

中山大学数据科学与计算机学院

计算机科学与技术专业

(2018-2019 秋季学期)

数据库系统

酒店房间预定系统

院 (系) 名 称: 数据科学与计算机学院

专 业 名 称: 计算机科学与技术

教 学 班 级: 计科教务二班

学 生 姓 名: 李新锐, 颜彬, 王永锋

学 生 学 号: 15323032, 16337269, 16337237

指 导 教 师: 阮文江

二〇一八年十二月二十一日

目 录

1	引言	1
2	系统分析与设计	2
2.1	系统需求分析	2
2.2	系统结构设计	2
3	数据库设计	4
3.1	概念设计	4
3.1.1	实体集	4
3.1.1.1	房间和房型实体集	4
3.1.1.2	用户, 地区和账户实体集	5
3.1.1.3	订单和操作实体集	6
3.1.2	联系集	6
3.1.3	E-R 图	7
3.2	逻辑设计	7
3.3	物理设计	8
3.3.1	索引简介	9
3.3.1.1	索引建立	9
3.3.2	效果评价	10
3.3.3	其他索引的需求分析	10
4	数据库创建和数据加载	12
4.1	视图	12
4.2	存储过程	12
4.2.1	查询可用房间	12
4.2.2	预定房间	13
4.2.3	退房	14
4.3	建立触发器	16
4.4	事务机制	16

4.5	自主存取控制	17
4.6	测试数据生成	17
5	数据库应用软件设计开发与测试	18
5.1	系统整体架构	18
5.2	后端 API 设计	18
5.3	系统界面说明	20
6	系统测试	22
6.1	用户预订房间	22
7	开发所遇到的问题与解决方案	24
7.1	如何保证数据一致性?	24
7.2	如何在查询订单时查询订单相关的操作明细?	25
7.3	用户权限的控制	25
7.4	存储过程的传参和返回值	25
7.5	NULL 与数字比较	26

1 引言

在本文中，我们小组对酒店客房预订管理系统进行了系统调查，分析与设计，进行了详尽的需求分析，并基于用户需求，设计了一个高效且规范的数据库模式。在此基础上，我们创建了 Mysql 数据库，并使用 Html 和 Javascript 编写了在线酒店客房预定管理系统。

本系统采用前后端分离的方式进行开发，前端使用了基于 nodejs 及多种组件开发而成，并通过 Ajax 与后端相连接，两者通过一套详细、准确且完备的 API 作解耦。后端则完成两部分的功能，一部分是处理复杂的业务逻辑，对各种诸如注册，增添新用户，新房间，或增加新订单等接口进行处理，保证对数据库操作的一致性，另一部分是对传入 API 的数据进行格式检查，防止出现 SQL 注入或者恶意攻击的情况。

在后端，除了处理复杂业务逻辑，还负责与数据库进行交互，我们编写了很多 SQL 查询与更新语句，在后端中根据接口传入的数据生成 SQL 语句，并交由数据库查询与更新。

在数据库层面，我们发现一些复杂的业务逻辑在后端编写时一定要确保一致性，否则会产生严重的逻辑错误。因此我们在数据库中编写了一些触发器与存储过程，以保证修改数据库信息时信息的一致性，对于在处理数据库数据时产生的错误，也在后端进行了适当的处理并反馈到前端中让用户知晓。

总之，我们遵循数据库设计与网站设计与编写的方法，实现了酒店预定系统，该系统包括客房预订、会员注册、用户管理、客房管理、客户与客房的信息增删改、以及订单管理等功能，已经能够满足大多数酒店的需求。

2 系统分析与设计

随着信息化时代的到来，使用数据库来对酒店客房预订信息进行管理能够大大提高信息管理的效率。基于网络对酒店管理系统的了解，我们确定了该系统所需要满足的需求，以及我们系统应具有的功能模块。

2.1 系统需求分析

我们的系统可供酒店经理，酒店前台人员以及顾客使用。其中，不同的用户对该系统有着不同的需求，以及相应的操作权限。

对于**顾客**而言，他们需要能够在系统中：

- 查询可用的房间。
- 预定指定的房间，提交订单。
- 取消自己已经订好的订单。
- 查看自己存在酒店中的个人信息，并且修改某些易变的项（如手机号码等）。

而对于**酒店前台人员**而言，他们不仅需要顾客本身可实现的需求，而且还应有更多信息的查询权限，更全面地说，应该如下所示：

- 查询酒店所有房间的信息。
- 查询酒店所有订单的信息。
- 可能需要根据需要，取消其他用户的订单。
- 增加新用户。

酒店经理，则作为整个酒店的管理者，应当具有更多信息的修改权限，更具体的说，应该如下所示：

- 增加酒店的房间类型。
- 酒店若新装修了房间，应当能够增加新的房间，或者修改已有的房间类型。
- 还需要能够对所有订单的情况做一个统计，计算酒店的营业额等信息。

