新床位
4 String sql = "UPDATE bed SET patientId = ? WHERE bedId = ?";
5 //向护士长发消息
6 String sql = "INSERT INTO message(toId, toType, messageContent,
  alreadyRead) VALUES(?,?,?,?,?)";
```

4.2 转移病人（包括自动转移）

主治医生更改病人的病情评级后，会主动触发转移操作。转移时，同样查看待转入是否有空闲的床位和病房护士，sql语句同5.1。

如果对应治疗区域的床位和病房护士至少有一个没有空余，则结束并向前端返回提示信息；如果对应治疗区域的床位和病房护士均有空余，则将病人加入该区域，更新护士和床位，并向对应区域的护士长发送提醒消息。

```
1 //更新病人的治疗区域和对应护士
2 String sql = "UPDATE patient SET nurseId = ?, areaId = ? WHERE patientId = ?";
3 //删除原有病床的关联，添加新病床的关联
4 String sql = "UPDATE bed SET patientId = ? WHERE bedId = ?";
5 //向护士长发消息
6 String sql = "INSERT INTO message(toId, toType, messageContent, alreadyRead) VALUES(?,?,?,?)";
```

如果转移成功，则从隔离区开始遍历所有病情评级和治疗区域不符（即待转移）的病人，试着对他们进行转移操作。转移操作和流程同上，进行递归调用直至没有病人可以进行转移为止。

此外，病人生命状态的改变（如：死亡），病床护士的添加以及病人的出院也会使得治疗区域可能变得可转入，因此当执行上述操作时，也会触发自动转移判断，递归查询是否可以执行转移操作。

4.3 查询是否满足出院条件

为了判断病人是否满足出院条件，需要病人最近两次核酸检测结果为阴性（两次检测间隔时间至少为24小时）且连续3天体温正常（低于37.3度）。对于前者，我们查询病人的核酸检测单，找到最近的一个核酸检测单以及第一个离最近核酸检测单的检测时间小于24小时的核酸检测单，判断其检测结果是否都为阴性。对于后者，我们查询病人对应的最近3个每日记录，并判断其体温是否都低于37.3度即可。最后，病人还需在轻症区，这个只需检查病人所在的治疗区域即可。

```
1 //按日期排序查询病人的核酸检测单
2 String sql = "SELECT * FROM test_result WHERE patientId = ? ORDER BY date DESC";
3 //找到最近的3个每日记录
4 String sql = "SELECT * FROM daily_record WHERE patientId = ? ORDER BY date DESC LIMIT 3";
```

4.4 添加核酸检测单/每日记录

主治医生可以为病人添加核酸检测单，病床护士可以为病人添加每日记录。sql语句如下所示。

```
1 //添加核酸检测
2 String sql = "INSERT INTO test_result(patientId, date, testResult, evaluation) VALUES(?,?,?,?)";
3 //添加每日记录
```



```
4 String sql = "INSERT INTO daily_record(patientId, temperature, symptom,
    lifeStatus, date) VALUES(?,?,?,?);";
```

添加核酸检测和每日记录后，可能会使得病人符合出院条件，因此每次添加核酸检测单/每日记录后需要判断病人是否符合出院条件，如果符合，则向病人的主治医生发送提示信息。

五、触发器说明

触发器主要用于进行外键约束无法完成的约束。本项目中共使用了三个触发器，分别用于限制病房内的病床数量、区域内的主治医生和护士长数量。由于这些数据在需求文档中不需要提供前端接口，因此只能在数据库端手动创建，这样就无法在后端判断是否满足约束条件，必须在数据库端进行约束，因此添加触发器进行约束，避免数据导入异常。

5.1 病床触发器

每当在bed表中插入数据前，检查床对应的房间对应的治疗区域，以及房间内当前的病床数。若是轻症区，则最多4张床；若是重症区，最多2张床；若是危重症区，最多一张床。如果不符合约束条件，则报错并拒绝添加。

