

数据库系统实验 2 设计报告

高梓源 2019K8009929026

肖展琪 2019K8009926014

桂庭辉 2019K8009929019

1 设计总述

经过设计与讨论，我们总结出如下的结论：优化数据库的依赖设计和应用层算法优化的考虑往往是分道扬镳的。优化数据库的存储模式与依赖关系往往倾向于简化其存储空间，会带来数据库的存储效率的优化，而基于应用层算法与数据结构的优化思路往往是采用更加丰富的对象模式去解决实际问题，后者会明显带来额外的依赖关系与空间开销，但换取的是时间复杂度的降低，应用性能的优化。

最开始设计阶段我们倾向于面向应用需求进行冗余设计，牺牲空间复杂度而换取时间复杂度的思路，但后来基于课程需求进行了方向的调整。事实上对存储效率和应用性能追求的背道而驰导致编程者必须做出权衡 (trade-off)。在本实验中的体现是将本准备存储称为表，描述城市-火车联系的特有数据结构改写成视图，并没有丢弃，而是采用运行时计算的方式，做出了一定的妥协，在数据集确定不更改的情况下，只会运行一次查询，性能可以接受，但是在实际应用中所带来的性能消耗甚为严重。

经过一系列的依赖去除和权衡，最终只保留了一个底线设计：在 BFS 或者 DFS 失败，未找到路径的情况下，搜索的开销极大，特别是 DFS 将无法停止，进行回溯。于是特别需要一个数据结构能够帮助在算法运行开始前判断是否能够找到路径，在此实验我们采用可达表，估量可直达和换乘一次可达的城市关系。又因为在数据预处理阶段生成可达表比专门使用查询形成可达表容易更多，因此我们特地将它放入 city 表作为一项冗余关系，但带来极大的编程上的便捷和性能开销的优化。

还有值得叙述的是编程思想的转变，sql 编程不同于一般问题求解级别的高级语言编程，后者往往有面向对象和面向过程两种编程思想，但是前者主要对象并非冯诺伊曼结构计算机模型，而是 NUMA 架构下文件系统存储，因此可称为面向存储结构编程。从后续代码中可见编程者仍然未能完成从面相对象/面向过程到面向存储结构的转变，控制流语句仍然很多，事实上大多都可以通过表间连接等操作转化，并且性能通常比多次查询更优。