2.2 系统结构设计

本系统可以分为这 5 个子系统，可见图 2.1，每一个子功能都会在后端编写一个或多个对应的 API 供前端调用。



图 2.1 系统子系统设计

3 数据库设计

在本节中，我们会按照概念设计、逻辑设计到物理设计的顺序讲解本项目的具体设计方式。

3.1 概念设计

根据需求分析的结果，我们进行了数据库的概念设计，分析了数据库中应该存在的实体集、联系集，并绘制了 ER 图。

3.1.1 实体集

描述酒店的房间需要房间实体集 (表 3.1) 和房型实体集 (表 3.2)。描述酒店的用户并记录用户个人和登录信息需要用户实体集 (表 3.3)、账户实体集 (表 3.4)、区域实体集 (表 3.5)。最后，酒店还需要记录所有订单 (表 3.6)，以及订单的操作明细 (表 3.7)。

下面对这些实体集作简要的描述和区分。

3.1.1.1 房间和房型实体集

房型实体代表房内的基础设施状况，例如房内是否有 wifi、是否包含早餐、容纳的人数等，可见表 3.2。

表 3.1 房间实体集

属性	备注
<u>id</u>	唯一
floor	层数
room_num	房间号
price	价格

而房间实体则对应着真实的每个房间。其具有所在层数、房号、价格等属性，可见表 3.1。

表 3.2 房型实体集

属性	备注
<u>id</u>	唯一
name	房型名称
capacity	最大容纳人数
breakfast	是否包含早餐
wifi	是否包含无线网络

3.1.1.2 用户，地区和账户实体集

用户实体代表一个真实的人。它有自己的身份信息，例如证件号、姓名、性别等。其中用户实体集中的 **name** 属性代表这个人的真实姓名，可见表 3.3。

表 3.3 用户实体集

属性	备注
<u>id</u>	唯一
region	顾客来自的地区
credential	证件号，唯一
name	姓名
gender	性别
birthdate	出生日期
phone	电话号码

账户实体集代表本应用上的一个账号。其中的账户名 (**username**) 是登陆名，**password** 用于登陆。**role** 为该账户在应用里的角色，它决定了该账户拥有的权限，可见表 3.4。

同时，为了保证每一个用户所在的地区有意义，我另外使用了一个地区实体用于确定用户所在的地区，以防止用户输入无效的地区，该实体可见表 3.5。

表 3.4 账户实体集

属性	备注
<u>id</u>	唯一
username	账户名
password	账户密码
role	用户角色，0 为管理者，3 为普通用户
balance	余额
bonus	积分

表 3.5 区域实体集

属性	备注
<u>id</u>	唯一
name	区域名称

3.1.1.3 订单和操作实体集

订单实体的信息包含了哪个用户、在何时、花费多少钱、在哪个时间段订购了哪个房间，可见表 3.6。

表 3.6 订单时实体集

属性	备注
<u>id</u>	唯一
status	订单状态（取消、正常）
check_in	预订的入住时间
check_out	预订的最后一天

操作实体集包含了用户对订单的两种操作，分别是发起订单和取消订单，可见表 3.7。

3.1.2 联系集

房型与房间实体、房间和订单实体、用户与订单实体、区域与用户实体之间有一对多的联系。对于一个订单只有完成下单和完成下单后又取消订单两种情况，

表 3.7 操作明细实体集

属性	备注
id	唯一
time	操作的发生时间
detail	操作类型（发起订单、取消订单）

因此订单实体与订单操作实体是一对一或一对二的联系。用户与账户之间是一对一联系。

3.1.3 E-R 图

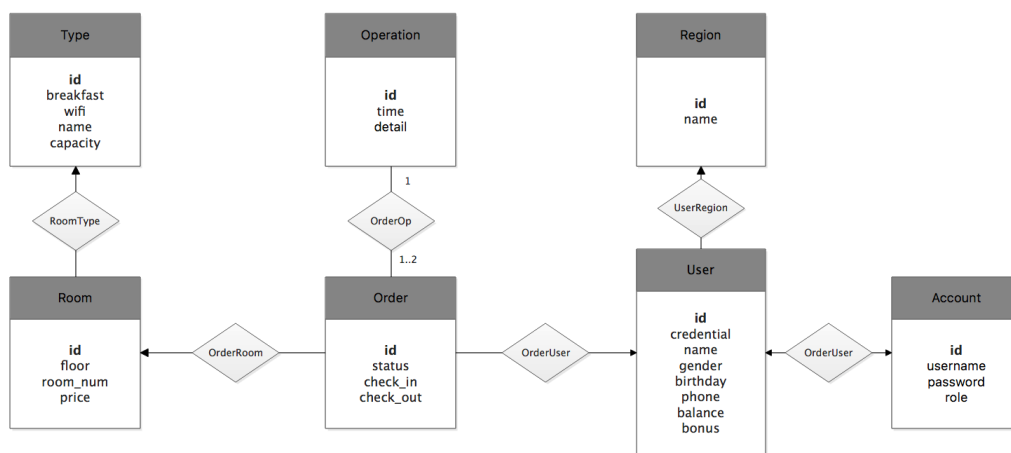


图 3.1

3.2 逻辑设计

在完成数据库的概念设计后，接下来要做的是将所有实体集合联系集分别转换为关系模式 R 。

有关实体集的转换规则如下：

- 具有 n 个简单描述性属性的强实体集转换为具有 n 个属性的关系模式
- 为复合属性的每个子属性在 R 中创建一个单独的属性
- 为多值属性 M 创建一个单独的关系模式，该模式中包含了 M 所在的实体集或联系集的主码
- 将派生属性实现为方法