```
1 CREATE TRIGGER `bed_limit` BEFORE INSERT ON `bed`
2   FOR EACH ROW BEGIN
3     IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 4) and
4       (1 in (SELECT areaId from room where roomId = NEW.roomId))
5     THEN
6       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN MILD';
7     END IF;
8     IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 2) and
9       (2 in (SELECT areaId from room where roomId = NEW.roomId))
10    THEN
11      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN SEVERE';
12    END IF;
13    IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 1) and
14      (3 in (SELECT areaId from room where roomId = NEW.roomId))
15    THEN
16      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN CRITICAL';
17    END IF;
18  END
```

5.2 医生触发器

每当在doctor表中插入数据前，检查要插入的doctor对应的治疗区域是否已经有doctor在管理了，若有，则报错并拒绝添加。

```
1 CREATE TRIGGER `doctor_limit` BEFORE INSERT ON `doctor`
2   FOR EACH ROW BEGIN
3     IF ((select count(doctorId) from doctor where areaId = NEW.areaId) >= 1)
4     THEN
5       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX DOCTOR IN AREA';
6     END IF;
7   END
```

5.3 护士长触发器

每当在chief_nurse表中插入数据前，检查要插入的chief_nurse对应的治疗区域是否已经有chief_nurse在管理了，若有，则报错并拒绝添加。

```
1 CREATE TRIGGER `chief_nurse_limit` BEFORE INSERT ON `chief_nurse`  
2   FOR EACH ROW BEGIN  
3     IF ((select count(chiefNurseId) from chief_nurse where areaId =  
4       NEW.areaId) >= 1)  
5     THEN  
6       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX CHIEF NURSE IN AREA';  
7     END IF;  
8   END
```

六、附录

6.1 典型测试流程

数据库中已存放3个主治医生和3个护士长（分别对应3个治疗区域），1个急诊护士，每个治疗区域2间房间，每间房间放满了所能放的对应的最多的床。

1. 登录轻症和危重症的护士长，分别添加1个病房护士和3个病房护士（**护士长对病房护士的增删**）
2. 登录急诊护士，在轻症和危重症区分别添加4个病人，检查各区域病人。（**急诊护士的收治和查询**）——此时轻症区3人（受限于护士数），危重症区2人（受限于病床数），隔离区3人（1轻症，2危重症）
3. 登危重症区的主治医生，修改1人评级为轻症，并进行各种查询（**主治医生修改病情评级，查询待转入的病人等**）——此时因为轻症区已满，此人症状为轻症但仍在重症区。
4. 登录危重症区的主治医生，修改1人生命状态为死亡（**主治医生修改生命状态**）——隔离区的一名危重症患者会被自动转移入危重症区。（**病人的自动转移**）
5. 登录危重症区的护士长，查看提示信息和各种查询，可以试着删除护士。（**护士长的查询，主动提示护士长转入的病人**）
6. 登录轻症区的主治医生，为1人添加2张核酸检测单（间隔>1天），为另1人添加2张核酸检测单（间隔<1天），在这两人对应的病床护士处添加3张符合出院条件的每日检测。——只有一人符合出院条件。（**主治医生为病人进行核算检测，查询待出院病人，主动提示主治医生可出院病人，病房护士添加每日记录和各种查询**）
7. 登录轻症区的主治医生，查看提示信息，让符合条件的病人出院。此时会把隔离区的轻症病人转入轻症区；而后给另一个添加符合条件的检测单，让他出院，此时会把危重症区待转入轻症区的病人转入轻症区。（**提醒主治医生病人可以出院，查询可以出院的病人，出院，自动转移**）

6.2 建表语句、索引语句和约束语句

```
1 SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";  
2 START TRANSACTION;  
3 SET time_zone = "+00:00";
```