2 模型设计

2.1 ER 图与范式分析

ER 图见下页图1。

2.1.1 users

候选键：uid, user_name, tel_num。作为非键属性的 password 与 email 没有其他的依赖关系。故 users 满足 BCNF。

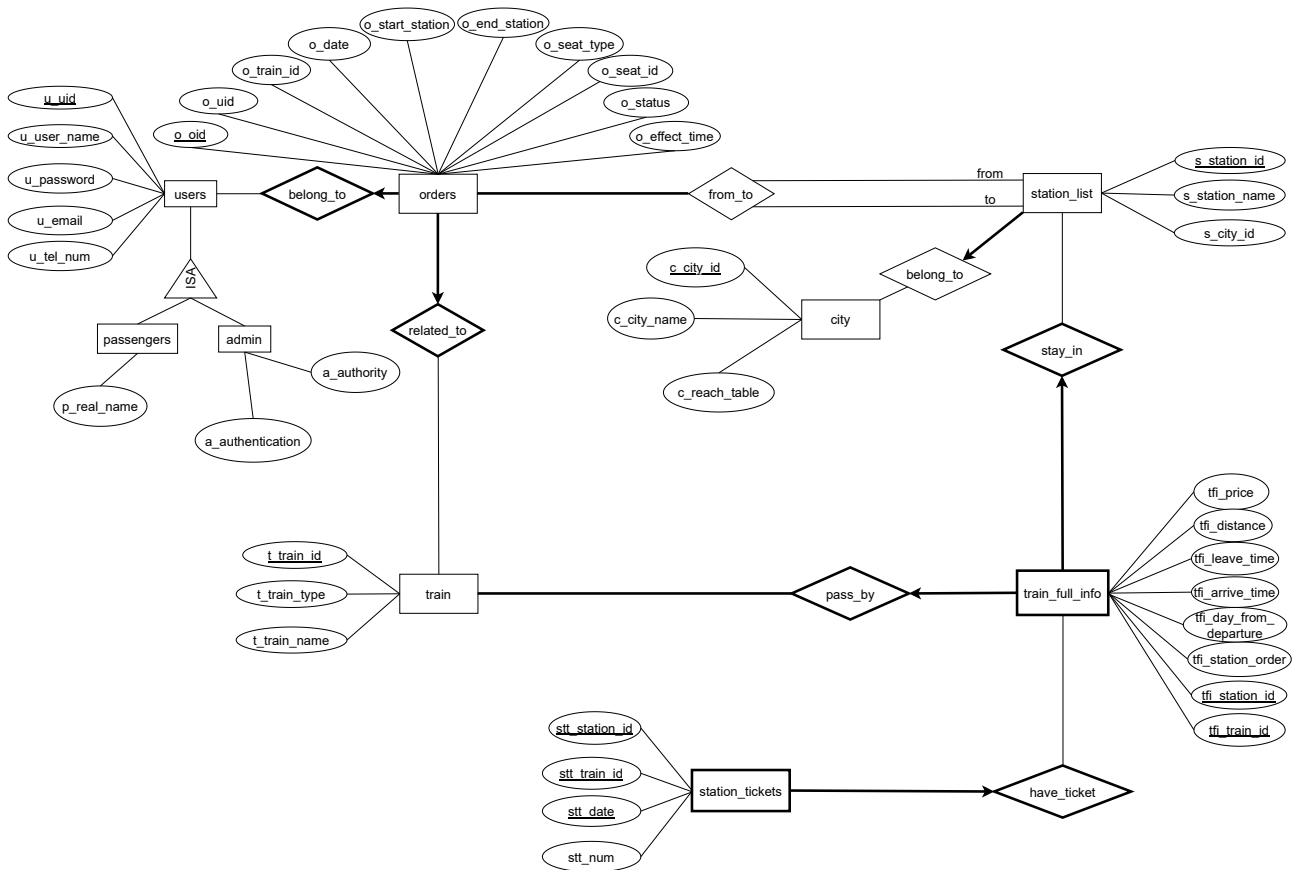


图 1: 火车订票系统 ER 图

passengers 中候选键为 pid，函数依赖关系仅有 $pid \rightarrow real_name$ ，故 passengers 满足 BCNF。

admin 中候选键为 aid，函数依赖关系仅有 $aid \rightarrow authority authentication$ ，故 admin 满足 BCNF，admin 登录时需要进行二次验证，因此设置了 authentication 域，此外使用 authority 域来表示该管理员的权限范围，后期可以进行扩展。

此处我们采用类层次的设计方式，由于需要明确区分 passengers 和 admin 两种身份，他们所需要的属性有部分重合也有部分是独有的。于是采用 ISA 表示实体集之间的继承关系，将其公共信息提取成名为 users 的共有结构。

```

1  create table if not exists users (
2      u_uid      serial primary key,
3      u_user_name varchar(20) unique,
4      u_password varchar(20) not null,
5      u_email    varchar(20) not null,
6      u_tel_num  integer[11] unique
7  );
8
9  create table if not exists passengers (
10     p_pid  integer not null,
11     p_real_name varchar(20) not null,
12     primary key (p_pid),
13     foreign key (p_pid) references users (u_uid)
14 );
15
16 create table if not exists admin (

```

```

17     a_aid            integer not null,
18     a_authentication varchar(20) not null ,
19     a_authority      admin_authority not null ,
20     primary key (a_aid),
21     foreign key (a_aid) references users (u_uid)
22 );

```

2.1.2 orders

主键为 oid，函数依赖关系仅有由 oid 确定其他非键属性，故 orders 满足 BCNF。

```

1 create table if not exists orders (
2     o_oid            serial primary key,
3     o_uid            integer,
4     o_train_id       integer not null,
5     o_date           date not null,
6     o_start_station integer not null,
7     o_end_station    integer not null,
8     o_seat_type       seat_type not null,
9     o_seat_id         integer not null,
10    o_status          order_status not null,
11    o_effect_time     timestamp not null,
12    foreign key (o_uid) references users (u_uid),
13    foreign key (o_train_id) references train (t_train_id),
14    foreign key (o_start_station) references station_list (s_station_id),
15    foreign key (o_end_station) references station_list (s_station_id)
16 );

```

2.1.3 train

候选键包括 train_id 与 train_name，二者均可确定 train_type，没有非平凡的依赖关系，故 train 满足 BCNF。

```

1 create table if not exists train (
2     t_train_id       serial primary key,
3     t_train_type      varchar(1) not null,
4     t_train_name      varchar(10) not null
5 );

```

2.1.4 city

主键为 city_id，其可唯一确定 city_name。此处引入的 reach_table，用于记录城市间在本实验需求下的可达关系，可通过 train、train_full_info、station_list 等信息推导得到，在存储上构成了一定冗余，但用于实现需求 5 时有助于查找两地间车次的时间开销，后文讨论需求时会详细讨论。除 reach_table 外该表满足 BCNF。

```

1 create table if not exists city (
2     c_city_id        integer primary key,
3     c_city_name       varchar(20) not null,
4     c_reach_table     boolean[]
5 );

```

2.1.5 station_list

候选键包括 station_id, station_name, 二者均可确定 city_id, 没有非平凡的依赖关系, 故 station_list 满足 BCNF。

```
1 create table if not exists station_list (  
2     s_station_id      serial primary key,  
3     s_station_name    varchar(20) not null,  
4     s_station_city_id integer    not null,  
5     foreign key (s_station_city_id) references city (c_city_id)  
6 );
```