- 将弱实体集表示为包含所有弱实体集属性以及所依赖的强实体集的主码的关系模式。 R 的主码包括弱实体集的部分码和所以来的强实体集的主码。

有关联系集的转换规则如下：

- 联系集转换为由联系集的描述性属性以及相关实体集的主码组成的关系模式
- 对于多对多的联系集， R 的主码包含所有相关实体集的主码
- 对于一对一的联系集，任意一个相关实体集的主码都可以作为 R 的主码
- 对于一对多的联系集，用关联的映射基数为“多”的那个实体集作为 R 的主码

转换完成后，还要消除关系模式的冗余，并合并部分关系模式。具体而言包括以下两种情况：

- 连接弱实体集与其依赖的强实体集的联系集的模式是多对一且没有描述性属性的，另外弱实体集的主码包含强实体集的主码，因此这样的联系集对应的关系模式是冗余的。
- 考虑多对一的联系集 AB 和相关的实体集 A, B ，若 A 在联系中的参与是全部的，那么可以将 A 与 AB 模式属性合并得到 A^* ，主码是 A 的主码，外码加入 A^* 中。

依照上述规则，由于 Type、Room、Operation、Order、User、Region、Account 均为简单强实体集于是可以首先生成包含对应属性的关系模式。再考虑联系集，由于 OrderUser、RoomType、OrderRoom、OrderOp、UserRegion 均为一对多的联系集，因此均可以按照上述规则消除。

最后得到以下关系模式：

RoomType(**id**, name, capacity, breakfast, wifi)

Room(**id**, floor, room_num, price, type_id)

Operation(**id**, time, detail, order_id)

Order(**id**, status, check_in, check_out, room_id, user_id)

Region(**id**, name)

User(**id**, credential, name, gender, birthdate, phone, bonus, balance, region_id)

Account(**id**, username, password, role)

3.3 物理设计

在这一部分中，我们主要对数据库中索引的设计做了一个比较详细的解释，说明了对数据库性能问题的考虑与我们的解决方案。

3.3.1 索引简介

在 MariaDB 中，有 4 种类型的索引（Mariadb 不严格地区分 key 和 index 的概念，并时常根据语境对他们混用）。

表 3.8列出了这 4 种索引。其中 unique 表示索引建立的数据项是否必须唯一，not null 表示数据项是否必须不为 null。

mariaDB 会自动为 primary key 和 foreign key 建立索引（表第一行）。其中如果对 unique 的数据项建立索引，可以建立一个 unique 索引（表第二行）。如果该数据项不是唯一的，则需要建立 plain indexes 或 full-text indexes。后两者最大的不同在于，如果使用 like ‘%word%’ 这种形式的查询，必须建立 full-text 索引，否则无效。

表 3.8 mariaDB 索引介绍

名称	<i>unique</i>	<i>not null</i>
primary(foreign) key	Y	Y
unique indexes	Y	N
plain indexes	N	N
full-text indexes	N	N

3.3.1.1 索引建立

搜索订单几乎是最常见的一种操作。从用户的行为来判断，大部分对订单的搜索都是基于订单日期的。故在 Order 表上对 check_in 和 check_out 添加索引十分有必要。

```
1 CREATE INDEX Order__index_check_in ON `Order` (check_in DESC);
```

在上面的索引建立中，显式地指明了索引使用降序排列。这是因为根据时间局部性，越“晚(刚刚发生)”的记录越可能被搜索，越“早(年代久远)”的记录越不可能被搜索。降序排列索引可以尽可能早地让索引被搜索到。

3.3.2 效果评价

```
1 explain
2 select * from `Order`
3 where check_in >= '2019-01-22';
```

比较有索引和无索引时的相关输出。没有建立索引时，仅能根据主键索引进行查询。查询了 300 多行。但建立了对 `check_in` 的索引后，以上的语句仅需要搜索 75 行，大大加快了搜索时间。

进一步测试以下的语句

```
1 explain select * from `Order`
2 where check_in >= '2019-01-22'
3 AND check_out <= '2020-01-01';
```

发现 mariaDB 的做法是，先根据索引 `check_in` 作 range scan，估计会检索到 75 个左右的行。然后对这些行作 where 筛选，得到最终的结果。

在上面的语句执行中，没有用到 `check_out` 索引，但这个索引依旧有它的必要指之处。mariaDB 会选择 `check_in` 或 `check_out` 中的一个索引作 range scan，很可能在下一次更合适的情况下会使用 `check_out`。维护 `check_in` 和 `check_out` 两个索引，可以让 mariaDB 有更多提升性能的可能性。

更进一步的分析，如果用户想要筛选订单，必行会出现 `user_id = <int>` 的情况。此时 `user_id` 一定是更适合的索引。但不能忽略的是，酒店的前台也需要筛选订单。后者往往是根据日期作筛选的，而且这个操作的发生极其频繁。故这个索引十分有必要。

3.3.3 其他索引的需求分析

大部分对 User 表的查询都是使用 id 进行查询的。user name 是用户的真名，对用户真名的查询操作其实很少。故现有的主键索引已经足够。

对 Account 的查询中，由于 username 被标记为 unique（IDE 自动为我们建立

了索引)，故对 Account 的大部分查询效率都很高。

一个比较微妙的操作是查询空闲房间。因为查询空闲房间涉及到查询房间 (Room)、查询房间类型 (RoomType) 和查询订单 (Order) 三个表的数据。实际上这个操作的效率也是很高的。对此详细的介绍见后文。

