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。

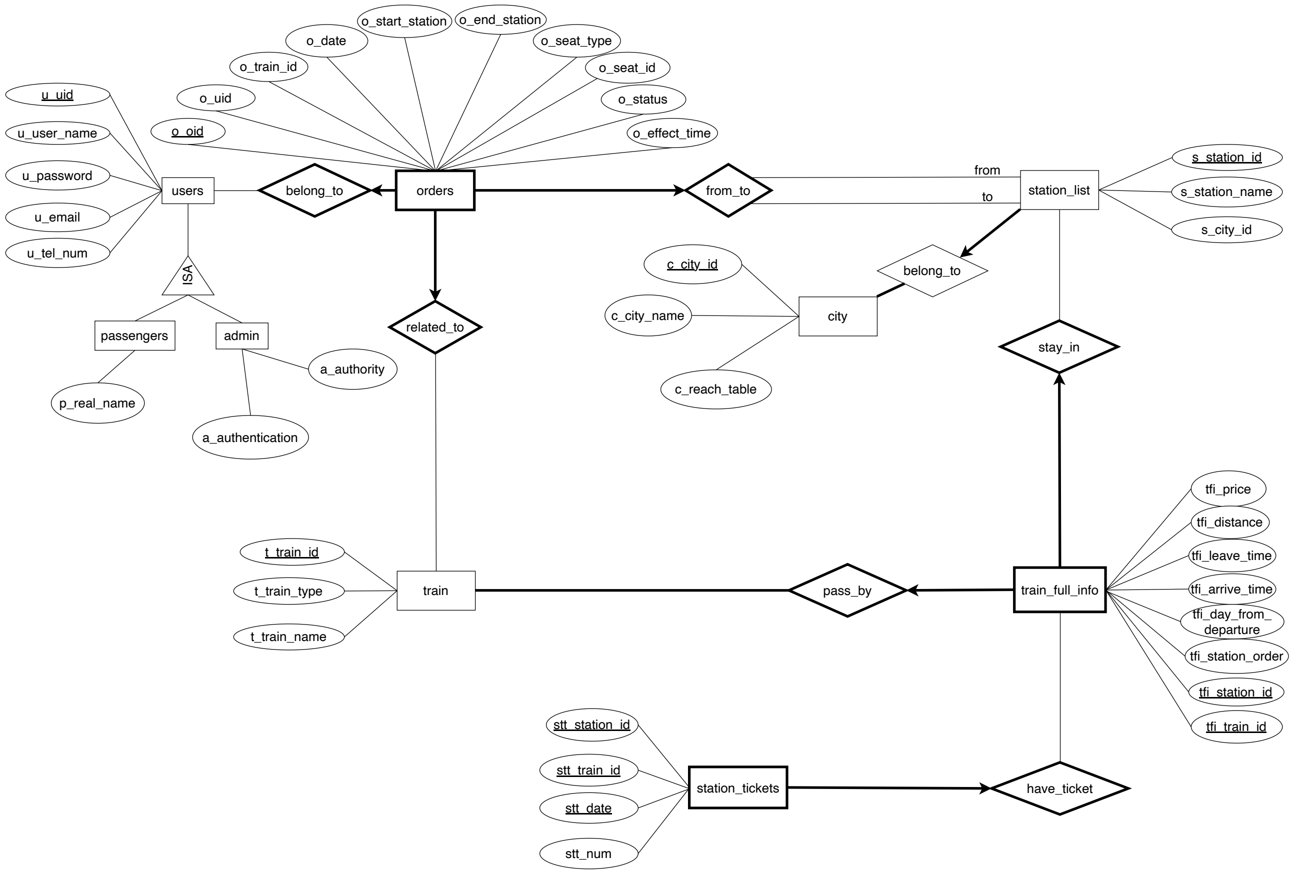

图 1: 火车订票系统 ER 图

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

admin 中候选键为 aid，函数依赖关系仅有 aid→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。

```
create table if not exists users (
    u_uid       serial primary key,
    u_user_name varchar(20) unique,
    u_password  varchar(20) not null,
    u_email     varchar(20) not null,
    u_tel_num   integer[11] unique
);

create table if not exists passengers (
    p_pid   integer not null,
    p_real_name varchar(20) not null,
    primary key (p_pid),
    foreign key (p_pid) references users (u_uid)
);

create table if not exists admin (
    a_aid            integer not null,
    a_authentication varchar(20) not null ,
    a_authority      admin_authority not null ,
    primary key (a_aid),
    foreign key (a_aid) references users (u_uid)
);
```

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。

```
create table if not exists orders (
    o_oid           serial primary key,
    o_uid           integer,
    o_train_id      integer      not null,
    o_date          date         not null,
    o_start_station integer      not null,
    o_end_station   integer      not null,
    o_seat_type     seat_type    not null,
    o_seat_id       integer      not null,
    o_status        order_status not null,
    o_effect_time   timestamp    not null,
    foreign key (o_uid) references users (u_uid),
    foreign key (o_train_id) references train (t_train_id),
    foreign key (o_start_station) references station_list (s_station_id),
    foreign key (o_end_station) references station_list (s_station_id)
);
```

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。

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

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。