2.1.6 train_full_info

该表记录每次列车每经停站的信息, 主键为 (train_id, station_id), 每个条目中其他信息可由该二元组确定。由于存在跨天运行的列车, 故而无法仅通过 train_id 与 arrive_time 或 leave_time 确定所有信息。故 train_full_info 满足 BCNF。

在获取数据时, 每车次列车初始域 day_from_departure 置 0, 列车运行时间每次跨越午夜零点则域 day_from_departure 自增 1, 由此记录列车准确的运行历时与进出站时间。

tfi_price 事实上也是较为独立随机的信息, 并没有必要把金额单独独立成表, 因此附加在此处。已知总共有 7 种座位, 为方便索引、替换等操作, 将其视为 7 个元素的 integer array, 而后可利用 postgresql 内置和自己添加的 array 函数进行处理。

```
1 create table if not exists train_full_info (  
2     tfi_train_id      integer,  
3     tfi_station_id    integer,  
4     tfi_station_order integer    not null,  
5     tfi_arrive_time    time      not null,  
6     tfi_leave_time    time      not null,  
7     tfi_day_from_departure integer not null,  
8     tfi_distance      integer    not null,  
9     tfi_price          decimal(5, 1)[7] not null default array [0.0, 0.0, 0.0, 0.0, 0.0,  
10    ↪ 0.0, 0.0],  
11     primary key (tfi_train_id, tfi_station_id),  
12     foreign key (tfi_train_id) references train (t_train_id),  
13     foreign key (tfi_station_id) references station_list (s_station_id)  
14 );
```

2.1.7 station_tickets

该表记录某站某日经停的某车各类型的余票, 主键为 (train_id, station_id, date), 没有非平凡的依赖关系, station_tickets 满足 BCNF。

stt_num 的处理和上小节 2.1.6 的 tfi_price 相同, 视为数组。

```
1 create table if not exists station_tickets (  
2     stt_station_id integer,  
3     stt_train_id   integer,  
4     stt_date       date      not null,  
5     stt_num        integer[7] not null default array [5, 5, 5, 5, 5, 5, 5],  
6     primary key (stt_station_id, stt_train_id),
```

```

7     foreign key (stt_station_id) references station_list (s_station_id),
8     foreign key (stt_train_id) references train (t_train_id),
9     foreign key (stt_station_id, stt_train_id) references train_full_info (tfi_station_id,
    ↪   tfi_train_id)
10 );