我们的应用经常需要返回“空闲的房间数”。可是怎么定义空闲呢？如果对于一段查询时间 [check_in, check_out], 不存在与之重叠的订单 (order)，那么我们就说这个房间是空闲的。但实际的查询并不是通过使用 check_in 和 check_out 的索引进行的。

```
1  explain
2  select R.id, R.floor, R.room_num, R.price, T.breakfast, T.wifi ,T.name, T.capacity
3  from Room as R, RoomType as T
4  where R.type_id = T.id
5         and T.capacity >= 2
6         and T.wifi like 1
7         and T.breakfast like 1
8         and not exists
9             ( select room_id, O.id
10              from `Order` as O
11              where status = 1
12                    and ((O.check_in <= '2018-11-20' and O.check_out >= '2018-12-30')
13                      or (O.check_in <= '2018-10-30' and O.check_out >= '2018-11-10'))
14              and O.room_id = R.id );
```

使用 explain 查看查询计划后，我们发现，mariaDB

- 首先使用 where 筛选掉不符合要求的 RoomType。由于 RoomType 是很小的一个表，这一步的效率非常高。
- 利用 Room 对 RoomType 的外键索引筛选合适的 Room。由于利用了索引，这一步速度也非常高。
- 利用 Order 对 Room 的外键索引筛选合适的 Order。由于利用了索引，这一步的速度也非常高。

这说明我们没有额外建立索引的必要了，现有的主键和外键索引已经能让数据库的执行速度足够高了。

只有订单的 check_in 和 check_out 需要额外的索引。其他操作都可以通过主键外键索引、unique 索引得到加速。

4 数据库创建和数据加载

4.1 视图

在酒店管理系统中，为了对账需要，所有历史订单都不能删除。但用户通常只关心还未退房的订单，系统在查询空房时，也只是对照那些还没退房的订单使用排除法进行查找。因此，保存一个仅包含还未退房的订单的视图能够方便数据库后台的管理和应用的实现，视图实现的代码如下。

```
1  create view VIEW_available_orders as
2  select *
3      from `Order` o
4      where o.status = 1 and o.check_out > CURDATE();
```

4.2 存储过程

4.2.1 查询可用房间

在酒店管理系统中，我们不可能存储未来每一天每间房间是否可用的状态。要查询某个指定时间段空闲的房间有哪些，必须根据当前有效的订单信息，查询并排除该时间段内已经被预定的房间。由于查询可用房间是酒店数据库最常用的功能，因此我们设计了存储过程：

```
1  create procedure PROC_find_avail_room
2  (in Arg_check_in Date, in Arg_check_out Date,
3   in Arg_capacity int, in Arg_wifi char(1), in Arg_breakfast char(1))
4  begin
5      select R.id,R.floor, R.room_num,R.price,
6      T.breakfast, T.wifi ,T.name, T.capacity
7      from Room as R, RoomType as T
8      where R.type_id = T.id and T.capacity >= Arg_capacity
9      and T.wifi like Arg_wifi and T.breakfast like Arg_breakfast and
10 (not exists (select * from VIEW_available_orders o
11              where R.id = o.room_id and Arg_check_in <= o.check_out and
12                  Arg_check_out >= o.check_in
13 ));
14 end;
```

该存储过程的参数说明可见表 4.1。

表 4.1 查询可用房间-参数说明

参数	类型	值
Arg_check_in	Date	预约的第一天
Arg_check_out	Date	预约的最后一天
Arg_capacity	int	预约的房型的最低容量
Arg_wifi	char(1)	是否要求要 WiFi
Arg_breakfast	char(1)	是否要求送早餐

可以通过以下方式调用该存储过程，表示查询入住时间为 2019 年 1 月 22 日，退房时间也为 2019 年 1 月 22 日测情况下，有 wifi 且可住两人的空闲房间。

```
1 call PROC_find_avail_room('2019-01-22', '2019-01-22', 2, '1', '%');
```

4.2.2 预定房间

预定房间在系统中设计多个操作，首先要在数据库中查询该房间是否可被预定，然后如果没有被预定，则需要在 Order 表中生成一个新订单，然后在 Operation 表中生成一条该订单的生成记录，同时还需要扣除指定用户余额，这一连串的操作应当以事务的形式来实现，因此我们将整个复杂的过程，使用存储过程在数据库中实现，代码可见如下：

```
1 create or replace procedure PROC_order_room
2 (in Arg_room_id int, in Arg_user_id int, in Arg_check_in Date, in Arg_check_out Date)
3 begin
4     declare L_price integer ;
5     declare L_days integer ;
6     declare L_already_occupied integer ;
7     select (DATEDIFF(Arg_check_out, Arg_check_in)+1) into L_days;
8     start transaction ;
9     select count(o.id) from VIEW_available_orders o
10 where Arg_room_id = o.room_id and Arg_check_in <= o.check_out and Arg_check_out >=
    ↪ o.check_in
11 into L_already_occupied;
12 IF (L_already_occupied) THEN
13 SIGNAL SQLSTATE '45000' SET
14 MYSQL_ERRNO = 30004,
```



```

15 MESSAGE_TEXT = 'Sorry, the room is already occupied';
16 elseif (L_days < 0) THEN
17 SIGNAL SQLSTATE '45000' SET
18 MYSQL_ERRNO = 30001,
19 MESSAGE_TEXT = 'Wrong order(Arg_check_in > Arg_check_out)';
20 else
21 select r.price from Room r
22 where r.id = Arg_room_id
23 into L_price;
24 insert into `Order` (room_id, user_id, check_in, check_out, status,fee) value
25 (Arg_room_id, Arg_user_id, Arg_check_in, Arg_check_out, 1, L_price * L_days);
26 insert into Operation(time, detail, order_id) value( now(), 1, LAST_INSERT_ID());
27 end if;
28 commit;
29 end;