```

5  /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
7  /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
8  /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
9  /*!40101 SET NAMES utf8mb4 */;
10 --
12 -- 数据库: `hospital`
13 --
15 -- -----
16 --
18 -- 表的结构 `area`
19 --
20 CREATE TABLE `area` (
22   `areaId` int(11) NOT NULL,
23   `type` varchar(255) NOT NULL
24 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
25 --
27 -- 转存表中的数据 `area`
28 --
29 INSERT INTO `area` (`areaId`, `type`) VALUES
31 (1, 'mild case area'),
32 (2, 'severe condition area'),
33 (3, 'critical condition area'),
34 (4, 'isolated area'),
35 (5, 'out of hospital');
36 -- -----
37 --
38 --
40 -- 表的结构 `bed`
41 --
42 CREATE TABLE `bed` (
44   `bedId` int(11) NOT NULL,
45   `patientId` varchar(128) DEFAULT '',
46   `roomId` int(11) NOT NULL
47 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
48 --
50 -- 转存表中的数据 `bed`
51 --
52 INSERT INTO `bed` (`bedId`, `patientId`, `roomId`) VALUES
54 (1, '', 1),
55 (2, '', 2),
56 (3, '', 3);
57 --
58 -- 触发器 `bed`
59 --
60 --
61 DELIMITER $$
62 CREATE TRIGGER `bed_limit` BEFORE INSERT ON `bed` FOR EACH ROW BEGIN
63   IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 4) and
64   (1 in (SELECT areaId from room where roomId = NEW.roomId))
65   THEN
66     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN MILD';

```

```

66     END IF;
67     IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 2) and
(2 in (SELECT areaId from room where roomId = NEW.roomId))
68     THEN
69         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN SEVERE';
70     END IF;
71     IF ((select count(bedId) from bed where roomId = NEW.roomId) >= 1) and
(3 in (SELECT areaId from room where roomId = NEW.roomId))
72     THEN
73         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX BED IN CRITICAL';
74     END IF;
75 END
76 $$
77 DELIMITER ;
78 -- -----
80 --
82 -- 表的结构 `chief_nurse`
83 --
85 CREATE TABLE `chief_nurse` (
86     `chiefNurseId` int(11) NOT NULL,
87     `name` varchar(255) NOT NULL,
88     `password` varchar(255) NOT NULL,
89     `areaId` int(11) NOT NULL
90 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
92 --
93 -- 转存表中的数据 `chief_nurse`
94 --
96 INSERT INTO `chief_nurse` (`chiefNurseId`, `name`, `password`, `areaId`)
VALUES
97 (1, 'Dale', '123456', 1),
98 (2, 'Eric', '123456', 2),
99 (3, 'Frank', '123456', 3);
100 --
102 -- 触发器 `chief_nurse`
103 --
104 DELIMITER $$
105 CREATE TRIGGER `chief_nurse_limit` BEFORE INSERT ON `chief_nurse` FOR EACH
ROW BEGIN
106     IF ((select count(chiefNurseId) from chief_nurse where areaId =
NEW.areaId) >= 1)
107     THEN
108         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX CHIEF NURSE IN AREA';
109     END IF;
110 END
111 $$
112 DELIMITER ;
113 -- -----
116 --
117 -- 表的结构 `daily_record`

```

```

118  --
120  CREATE TABLE `daily_record` (
121      `recordId` int(11) NOT NULL,
122      `patientId` varchar(128) NOT NULL,
123      `temperature` float NOT NULL,
124      `symptom` varchar(255) DEFAULT NULL,
125      `lifeStatus` int(11) NOT NULL,
126      `date` datetime NOT NULL
127  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
128  -----
129  --
130  -- 表的结构 `doctor`
131  --
132  CREATE TABLE `doctor` (
133      `doctorId` int(11) NOT NULL,
134      `name` varchar(255) NOT NULL,
135      `password` varchar(255) NOT NULL,
136      `areaId` int(11) NOT NULL
137  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
138  --
139  -- 转存表中的数据 `doctor`
140  --
141  INSERT INTO `doctor` (`doctorId`, `name`, `password`, `areaId`) VALUES
142  (1, 'Alice', '123456', 1),
143  (2, 'Bob', '123456', 2),
144  (3, 'Cindy', '123456', 3);
145  --
146  -- 触发器 `doctor`
147  --
148  DELIMITER $$
149  CREATE TRIGGER `doctor_limit` BEFORE INSERT ON `doctor` FOR EACH ROW BEGIN
150      IF ((select count(doctorId) from doctor where areaId = NEW.areaId) >= 1)
151      THEN
152          SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'MAX DOCTOR IN AREA';
153      END IF;
154  END
155  $$
156  DELIMITER ;
157  -----
158  --
159  -- 表的结构 `emergency_nurse`
160  --
161  CREATE TABLE `emergency_nurse` (
162      `emergencyNurseId` int(11) NOT NULL,
163      `name` varchar(255) NOT NULL,
164      `password` varchar(255) NOT NULL
165  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
166  --
167  -- 转存表中的数据 `emergency_nurse`

```