```

2.2 关系模式

参考 TPCB 文档可画出如下关系模式图：

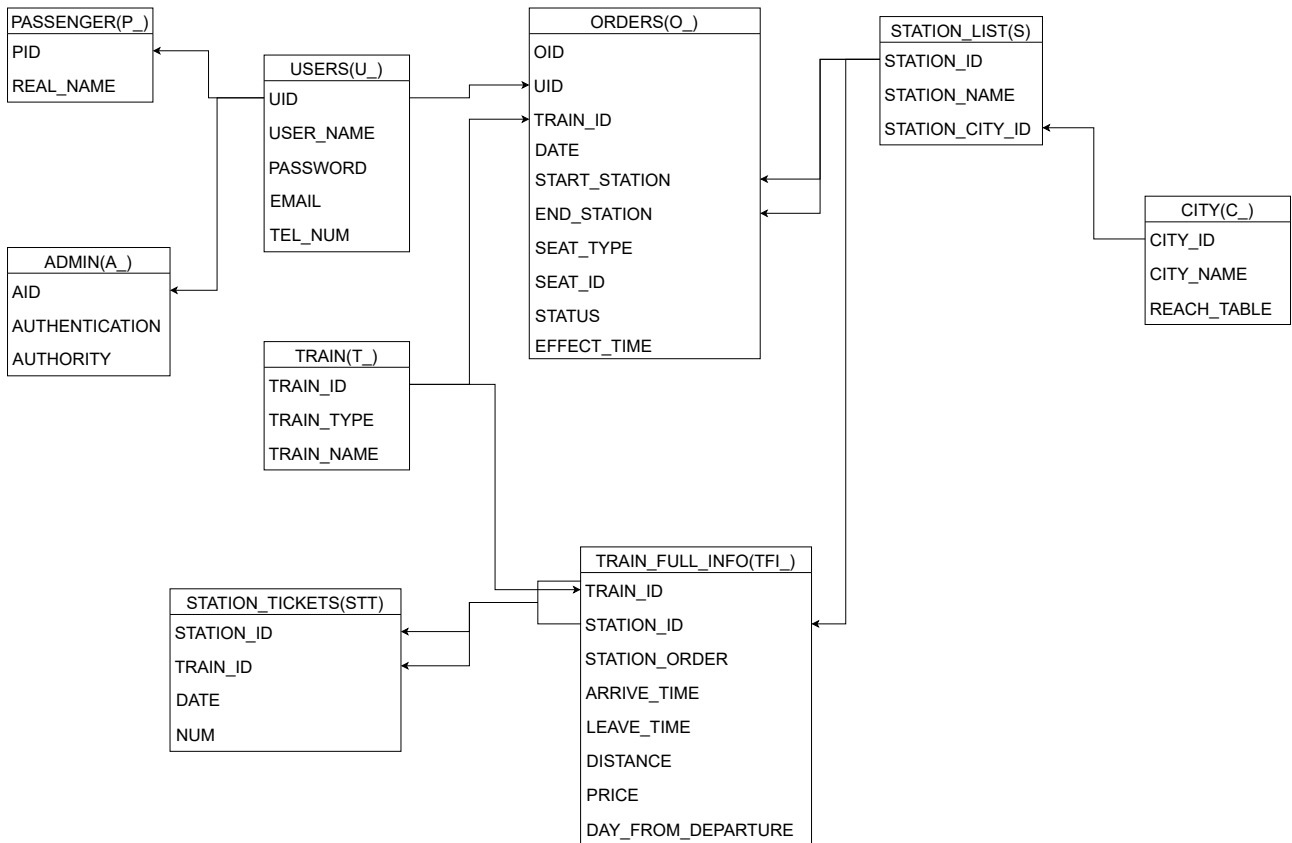


图 2: 火车订票系统关系模式图

具体的关系模式可见上一小节中的 `create table` 语句，其中涉及到的自定义枚举类型补充如下：

```

1 create type seat_type as enum ('YZ', 'RZ', 'YW_S', 'YW_Z', 'YW_X', 'RW_S', 'RW_X');
2 create type order_status as enum ('COMPLETE', 'PRE_ORDERED', 'CANCELED');
3 create type admin_authority as enum ('ALL');

```

psql 中目前还没有查到宏的实现，枚举类型的一大优点在于可以充当统一接口宏的作用。

3 需求实现

3.1 需求 1~3：记录相关信息

需求 1 要求记录每车次列车相关信息，可通过 2.1.3、2.1.5、2.1.6 中讨论的表记录。

需求 2 要求记录列车座位情况，可通过 2.1.7 中讨论的表与上述列车信息记录。

需求 3 中对乘客信息的记录部分可通过 2.1.1 中 users 与 passengers 表记录。注册与登录操作对应 sql 语句如下，其中对密码的加密工作交由前端 php 完成。

```
1  /* user registration */
2  /* @param: user_name */
3  /*       : user_password */
4  /*       : phone_num */
5  /*       : user_email */
6  /* @return: uid */
7  begin
8      select * into uid from insert_all_info_into_u__(user_name, user_password, phone_num,
9      ↪ user_email);
10
11 end;
12
13 /* user login */
14 /* @param: user_name */
15 /*       : user_password */
16 /* @return: uid */
17 /*       : error */
18 begin
19     select * from query_uid_from_uname_password_u__(user_name, user_password) into uid, error;
20 end;
```

上述代码中使用到的两个函数的 sql 实现如下：

```
1  /* query_uid_from_uname_password_u__ */
2  /* @param: user_name */
3  /*       : user_password */
4  /* @return: uid */
5  /*       : error_type */
6  begin
7      if (select * from users where u_user_name = user_name) is null then
8          uid := 0;
9          error := 'ERROR_NOT_FOUND_UNAME';
10     else
11         select u_uid into uid from users where u_user_name = user_name and u_password =
12         ↪ user_password;
13         if uid is null then
14             uid := 0;
15             error := 'ERROR_NOT_CORRECT_PASSWORD';
16         else
17             error := 'NO_ERROR';
18         end if;
19     end if;
20 end;
21
22 /* insert_all_info_into_u__ */
23 /* @param: user_name */
24 /*       : user_password */
25 /*       : phone_num */
26 /*       : user_email */
27 /* @return: uid */
28 /*       : err */
29 begin
30     if (select * from users where u_user_name = user_name) is not null then
31         uid := 0;
```

```

31     err := 'ERROR_DUPLICATE_UNAME';
32 else
33     if (select * from users where u_tel_num = phone_num) is not null then
34         uid := 0;
35         err := 'ERROR_DUPLICATE_U_TEL_NUM';
36     else
37         insert into users (u_user_name, u_password, u_email, u_tel_num)
38             values (user_name, user_password, user_email, phone_num);
39         select currval(pg_get_serial_sequence('users', 'u_uid')) into uid;
40         err := 'NO_ERROR';
41     end if;
42 end if;
43 end;