```

该存储过程的参数说明可见表 4.2。

表 4.2 预定房间-存储过程

参数	类型	值
Arg_room_id	int	想预订的房间 id
Arg_user_id	int	进行预订的用户
Arg_check_in	Date	预约的入住时间
Arg_check_out	Date	预约的最后一天

4.2.3 退房

退房操作同样涉及到较多的操作，首先需要判断该房间是否已经预定，只有已经预定了的房间才能够被取消；在确定能够退房后，需要以事务的方式完成以下的一连串操作：在 **Order** 表中将指定订单的状态设置为取消，然后计算退房费用并返还到用户账户中，同时还需要在 **Operation** 表中生成一条新纪录，记录该订单的取消操作。以事务的方式完成上述一系列操作能够保证数据库中数据的一致性。

该存储过程的代码可见如下，只需要为该存储过程提供订单编号即可进行操作。

```

1 create procedure PROC_cancel_order(IN Arg_order_id int)
2 begin
3     declare O_status int;
4     declare O_check_in Date;
5     declare O_check_out Date;

```

```

6  declare O_refund_days int;
7  select -1 into O_status;
8  select O.status from VIEW_available_orders O
9  where O.id = Arg_order_id
10 into O_status;
11 select O.check_in
12 from VIEW_available_orders O
13 where O.id = Arg_order_id
14 into O_check_in;
15 select O.check_out
16 from VIEW_available_orders O
17 where O.id = Arg_order_id
18 into O_check_out;
19 IF (CURDATE() >= O_check_out or O_status = 0 or O_status = -1) THEN
20 SIGNAL SQLSTATE '45000' SET
21 MYSQL_ERRNO = 30005,
22 MESSAGE_TEXT = 'Sorry, cancelling this order is not available';
23 elseif (CURDATE() < O_check_in) Then
24 start transaction;
25 update User, `Order`
26 set User.balance = User.balance + `Order`.fee
27 where `Order`.id = Arg_order_id and `Order`.user_id = User.id;
28 update `Order`
29 set `Order`.status = 0
30 where `Order`.id = Arg_order_id;
31 insert into Operation(time, detail, order_id)
32 value (now(), 2, Arg_order_id );
33 commit;
34 else
35 start transaction;
36 select DATEDIFF(O_check_out, CURDATE())
37 into O_refund_days;
38 update User, `Order`
39 set User.balance =
40     User.balance + O_refund_days *
41     `Order`.fee / (O_check_out - O_check_in + 1)
42     where User.id = `Order`.user_id and `Order`.id = Arg_order_id;
43 update `Order`
44 set `Order`.status = 0
45 where `Order`.id = Arg_order_id;
46 insert into Operation(time, detail, order_id) value (now(), 2, Arg_order_id);
47 commit ;
48 end if;
49 end;

```

4.3 建立触发器

用户每次订房时要从余额中扣取房费并给予积分奖励，这一操作必须在订单生成前完成，并对用户余额进行检查，避免用户余额不足，该触发器的实现可见以下代码：

```
1 create or replace trigger TRI_order_fee
2   before insert on `Order`
3   for each row
4   begin
5     declare fee integer;
6     declare days integer;
7     declare old_balance integer ;
8     select balance into old_balance
9     from User u
10    where u.id = New.user_id;
11    IF (old_balance = null or old_balance - New.fee < 0) THEN
12      SIGNAL SQLSTATE '45000' SET
13      MYSQL_ERRNO = 30001,
14      MESSAGE_TEXT = 'You dont have enough balance';
15    else
16      update User u
17      set balance = old_balance - New.fee
18      where u.id = NEW.user_id;
19    END IF;
20  end;
```

4.4 事务机制

出于安全性考虑，我们将用户的账户与用户的个人信息分在两个表中分开存储。而酒店一般要求用户实名制登记，因此在注册时不仅要在账户表中添加账户，而且一定要在用户信息表相应地记录真实姓名和证件号码。为了确保用户注册操作的原子性，我们采用了事务机制，将两次插入操作在一次事务中提交。

通过，对订单的操作同样也采用了事务机制，“修改用户余额”，“生成订单”，“生成订单操作明细”三个操作的执行必须一一整个事务的方式来执行，否则可能会出现不一致的情况。