```
create table if not exists city (
    c_city_id     integer primary key,
    c_city_name   varchar(20) not null,
    c_reach_table boolean[]
);
```

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。

```sql
create table if not exists station_list (
    s_station_id      serial primary key,
    s_station_name    varchar(20) not null,
    s_station_city_id integer     not null,
    foreign key (s_station_city_id) references city (c_city_id)
);
```

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。

```sql
create table if not exists train_full_info (
    tfi_train_id            integer,
    tfi_station_id          integer,
    tfi_station_order       integer          not null,
    tfi_arrive_time         time             not null,
    tfi_leave_time          time             not null,
    tfi_day_from_departure  integer          not null,
    tfi_distance            integer          not null,
    tfi_price               decimal(5, 1)[7] not null default array [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    primary key (tfi_train_id, tfi_station_id),
    foreign key (tfi_train_id) references train (t_train_id),
    foreign key (tfi_station_id) references station_list (s_station_id)
);
```

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。

```sql
create table if not exists station_tickets (
    stt_station_id integer,
    stt_train_id   integer,
    stt_date       date       not null,
    stt_num        integer[7] not null default array [5, 5, 5, 5, 5, 5, 5],
    primary key (stt_station_id, stt_train_id),
    foreign key (stt_station_id) references station_list (s_station_id),
    foreign key (stt_train_id) references train (t_train_id),
    foreign key (stt_station_id, stt_train_id) references train_full_info (tfi_station_id, tfi_train_id)
);
```