```

3.2 需求 4：查询具体车次

需求 1 中已根据车次记录列车信息，根据表 train、station_list、train_full_info 查询信息并组织正确的输出形式即可。

```

1  /*@param: train_name */
2  /*      : q_date */
3  begin
4      select query_train_id_from_name__t__(train_name) into train_id;
5      select query_start_time_from_id__tfi__(train_id) into start_time;
6      return query select tfi_station_order as station_order,
7                          s_station_name as station,
8                          s_station_id as station_id,
9                          c_city_name as city_name,
10                         c_city_id as city_id,
11                         tfi_arrive_time as arrive_time,
12                         tfi_leave_time as leave_time,
13                         (select * from get_actual_interval_bt_time(tfi_arrive_time,
14                             ⇨ tfi_leave_time, 0)) as stay_time,
15                         (select * from get_actual_interval_bt_time(start_time, tfi_arrive_time,
16                             ⇨ (select query_day_from_departure_from_id__tfi__(train_id,
17                                 ⇨ tfi_station_id)))) as durance,
18                         tfi_distance as distance,
19                         tfi_price as seat_price,
20                         stt_num as seat_num
21      from train_full_info tfi
22          left join station_list s on tfi.tfi_station_id = s.s_station_id
23          left join city c on s.s_station_city_id = c.c_city_id
24          left join train t on tfi.tfi_train_id = t.t_train_id
25          left join station_tickets stt on tfi.tfi_station_id =
26              ⇨ stt.stt_station_id
27      where t_train_id = train_id
28          and stt.stt_date = q_date;
29 end;
30
31 /* get_actual_interval_bt_time */
32 /* @param: start_time */
33 /*      : end_time */
34 /*      : days_added */

```

```

31  /* @return: actual_interval */
32  begin
33      if days_added = 0 and start_time > end_time then
34          actual_interval := interval '24 hours' + end_time - start_time;
35      else
36          actual_interval := (days_added || 'days')::interval + end_time - start_time;
37      end if;
38  end

```

此处使用的函数 `query_train_id_from_name__t__` 等均为简单的匹配查找，此处与后文均不详述此类函数。

3.3 需求 5、6：查询两地之间的车次与返程信息

需求 6 相对需求 5 仅交换起始城市，可交由前端完成，sql 逻辑复用需求 5 即可。

直达列车的查询逻辑较为简单，对起始站点查询此处离开的列车，遍历这些可能的列车，通过输入的出发时间过滤掉发车时间早于出发时间的列车，通过车次与城市即可确定可能的终点站，而后则是余票与价格的计算。余票与车次座位的设计维护在需求 7 中说明。

此处编程风格是较为明显的面向过程思想，使用中间变量来转化存储函数调用结果，而后对于表的组织也通过中间变量从而更加可读。但事实上此处一定存在面向存储结构的编程方案，一开始 `train_id_list` 并不作为 `array`，而是直接使用 `table` 提取到 `train_id` 后与其他相关表连接，中间也可通过函数调用组织相关信息，但总体结构仍然是面向表间关系。

```

1  /* @func: get_train_bt_cities_directly */
2  /* @param: from_city_id */
3  /*          : to_city_id */
4  /*          : q_date */
5  /*          : q_time */
6  begin
7      select check_reach_table(from_city_id, to_city_id) into city_reachable;
8      if city_reachable then
9          train_id_list := array(
10             select from_city_train.ct_train_id
11                 from city_train from_city_train
12                     join city_train to_city_train on from_city_train.ct_train_id =
13                                                         ⇨ to_city_train.ct_train_id
14                     where (select get_priority(from_city_id, from_city_train.ct_train_id)) <
15                             (select get_ct_priority(to_city_id, to_city_train.ct_train_id))
16             );
17      <<scan_train_list>>
18      foreach train_idi in array train_id_list
19          loop
20              -- 2 ways of accomplishment --
21              -- leave station --
22              select get_station_id_from_cid_tid(from_city_id, train_idi) into
23                  ⇨ station_leave_id;
24              select query_station_name_from_id_s__(station_leave_id) into station_leave_name;
25              select q_all_info_leave.leave_time,
26                     q_all_info_leave.day_from_departure,
27                     q_all_info_leave.distance,