这一部分的代码实现，可见节 4.2 存储过程一节。

4.5 自主存取控制

为了防止用户账户失窃、账单等数据因操作不当丢失，我们设计了自主存取控制机制，将数据库后台人员分为管理员和审计员，对用户权限进行如下分配：

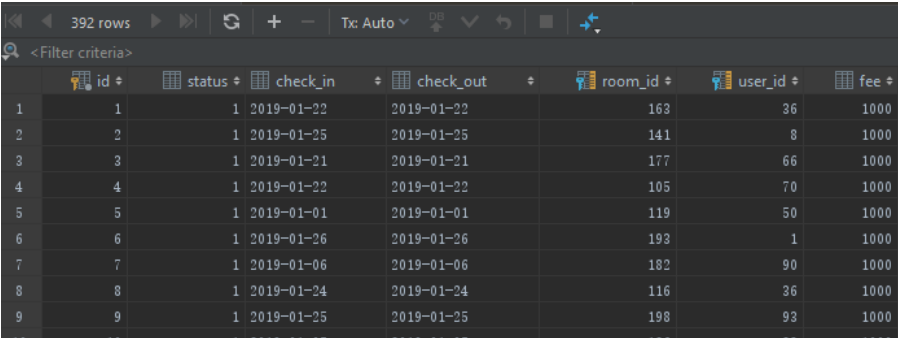
- 只有管理员有查看和修改 Account 表的权限
- 审计员对其他表只有查看权限
- 管理员对 Order 表有除了 delete、alter、delete history 之外的其他权限
- 管理员对其他表有所有权限

4.6 测试数据生成

在这一步，我使用了 python 编写生成数据的脚本，并在其中对表中已存在的约束做了适当的控制，保证初始化的数据在数据库中是一致的。具体而言，我关注到了以下细节：

- Room 表中的 type_id，受到 RoomType 表中 id 一列的外键约束。
- Account 表中的 id，受到 User 表中 id 一列的外键约束。
- Order 表中的 user_id，room_id，均受到 User 表 id 与 Room 表 id 的外键约束。
- Order 表中预约成功的订单，在 Operation 表中应有一条对应的生成订单记录。
- Order 表中预约失败的订单，在 Operation 表中应有两条对应的订单操作记录，一条为生成，另一条为取消。

测试数据示例可见图 4.1



	id	status	check_in	check_out	room_id	user_id	fee
1	1	1	2019-01-22	2019-01-22	163	36	1000
2	2	1	2019-01-25	2019-01-25	141	8	1000
3	3	1	2019-01-21	2019-01-21	177	66	1000
4	4	1	2019-01-22	2019-01-22	105	70	1000
5	5	1	2019-01-01	2019-01-01	119	50	1000
6	6	1	2019-01-26	2019-01-26	193	1	1000
7	7	1	2019-01-06	2019-01-06	182	90	1000
8	8	1	2019-01-24	2019-01-24	116	36	1000
9	9	1	2019-01-25	2019-01-25	198	93	1000
10	10	1	2019-01-07	2019-01-07	126	82	1000

图 4.1 测试数据示例

关于生成数据相关的 python 代码，由于与该系统设计关系不大，因此不放在报告中呈现。

5 数据库应用软件开发与测试

5.1 系统整体架构

我们系统设计采用了前后端分离的模式，具体可见图 5.1。

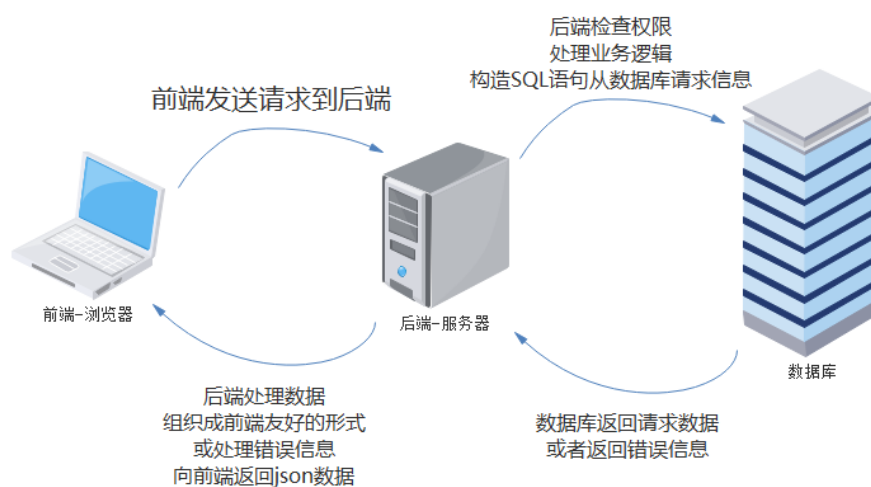


图 5.1 系统整体架构

5.2 后端 API 设计

关于后端的 API 设计，我们编写了一份完整且规范的 API 文档，其中含有每一个 API 的参数说明，返回值，以及错误信息的设置，文档将会在项目中另外附上一份文件进行说明，截图可见图 5.2：