2.2 关系模式

参考 TPCH 文档可画出如下关系模式图：

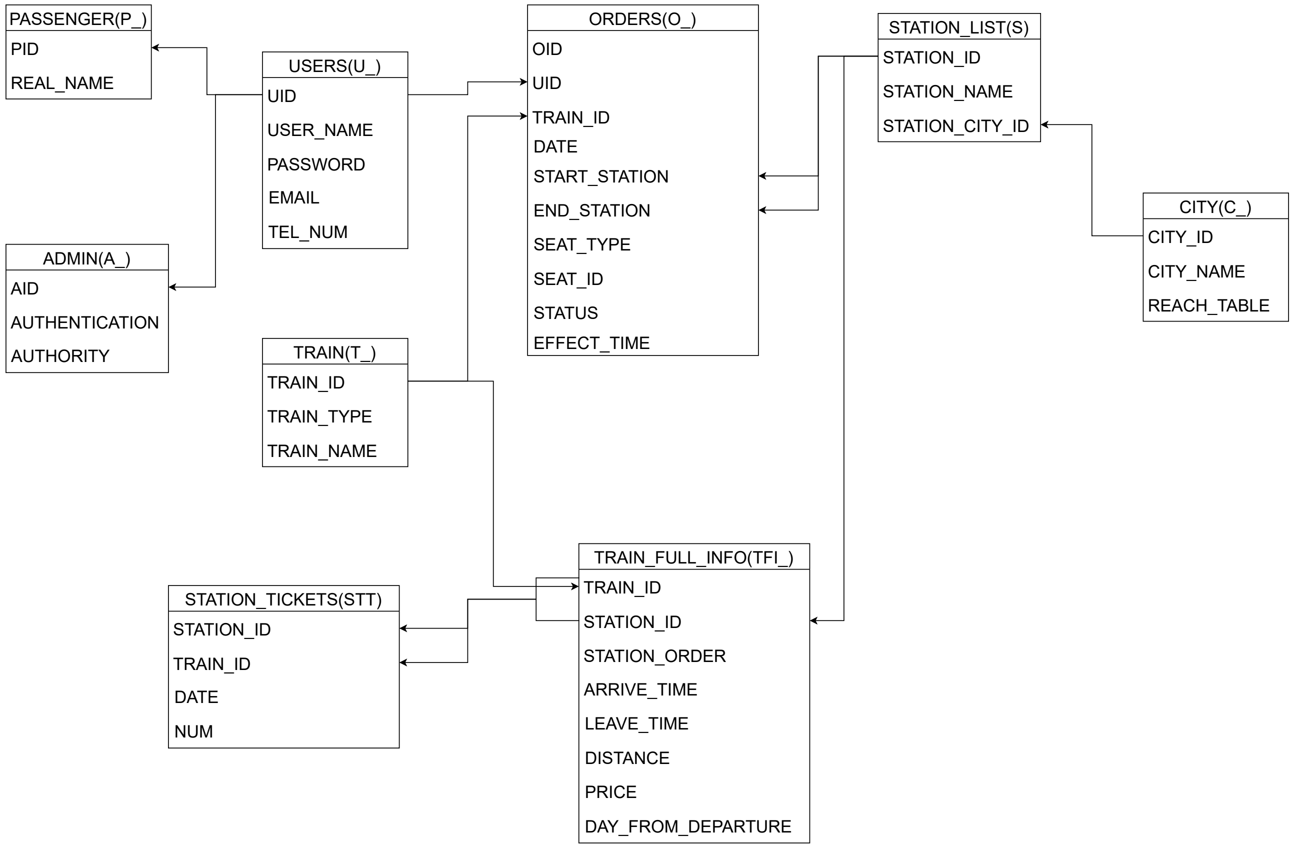

图 2: 火车订票系统关系模式图

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：

```
create type seat_type as enum ('YZ', 'RZ', 'YW_S', 'YW_Z', 'YW_X', 'RW_S', 'RW_X');
create type order_status as enum ('COMPLETE', 'PRE_ORDERED', 'CANCELED');
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。

```
/* user registration */
/* @param: user_name */
/*        : user_password */
/*        : phone_num */
/*        : user_email */
/* @return: uid */
begin
    select * into uid from insert_all_info_into__u__(user_name, user_password, phone_num,
    ↪  user_email);
end;

/* user login */
/* @param: user_name */
/*        : user_password */
/* @return: uid */
/*         : error */
begin
    select * from query_uid_from_uname_password__u__(user_name, user_password) into uid, error;
end;
```

上述代码中使用到的两个函数的 sql 实现如下：

```
/* query_uid_from_uname_password__u__ */
/* @param: user_name */
/*        : user_password */
/* @return: uid */
/*         : error_type */
begin
    if (select * from users where u_user_name = user_name) is null then
        uid := 0;
        error := 'ERROR_NOT_FOUND_UNAME';
    else
        select u_uid into uid from users where u_user_name = user_name and u_password =
        ↪  user_password;
        if uid is null then
            uid := 0;
            error := 'ERROR_NOT_CORRECT_PASSWORD';
        else
            error := 'NO_ERROR';
        end if;
    end if;
end;