```



```

26         q_all_info_leave.price
27     into station_leave_time, station_leave_day, station_leave_distance,
        ↳ station_arrive_price
28     from query_train_all_info_from_tid_sid__tfi__(train_idi, station_leave_id)
        ↳ q_all_info_leave;
29 -- check time --
30 if station_leave_time < q_time then
31     continue scan_train_list;
32 end if;
33 select query_train_name_from_id__t__(train_idi) into train_namei;
34 -- arrive station --
35 select get_station_id_from_cid_tid(to_city_id, train_idi) into station_arrive_id;
36 select query_station_name_from_id__s__(station_arrive_id) into
        ↳ station_arrive_name;
37 select q_all_info_arrive.leave_time,
38         q_all_info_arrive.day_from_departure,
39         q_all_info_arrive.distance,
40         q_all_info_arrive.price
41     into station_arrive_time, station_arrive_day, station_arrive_distance,
        ↳ station_leave_price
42     from query_train_all_info_from_tid_sid__tfi__(train_idi, station_arrive_id)
        ↳ q_all_info_arrive;
43 -- seats and price calculation --
44 for seat_i in 1..7
45     loop
46         select array_set(station_arrive_price, station_arrive_price[seat_i],
47                         station_arrive_price[seat_i] -
48                             ↳ station_leave_price[seat_i])
49                         into res_price;
49     end loop;
50 select get_min_seat.seat_num
51     into seat_nums
52     from get_min_seats(train_idi, q_date, station_leave_id, station_arrive_id,
53                     array ['YZ', 'RZ', 'YW_S', 'YW_Z', 'YW_X', 'RW_S',
54                         ↳ 'RW_X']) get_min_seat;
55 -- return row --
56 for r in
57     select train_namei as train_name,
58            train_idi as train_id,
59            station_leave_name as station_from_name,
60            station_leave_id as station_from_id,
61            station_arrive_name as station_to_name,
62            station_arrive_id as station_to_id,
63            station_leave_time as leave_time,
64            station_arrive_time as arrive_time,
65            (station_arrive_day - station_leave_day || 'days')::interval +
66                ↳ station_arrive_time -
67            station_leave_time as durance,
68            station_arrive_distance - station_leave_distance as distance,
69            res_price as seat_price,
70            seat_nums as seat_nums,
71            false as transfer_first,
72            false as transfer_late
73     loop

```

```

72         return next r;
73     end loop;
74 end loop;
75 end if;
76 return;
77 end;

```

对于换乘一次的情况，对于出发城市，查找其可直达的所有城市，遍历这些城市以它们为换乘城市，查找到目的城市的可能列车（即查询从换乘城市到目的城市的直达列车），事先通过 `city` 表中 `reach_table` 域记录城市间的可达与否可以在遍历换乘城市时避免与目的城市完全不可达的城市作为换乘城市时的查询操作（即上文函数 `get_train_bt_cities_directly` 开头的 `city_reachable` 检查），减小开销。

具体实现上类似 BFS，通过队列记录可能的换乘城市，遍历所有可能的情况直到清空队列。

以下代码指定 `query_transfer` 为真时查询换乘一次的结果，为否则调用上文函数查询直达列车。

此处为了 BFS 算法书写的方便仍然暂时先按照面向过程的风格书写。`src_city` 以 `array` 的数据结构存储每层查找的城市。此处更改为面向存储结构的风格较为简单，可以直接将所有经过源城市 `city_from` 之后的列车在之后经过的城市名查询得到，在和 `station_list` 与 `train_full_info` 进行连接，查找得到每站经过的车次，形成一个大表后与目的城市 `city_to` 经停列车比对，一次性得到所有换乘后半行程的列车信息。

```

1  /* @param: city_from */
2  /*      : city_to */
3  /*      : q_date */
4  /*      : q_time */
5  /*      : query_transfer */
6  begin
7      select query_city_id_from_name__c__(city_from) into from_city_id;
8      select query_city_id_from_name__c__(city_to) into to_city_id;
9      select check_reach_table(from_city_id, to_city_id) into city_reachable;
10     if city_reachable then
11         if not query_transfer then
12             for r in
13                 select * from get_train_bt_cities_directly(from_city_id, to_city_id, q_date,
14                     ↪ q_time)
15             loop
16                 return next r;
17             end loop;
18         else
19             -- first set of transfer trains must be ones passing from city --
20             -- so outside loop --
21             passing_trains := array(select query_train_id_list_from_cid__ct__(from_city_id));
22             while (select array_length(src_city, 1)) > 0 and (select array_position(src_city,
23                 ↪ to_city_id)) is not null
24             loop
25                 select array_length(src_city, 1) into current_level_city_num;
26                 for city_i in 1..current_level_city_num
27                 loop
28                     neighbour_city := array(select get_ct_next_city_list(src_city[1],
29                         ↪ passing_trains));
30                     src_city := array(array_cat(src_city, neighbour_city));
31                     -- initially from_city_id was in src_city --
32                     -- so remove it first because we have dealt with it --
33                     src_city := array(array_remove_elem(src_city, 1));

```

```

31         -- then src_city[1] is middle city to transfer --
32         if (select array_length(src_city, 1)) > 1 then
33             -- ... --
34         end if;
35     end loop;
36 end loop;
37 return;
38 end if;
39 end if;
40 end;