图 5.2 后端 API 文档示例

5.3 系统界面说明

关于系统的界面，需要注意到，酒店经理所看到的系统界面与用户所看到的系统界面应当是不同的，用户界面可见图 5.3，酒店经理使用界面可见图 5.4。

RoomsOrders

logoutprofile

check in

2018-12-22

指定入住日期

check out

2018-12-22

指定退房日期

capacity

let it empty to ignore

入住人数

requirment

wifi

breakfast included

对房间的一些其他要求

Query Available

clear

筛选符合条件的“可预订”房间（会排除掉已经被预定的房间）

room number

let it empty to ignore

result

id	floor	room_num	price	breakfast	wifi	name	capacity	action
100	1	100	100	No	No	小小房	1	order
101	1	101	100	No	No	小小房	1	order
102	1	102	100	No	No	小小房	1	order
103	1	103	100	No	No	小小房	1	order
104	1	104	100	No	No	小小房	1	order
105	1	105	100	No	No	小小房	1	order

图 5.3 用户界面

RoomsUsersOrdersRoot

logoutprofile

check in

2018-12-22

check out

2018-12-22

capacity

let it empty to ignore

requirement

wifi

breakfast included

Query Available

clear

room number

let it empty to ignore

price

let it empty to ignore

floor

let it empty to ignore

Add

Room Type

new room type name he

Add Type

查询可用房间功能模块

增加房间功能模块

房型管理功能模块

可用于增加房间类型

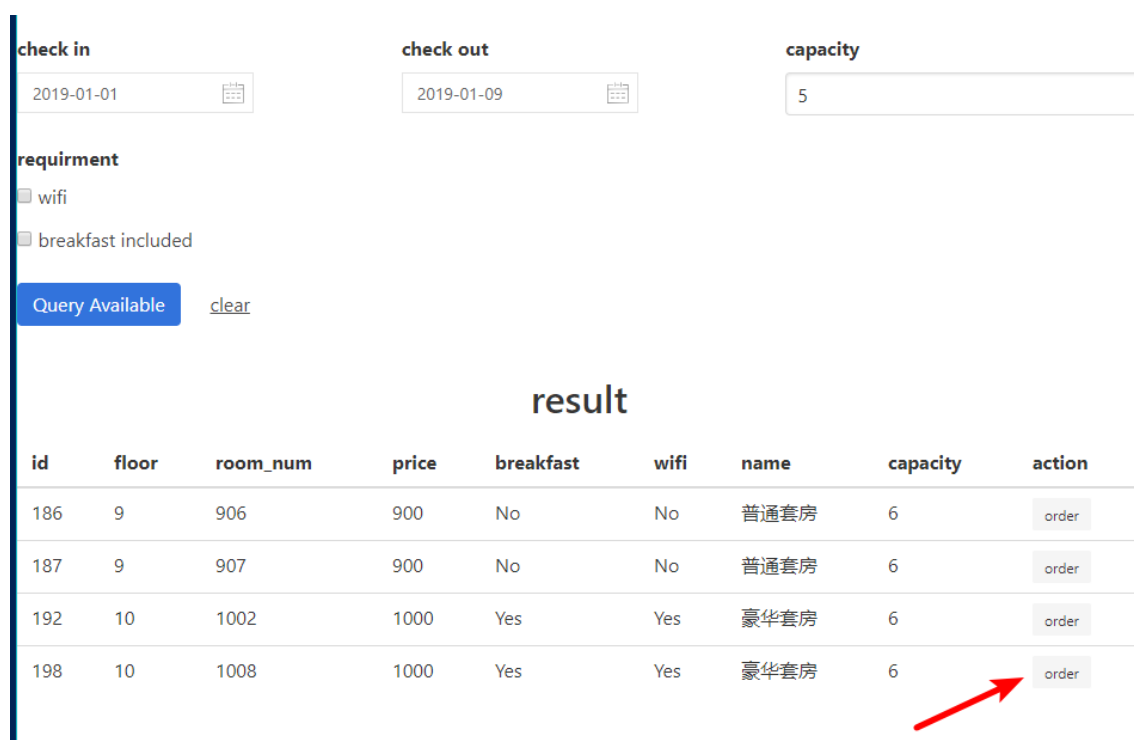
图 5.4 酒店经理使用界面

6 系统测试

在这一部分，我演示了一个在本系统中比较典型的操作，用户预订房间与查询取消订单，更详细的说明可见报告附件中的屏幕录像文件。

6.1 用户预订房间

一个用户名为“123”的用户要预订“2019-01-01” - “2019-01-09”的至少能住 5 人的房间，因此他打开了该酒店房间预订系统，并使用用户名“123”，密码“123”登录进入系统，并选择对应的房间。



result								
id	floor	room_num	price	breakfast	wifi	name	capacity	action
186	9	906	900	No	No	普通套房	6	order
187	9	907	900	No	No	普通套房	6	order
192	10	1002	1000	Yes	Yes	豪华套房	6	order
198	10	1008	1000	Yes	Yes	豪华套房	6	order

图 6.1 预订房间

当看到返回了‘ok’信息后，便是预订成功了，按下“Orders”按钮，选好指定日期之后，就可以查看刚刚下的订单，可见图 6.2。

可以看到，由于定了 9 天，因此系统生成的订单里也的确生成了 $9 \times 1000 = 9000$ 元的订单，同时用户的余额少了 9000，可见图 6.3。