/* insert_all_info_into__u__ */
/* @param: user_name */
/*        : user_password */
/*        : phone_num */
/*        : user_email */
/* @return: uid */
/*         : err */
begin
    if (select * from users where u_user_name = user_name) is not null then
        uid := 0;
        err := 'ERROR_DUPLICATE_UNAME';
    else
        if (select * from users where u_tel_num = phone_num) is not null then
            uid := 0;
            err := 'ERROR_DUPLICATE_U_TEL_NUM';
        else
            insert into users (u_user_name, u_password, u_email, u_tel_num)
                values (user_name, user_password, user_email, phone_num);
            select currval(pg_get_serial_sequence('users', 'u_uid')) into uid;
            err := 'NO_ERROR';
        end if;
    end if;
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。

```
/*@param: train_name  */
/*      : q_date */
begin
    select query_train_id_from_name__t__(train_name) into train_id;
    select query_start_time_from_id__tfi__(train_id) into start_time;
    return query select tfi_station_order as station_order,
                        s_station_name as station,
                        s_station_id as station_id,
                        c_city_name as city_name,
                        c_city_id as city_id,
                        tfi_arrive_time as arrive_time,
                        tfi_leave_time as leave_time,
                        (select * from get_actual_interval_bt_time(tfi_arrive_time,
                        ↪   tfi_leave_time, 0)) as stay_time,
                        (select * from get_actual_interval_bt_time(start_time, tfi_arrive_time,
                        ↪   (select query_day_from_departure_from_id__tfi__(train_id,
                        ↪   tfi_station_id)))) as durance,
                        tfi_distance as distance,
                        tfi_price as seat_price,
                        stt_num as seat_num
                     from train_full_info tfi
                              left join station_list s on tfi.tfi_station_id = s.s_station_id
                              left join city c on s.s_station_city_id = c.c_city_id
                              left join train t on tfi.tfi_train_id = t.t_train_id
                              left join station_tickets stt on tfi.tfi_station_id =
                              ↪   stt.stt_station_id
                     where t_train_id = train_id
                       and stt.stt_date = q_date;
end;

/* get_actual_interval_bt_time */
/* @param: start_time */
/*       : end_time */
/*       : days_added */
/* @return: actual_interval */
begin
    if days_added = 0 and start_time > end_time then
        actual_interval := interval '24 hours' + end_time - start_time;
    else
^^I     actual_interval := (days_added || 'days')::interval + end_time - start_time;
    end if;
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。