```

178  --
180  INSERT INTO `emergency_nurse` (`emergencyNurseId`, `name`, `password`)
VALUES
181  (1, 'Tom', '123456');
182  -- -----
183  --
184  -- 表的结构 `illness_evaluation`
185  --
186  CREATE TABLE `illness_evaluation` (
187      `evaluationId` int(11) NOT NULL,
188      `evaluation` varchar(255) NOT NULL
189  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
190  --
191  -- 转存表中的数据 `illness_evaluation`
192  --
193  INSERT INTO `illness_evaluation` (`evaluationId`, `evaluation`) VALUES
194  (1, 'mild case'),
195  (2, 'severe condition'),
196  (3, 'critical condition');
197  -- -----
198  --
199  -- 表的结构 `life_status`
200  --
201  CREATE TABLE `life_status` (
202      `lifeStatusId` int(11) NOT NULL,
203      `lifeStatus` varchar(255) NOT NULL
204  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
205  --
206  -- 转存表中的数据 `life_status`
207  --
208  INSERT INTO `life_status` (`lifeStatusId`, `lifeStatus`) VALUES
209  (1, 'healed'),
210  (2, 'in hospital'),
211  (3, 'dead');
212  -- -----
213  --
214  -- 表的结构 `message`
215  --
216  CREATE TABLE `message` (
217      `messageId` int(11) NOT NULL,
218      `toId` int(11) NOT NULL,
219      `toType` int(11) NOT NULL,
220      `messageContent` varchar(1024) NOT NULL,
221      `alreadyRead` int(11) NOT NULL
222  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
223  -- -----
224  --
225  -- 表的结构 `patient`
226  --

```

```

243 CREATE TABLE `patient` (
244     `patientId` varchar(128) NOT NULL,
245     `name` varchar(255) NOT NULL,
246     `address` varchar(2047) NOT NULL,
247     `gender` varchar(63) NOT NULL,
248     `telephone` varchar(63) NOT NULL,
249     `areaId` int(11) NOT NULL,
250     `evaluation` int(11) NOT NULL,
251     `lifeStatus` int(11) NOT NULL,
252     `nurseId` int(11) DEFAULT 0
253 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
254 -----
255 --
256 -- 表的结构 `room`
257 --
258 CREATE TABLE `room` (
259     `roomId` int(11) NOT NULL,
260     `areaId` int(11) NOT NULL
261 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
262 --
263 -- 转存表中的数据 `room`
264 --
265 INSERT INTO `room` (`roomId`, `areaId`) VALUES
266 (1, 1),
267 (2, 2),
268 (3, 3);
269 -----
270 --
271 -- 表的结构 `test_result`
272 --
273 CREATE TABLE `test_result` (
274     `testResultId` int(11) NOT NULL,
275     `patientId` varchar(128) NOT NULL,
276     `date` datetime NOT NULL,
277     `testResult` varchar(255) NOT NULL,
278     `evaluation` int(11) NOT NULL
279 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
280 -----
281 --
282 -- 表的结构 `ward_nurse`
283 --
284 CREATE TABLE `ward_nurse` (
285     `wardNurseId` int(11) NOT NULL,
286     `name` varchar(255) NOT NULL,
287     `password` varchar(255) NOT NULL,
288     `areaId` int(11) NOT NULL
289 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
290 --
291 -- 转存表中的数据 `ward_nurse`

```