Rooms

Orders

logout

profile

user name

begin date

end date

123

2019-01-01

2019-01-11

Query

clear

INFO

ok

result

id	status	check_in	check_out	room_id	user_id	fee	action
136	1	2019-01-07T16:00:00.000Z	2019-01-07T16:00:00.000Z	185	11	1000	Cancel
237	1	2019-01-08T16:00:00.000Z	2019-01-08T16:00:00.000Z	189	11	1000	Cancel
1538	0	2018-12-31T16:00:00.000Z	2019-01-08T16:00:00.000Z	198	11		
1539	0	2018-12-31T16:00:00.000Z	2019-01-08T16:00:00.000Z	198	11	1000	
1540	1	2018-12-31T16:00:00.000Z	2019-01-08T16:00:00.000Z	192	11	9000	Cancel

图 6.2 查看订单

Edit

Back

Username

Real Name

User ID

123

董丽华

11

Credential

Gender

Phone

663400473678154223

Female

13593611501

balance

bonus

birthdate

972999

999999999

2018-12-22

图 6.3 用户个人信息

7 开发所遇到的问题与解决方案

在开发本数据库系统的过程中，我们遇到了很多问题，例如对可能改变数据库数据的一致性的用户请求该如何处理？在数据库增加一项如何自动生成 id 等。对这些开发细节的说明与解决，我们在下面分成了几个小节分别说明。

7.1 如何保证数据一致性？

在一些操作中，对于用户可能仅仅是一个点击的操作，但是在系统后台执行该请求时，往往需要涉及到多个对数据库的写操作。就以用户请求预定房间为例，该操作要求更新用户余额，更新 Order 表和 Operation 表等操作。

为了保证数据的一致性，需要将这些操作以事务的方式进行操作。在我们的实现中，有两种方式来实现事务，一种是在后端的代码中，使用数据库连接器提供的事务接口实现事务，可见图 7.1。

在另一种实现中，我们还编写了多个存储过程，在存储过程中使用“start transaction”，与“commit”语句实现事务操作，可见节 4.2。

```
console.log(req.body)
let query: string = 'update `Order` as O set status = 0
let query2 : string = 'insert into Operation(time, detail
, 2, ? );'
let arg : string[] = []
try {
  if (order_id == '') {
    throw "order_id is empty or underfined!!"
  }
  arg.push(order_id)
  pool.getConnection()
  .then(conn => {
    conn.beginTransaction()
    .then(() => {
      conn.query(query, arg);
      return conn.query(query2, arg);
    })
    .then(() => {
      conn.commit();
      res.json({
        'error_code': 0,
        'error_msg': 'ok'
      })
    })
  })
}
```

执行多条sql语句

图 7.1 后端实现事务

7.2 如何在查询订单时查询订单相关的操作明细?

本次实验中网站后端使用的 JavaScript 语言采用了内建多个消息循环的执行模式向数据库发送一个查询请求仅仅是将这个查询请求“登记”到消息循环中，查询异步执行，程序将立即返回。因此，如果想在执行查询一个订单的查询语句中，直接嵌套另一个循环来查询操作明细是不可行的。解决方法有两种：

- 使用事务机制，确保执行完所有语句后程序才返回
- 前端首先调用一次查询订单操作，再依据订单 id 查询操作明细

第一种方法更适合用来保证一致性。因此我们采用了第二种方法。

7.3 用户权限的控制

关于不同用户权限控制的实现，我们曾经有过争论，是在前后端层面处理用户的权限，还是使用数据库的用户权限机制来对不同用户的访问做限制呢？最终我们的方案采取了在前后端对用户的权限进行处理。在后端，我们使用了 session 中的信息进行了用户信息的检查，代码可见图 7.2，通过检查用户信息，能够做到本用户只能查看本用户的信息，在其他接口中，也是用了类似的方法对权限进行了检查，以防止没有登录的人，或者权限较低的用户通过直接访问 api 的方式获取数据库的数据。

```
33 app.all('/api/alter_user_info', (req: Request, res: Response) => {  
34   try {  
35     let user_id: number = parseInt(req.body.user_id);  
36     if (user_id !== req.session.user_id) {  
37       throw('you are not user ' + user_id);  
38     }  
39     let credential: string = req.body.credential;  
40     let name: string = req.body.name;
```

图 7.2 用户权限控制在后端的实现

7.4 存储过程的传参和返回值

存储过程相当于是存储并编译在数据库中的一段子程序。它在传参和返回值上和一般的查询语句都有所不同：

在查询房间时，用户如果不对是否提供 wifi、早餐做出要求，那么就应该既返回提供 wifi 的房间，又返回不提供 wifi 的房间。我们发现可以使用 like 谓词实现这个需求：*where wifi like '%'*。但在把这条查询语句写入存储过程时发生了问题，实验发现：存储过程在传参过程中不允许参数发生隐式类型转换，如果把参数

Arg_wifi 声明为 bool，那么刚刚这条查询语句就会出错，因此必须将参数 Arg_wifi 声明为 char(1)。

此外存储过程的返回值在后端中是 list of list，而一般的查询语句的返回值是 list。这一点也要特殊处理。

7.5 NULL 与数字比较

实验发现：如果一个属性 num 的值为 null，那么它或它的与其他数字运算的结果出现在 if 语句中和其他数字比较，if 语句均会返回 false。因此在判断用户余额是否足够订房时最好要检查余额是否为 null，避免由于数据库数据上的失误出现部分用户可以无限订房的情况。