```

对于每个可能的换乘城市，遍历出发城市到其的每次直达列车，查询其到目的城市的每次直达列车，根据上游列车与下游列车是否在同站换乘检查不同的时间要求是否满足。

```

1  if (select array_length(src_city, 1)) > 1 then
2      for r in
3          (select *
4              from get_train_bt_cities_directly(from_city_id, src_city[1], q_date, q_time))
5      loop
6          for j in
7              (select *
8                  from get_train_bt_cities_directly(src_city[1], to_city_id, q_date, q_time
9                  ↪ + r.duranc + interval '1 hour'))
10         loop
11             if r.station_to_id = j.station_from_id then
12                 select *
13                     from get_actual_interval_bt_time(r.arrive_time, j.leave_time, 0)
14                     into transfer_interval;
15                 if transfer_interval >= interval '1 hour'
16                     and transfer_interval <= interval '4 hours'
17                 then
18                     return next r;
19                     return next j;
20                 end if;
21             else
22                 if transfer_interval >= interval '2 hours'
23                     and transfer_interval <= interval '4 hours'
24                 then
25                     return next r;
26                     return next j;
27                 end if;
28             end if;
29         end loop;
30     end loop;
31 end if;

```

3.4 需求 7：预订车次座位

余票与车次座位的设计上，我们在 `station_tickets` 表中记录某天某车在某站的各种类型余票数，每当乘客购买某天某一区间车票，则将该天该车该区间内（包括左端不包括右端）所有对应类型余票数减 1。查询时对某车某天某区间的某类型余票数则为该区间内每站该类型余票数的最小值。获取座位时先查找区间内合法的座位，其后更新 `station_tickets` 表内容。

```

1  /* try_occupy_seats */
2  /* @param: train_id */
3  /*      : order_date */
4  /*      : station_from_id */
5  /*      : station_to_id */
6  /*      : seat_type */
7  /*      : seat_num */
8  /* @return: succeed */
9  /*      : left_seat */
10 begin
11     select query_station_order_from_tid_sid__tfi__(train_id, station_from_id) into
        ↳ station_start_order;
12     station_order_ptr = station_start_order;
13     select query_station_order_from_tid_sid__tfi__(train_id, station_to_id) into
        ↳ station_end_order;
14     -- find min_seat --
15     select get_min_seat.seat_num
16         into min_seat
17         from get_min_seats(train_id, order_date, station_from_id, station_to_id, array
        ↳ [seat_type]) get_min_seat
18         where in_order = 1;
19     -- check satisfiability --
20     if min_seat < seat_num then
21         succeed := false;
22         left_seat := min_seat;
23     else
24         succeed := true;
25         left_seat := 5 - min_seat;
26         -- second loop, update station tickets --
27         while station_order_ptr != station_end_order
28             loop
29                 update station_tickets
30                 set stt_num = (select array_set(stt_num, seat_type, stt_num[seat_type] -
        ↳ seat_num))
31                 where stt_train_id = train_id
32                     and stt_station_id = station_id_ptr
33                     and stt_date = order_date;
34                 station_order_ptr := station_order_ptr + 1;
35                 select query_station_id_from_tid_so__tfi__(train_id, station_order_ptr) into
        ↳ station_id_ptr;
36             end loop;
37     end if;
38 end;

```

用户预定座位时，在确定区间车次日期等信息申请订单后即暂时拥有座位（如有），此时记订单状态为 PRE_ORDERED，点击确认后将订单置为 ORDERED。有关订单状态的维护将在需求 8 内讨论。

```

1  /* pre_order_train */
2  /* @param: train_id */
3  /*      : station_from_id */
4  /*      : station_to_id */
5  /*      : seat_type */
6  /*      : seat_num */
7  /*      : order_date */
8  /* @return: succeed */

```

```

9  /*      : seat_id */
10 /*      : order_id */
11 begin
12     select succeed, left_seat
13         into succeed, seat_id
14         from try_occupy_seats(train_id, order_date, station_from_id, station_to_id, seat_type,
15                               ↪ seat_num);
16     if succeed then
17         insert into orders (o_train_id, o_date, o_start_station, o_end_station, o_seat_type,
18                               ↪ o_seat_id,
19                               o_status, o_effect_time)
20         select train_id,
21                order_date,
22                station_from_id,
23                station_to_id,
24                seat_type,
25                seat_id,
26                'PRE_ORDERED',
27                now();
28         select currval(pg_get_serial_sequence('orders', 'o_oid')) into order_id;
29     else
30         order_id := 0;
31     end if;
32 end;
33
34 /* order_train_seats */
35 /* @param: order_id */
36 /*      : uid */
37 /* @return: succeed */
38 begin
39     update orders
40     set (o_uid, o_status) = (uid, 'ORDERED')
41     where o_oid = order_id;
42 end;