```
304  --
305  INSERT INTO `ward_nurse` (`wardNurseId`, `name`, `password`, `areaId`)
    VALUES
307  (1, 'A', '12345', 1),
308  (2, 'Helen', '123456', 2);
309  --
311  -- 转储表的索引
312  --
313  --
315  -- 表的索引 `area`
316  --
317  ALTER TABLE `area`
318      ADD PRIMARY KEY (`areaId`);
319  --
321  -- 表的索引 `bed`
322  --
323  ALTER TABLE `bed`
324      ADD PRIMARY KEY (`bedId`),
325      ADD KEY `room_bed` (`roomId`);
326  --
328  -- 表的索引 `chief_nurse`
329  --
330  ALTER TABLE `chief_nurse`
331      ADD PRIMARY KEY (`chiefNurseId`),
332      ADD KEY `chief_nurse_area` (`areaId`);
333  --
335  -- 表的索引 `daily_record`
336  --
337  ALTER TABLE `daily_record`
338      ADD PRIMARY KEY (`recordId`),
339      ADD KEY `life_status_record` (`lifeStatus`),
340      ADD KEY `patient_daily_record` (`patientId`);
341  --
343  -- 表的索引 `doctor`
344  --
345  ALTER TABLE `doctor`
346      ADD PRIMARY KEY (`doctorId`),
347      ADD KEY `doctor_area` (`areaId`);
348  --
350  -- 表的索引 `emergency_nurse`
351  --
352  ALTER TABLE `emergency_nurse`
353      ADD PRIMARY KEY (`emergencyNurseId`);
354  --
356  -- 表的索引 `illness_evaluation`
357  --
358  ALTER TABLE `illness_evaluation`
359      ADD PRIMARY KEY (`evaluationId`);
360  --
```



```
362 -- 表的索引 `life_status`
363 --
364 ALTER TABLE `life_status`
365     ADD PRIMARY KEY (`lifeStatusId`);
366 --
367 -- 表的索引 `message`
368 --
369 --
370 ALTER TABLE `message`
371     ADD PRIMARY KEY (`messageId`),
372     ADD KEY `toId` (`toId`),
373     ADD KEY `toType` (`toType`);
374 --
375 -- 表的索引 `patient`
376 --
377 --
378 ALTER TABLE `patient`
379     ADD PRIMARY KEY (`patientId`),
380     ADD KEY `evaluation` (`evaluation`),
381     ADD KEY `area` (`areaId`),
382     ADD KEY `life_status` (`lifeStatus`),
383     ADD KEY `nurseId` (`nurseId`);
384 --
385 -- 表的索引 `room`
386 --
387 --
388 ALTER TABLE `room`
389     ADD PRIMARY KEY (`roomId`),
390     ADD KEY `area_room` (`areaId`);
391 --
392 -- 表的索引 `test_result`
393 --
394 --
395 ALTER TABLE `test_result`
396     ADD PRIMARY KEY (`testResultId`),
397     ADD KEY `evaluation_test` (`evaluation`),
398     ADD KEY `patient_test` (`patientId`);
399 --
400 -- 表的索引 `ward_nurse`
401 --
402 --
403 ALTER TABLE `ward_nurse`
404     ADD PRIMARY KEY (`wardNurseId`),
405     ADD KEY `ward_nurse_area` (`areaId`);
406 --
407 -- 在导出的表使用AUTO_INCREMENT
408 --
409 --
410 --
411 -- 使用表AUTO_INCREMENT `area`
412 --
413 --
414 ALTER TABLE `area`
415     MODIFY `areaId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
416 --
417 -- 使用表AUTO_INCREMENT `bed`
```

```
419  --
420  ALTER TABLE `bed`
421      MODIFY `bedId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;
422  --
423  -- 使用表AUTO_INCREMENT `chief_nurse`
424  --
425  --
426  ALTER TABLE `chief_nurse`
427      MODIFY `chiefNurseId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
428  --
429  -- 使用表AUTO_INCREMENT `daily_record`
430  --
431  --
432  ALTER TABLE `daily_record`
433      MODIFY `recordId` int(11) NOT NULL AUTO_INCREMENT;
434  --
435  -- 使用表AUTO_INCREMENT `doctor`
436  --
437  --
438  ALTER TABLE `doctor`
439      MODIFY `doctorId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
440  --
441  -- 使用表AUTO_INCREMENT `illness_evaluation`
442  --
443  --
444  ALTER TABLE `illness_evaluation`
445      MODIFY `evaluationId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
446  --
447  -- 使用表AUTO_INCREMENT `life_status`
448  --
449  --
450  ALTER TABLE `life_status`
451      MODIFY `lifeStatusId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
452  --
453  -- 使用表AUTO_INCREMENT `message`
454  --
455  --
456  ALTER TABLE `message`
457      MODIFY `messageId` int(11) NOT NULL AUTO_INCREMENT;
458  --
459  -- 使用表AUTO_INCREMENT `room`
460  --
461  --
462  ALTER TABLE `room`
463      MODIFY `roomId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
464  --
465  -- 使用表AUTO_INCREMENT `test_result`
466  --
467  --
468  ALTER TABLE `test_result`
469      MODIFY `testResultId` int(11) NOT NULL AUTO_INCREMENT;
470  --
471  -- 使用表AUTO_INCREMENT `ward_nurse`
472  --
473  --
474  ALTER TABLE `ward_nurse`
475      MODIFY `wardNurseId` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;
476  --
```

```
478 -- 限制导出的表
479 --
480 --
482 -- 限制表 `bed`
483 --
484 ALTER TABLE `bed`
485     ADD CONSTRAINT `room_bed` FOREIGN KEY (`roomId`) REFERENCES `room`
486     (`roomId`);
487 --
488 -- 限制表 `chief_nurse`
489 --
490 ALTER TABLE `chief_nurse`
491     ADD CONSTRAINT `chief_nurse_area` FOREIGN KEY (`areaId`) REFERENCES
492     `area` (`areaId`);
493 --
494 -- 限制表 `daily_record`
495 --
496 ALTER TABLE `daily_record`
497     ADD CONSTRAINT `life_status_record` FOREIGN KEY (`lifeStatus`)
498     REFERENCES `life_status` (`lifeStatusId`),
499     ADD CONSTRAINT `patient_daily_record` FOREIGN KEY (`patientId`)
500     REFERENCES `patient` (`patientId`);
501 --
502 -- 限制表 `doctor`
503 --
504 ALTER TABLE `doctor`
505     ADD CONSTRAINT `doctor_area` FOREIGN KEY (`areaId`) REFERENCES `area`
506     (`areaId`);
507 --
508 -- 限制表 `patient`
509 --
510 ALTER TABLE `patient`
511     ADD CONSTRAINT `area` FOREIGN KEY (`areaId`) REFERENCES `area`
512     (`areaId`),
513     ADD CONSTRAINT `evaluation` FOREIGN KEY (`evaluation`) REFERENCES
514     `illness_evaluation` (`evaluationId`),
515     ADD CONSTRAINT `life_status` FOREIGN KEY (`lifeStatus`) REFERENCES
516     `life_status` (`lifeStatusId`);
517 --
518 -- 限制表 `room`
519 --
520 ALTER TABLE `room`
521     ADD CONSTRAINT `area_room` FOREIGN KEY (`areaId`) REFERENCES `area`
522     (`areaId`);
523 --
524 -- 限制表 `test_result`
525 --
526 ALTER TABLE `test_result`
```

```
524     ADD CONSTRAINT `evaluation_test` FOREIGN KEY (`evaluation`) REFERENCES
      `illness_evaluation` (`evaluationId`),
525     ADD CONSTRAINT `patient_test` FOREIGN KEY (`patientId`) REFERENCES
      `patient` (`patientId`);
526 --
527 -- 限制表 `ward_nurse`
528 --
529 --
530 ALTER TABLE `ward_nurse`
531     ADD CONSTRAINT `ward_nurse_area` FOREIGN KEY (`areaId`) REFERENCES
      `area` (`areaId`);
532 COMMIT;
533 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
534 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
535 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```