```

3.5 需求 8：查询订单和删除订单

订单相关信息记录在 orders 表内，给定用户、出发日期范围等信息后即可查询获得。

```

1 select o_oid as order_id,
2        t_train_name as train_name,
3        o_train_id as train_id,
4        s_start.s_station_name as station_leave,
5        o_start_station as station_id,
6        s_arrive.s_station_name as station_arrive,
7        tfi_start.tfi_leave_time as start_time,
8        tfi_end.tfi_arrive_time as arrive_time,
9        (select * from get_actual_interval_bt_time(tfi_start.tfi_leave_time, tfi_end.tfi_arrive_time,
10            tfi_start.tfi_day_from_departure - tfi_end.tfi_day_from_departure)) as durance,
11        tfi_end.tfi_distance - tfi_start.tfi_distance as distance,
12        o_seat_type as seat_type,
13        o_seat_id as seat_id,
14        o_status as status,
15        tfi_end.tfi_price[o_seat_type] - tfi_start.tfi_price[o_seat_type] + 5 as price

```

```

16     from orders
17         left join station_list s_start on orders.o_start_station = s_start.s_station_id
18         left join station_list s_arrive on orders.o_end_station = s_arrive.s_station_id
19         left join train on orders.o_train_id = train.t_train_id
20         left join train_full_info tfi_start on s_start.s_station_id =
21             ⇨ tfi_start.tfi_station_id
22         and orders.o_train_id = tfi_start.tfi_train_id
23         left join train_full_info tfi_end on s_start.s_station_id = tfi_end.tfi_station_id
24         and orders.o_train_id = tfi_end.tfi_train_id
25     where o_uid = uid
26     and o_date >= start_query_date
27     and o_date <= end_query_date;

```

我们将订单的状态分为三种：取消、预约、确认。用户发起申请后确认订单前订单为预约态，此时对订单的操作仅有点击确认完成订单进入确认态，30 分钟内未确认的订单将会自动老化删除。确认态的订单可被手动取消。确认操作的 sql 语句在需求 7 中已经给出，以下给出取消与老化操作的 sql 语句：

```

1  /* user_cancel sql */
2  /* @param: order_id */
3  /* @return: succeed */
4  begin
5      select o_train_id, o_date, o_start_station, o_end_station, o_seat_type
6          into train_id, order_date, start_station, end_station, seat_type
7          from orders
8          where o_oid = order_id
9          and o_status = 'COMPLETE';
10     select release_seats(train_id, order_date, start_station, end_station, seat_type, 1);
11     update orders
12     set o_status = 'CANCELED'
13     where o_oid = order_id;
14 end;
15
16 /* remove_outdated_order */
17 delete
18     from orders
19     where (select * from get_actual_interval_bt_time(orders.o_effect_time, now(), 0))
20         > interval '30 minutes'
21     and orders.o_status = 'PRE_ORDERED';

```

3.6 需求 9：管理员

管理员相关需求中，查看每个用户订单一项可复用需求 8 中查看订单的实现。查询当前注册用户列表的实现也较简单，对 passengers 表^①查询即可。

```

1  select p_pid as uid,
2         u_user_name as uname,
3         array(
4             select o_oid from orders where o_uid = p_pid
5         ) as orders
6  from passengers
7  left join users u on passengers.p_pid = u.u_uid;

```

对总订单数、总票价、最热点车次排序的查询涉及到订单的取消态，由于我们额外引入了预约态，此处约定这三处查询仅包括已确认的订单，不统计取消订单与已申请未确认的订单。

^①我们将所有用户 users 分为 passengers 和 admin 两类，约定管理员的查询功能为查询所有 passengers

```

1  -- 2 views to select top 10 train --
2  create or replace view top_10_train_tickets(train_id, count_num)
3  as
4  select o_train_id as train_id, count(*) as count_num
5      from orders
6      where o_status = 'COMPLETE'
7      group by o_train_id
8      order by count_num
9      limit 10;
10
11 create or replace view top_10_train_ids(train_id)
12 as
13 select train_id
14      from top_10_train_tickets;
15
16
17 /* admin_query_orders */
18 /* @return: total_order_num */
19 /*         : total_price */
20 /*         : hot_trains */
21 begin
22     select count(*),
23            sum(tfi_end.tfi_price[o_seat_type] - tfi_start.tfi_price[o_seat_type] + 5)
24     into total_order_num, total_price
25     from orders
26          left join train_full_info tfi_start on o_start_station =
27          ↪ tfi_start.tfi_station_id
28          and orders.o_train_id = tfi_start.tfi_train_id
29          left join train_full_info tfi_end on o_end_station = tfi_end.tfi_station_id
30          and orders.o_train_id = tfi_end.tfi_train_id
31          and orders.o_status = 'COMPLETE';
32     hot_trains := array(select t_train_name
33                        from train
34                        left join top_10_train_ids on train.t_train_id =
35                        ↪ top_10_train_ids.train_id);
36 end;

```

成员分工

高梓源：参与需求分析、算法设计、数据表设计，编写 sql 语句。

肖展琪：参与需求分析、算法设计、数据表设计，完成 ER 图、关系模式等宏观设计。

桂庭辉：参与需求分析、算法设计、数据表设计，撰写实验报告。