```
/* @func: get_train_bt_cities_directly */
/* @param: from_city_id */
/*         : to_city_id */
/*         : q_date */
/*         : q_time */
begin
    select check_reach_table(from_city_id, to_city_id) into city_reachable;
    if city_reachable then
        train_id_list := array(
                select from_city_train.ct_train_id
                    from city_train from_city_train
                             join city_train to_city_train on from_city_train.ct_train_id =
                             ↪  to_city_train.ct_train_id
                    where (select get_ct_priority(from_city_id, from_city_train.ct_train_id)) <
                          (select get_ct_priority(to_city_id, to_city_train.ct_train_id))
            );
        <<scan_train_list>>
        foreach train_idi in array train_id_list
            loop
                -- 2 ways of accomplishment --
                -- leave station --
                select get_station_id_from_cid_tid(from_city_id, train_idi) into
                ↪  station_leave_id;
                select query_station_name_from_id__s__(station_leave_id) into station_leave_name;
                select q_all_info_leave.leave_time,
                       q_all_info_leave.day_from_departure,
                       q_all_info_leave.distance,
                       q_all_info_leave.price
                    into station_leave_time, station_leave_day, station_leave_distance,
                    ↪  station_arrive_price
                    from query_train_all_info_from_tid_sid__tfi__(train_idi, station_leave_id)
                    ↪  q_all_info_leave;
                -- check time --
                if station_leave_time < q_time then
                    continue scan_train_list;
                end if;
                select query_train_name_from_id__t__(train_idi) into train_namei;
                -- arrive station --
                select get_station_id_from_cid_tid(to_city_id, train_idi) into station_arrive_id;
                select query_station_name_from_id__s__(station_arrive_id) into
                ↪  station_arrive_name;
                select q_all_info_arrive.leave_time,
                       q_all_info_arrive.day_from_departure,
                       q_all_info_arrive.distance,
                       q_all_info_arrive.price
                    into station_arrive_time, station_arrive_day, station_arrive_distance,
                    ↪  station_leave_price
                    from query_train_all_info_from_tid_sid__tfi__(train_idi, station_arrive_id)
                    ↪  q_all_info_arrive;
                -- seats and price calculation --
                for seat_i in 1..7
                    loop
                        select array_set(station_arrive_price, station_arrive_price[seat_i],
                                         station_arrive_price[seat_i] -
                                         ↪  station_leave_price[seat_i])
                            into res_price;
                    end loop;
                select get_min_seat.seat_num
                    into seat_nums
                    from get_min_seats(train_idi, q_date, station_leave_id, station_arrive_id,
                                       array ['YZ', 'RZ', 'YW_S', 'YW_Z', 'YW_X', 'RW_S',
                                       ↪  'RW_X']) get_min_seat;
                -- return row --
                for r in
                    select train_namei as train_name,
                           train_idi as train_id,
                           station_leave_name as station_from_name,
                           station_leave_id as station_from_id,
                           station_arrive_name as station_to_name,
                           station_arrive_id as station_to_id,
                           station_leave_time as leave_time,
                           station_arrive_time as arrive_time,
                           (station_arrive_day - station_leave_day || 'days')::interval +
                           ↪  station_arrive_time -
                           station_leave_time as durance,
                           station_arrive_distance - station_leave_distance as distance,
                           res_price as seat_price,
                           seat_nums as seat_nums,
                           false as transfer_first,
                           false as transfer_late
                    loop
```

```
                        return next r;
                    end loop;
            end loop;
    end if;
    return;
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。

```
/* @param: city_from */
/*       : city_to */
/*       : q_date */
/*       : q_time */
/*       : query_transfer */
begin
    select query_city_id_from_name__c__(city_from) into from_city_id;
    select query_city_id_from_name__c__(city_to) into to_city_id;
    select check_reach_table(from_city_id, to_city_id) into city_reachable;
    if city_reachable then
        if not query_transfer then
            for r in
                select * from get_train_bt_cities_directly(from_city_id, to_city_id, q_date,
                ↪   q_time)
                loop
                    return next r;
                end loop;
        else
            -- first set of transfer trains must be ones passing from city --
            -- so outside loop --
            passing_trains := array(select query_train_id_list_from_cid__ct__(from_city_id));
            while (select array_length(src_city, 1)) > 0 and (select array_position(src_city,
            ↪   to_city_id)) is not null
                loop
                    select array_length(src_city, 1) into current_level_city_num;
                    for city_i in 1..current_level_city_num
                        loop
                            neighbour_city := array(select get_ct_next_city_list(src_city[1],
                            ↪   passing_trains));
                            src_city := array(select array_cat(src_city, neighbour_city));
                            -- initially from_city_id was in src_city --
                            -- so remove it first because we have dealt with it --
                            src_city := array(select array_remove_elem(src_city, 1));
                            -- then src_city[1] is middle city to transfer --
                            if (select array_length(src_city, 1)) > 1 then
                                -- ... --
                            end if;
                        end loop;
                end loop;
            return;
        end if;
    end if;
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。

```
if (select array_length(src_city, 1)) > 1 then
    for r in
        (select *
                from get_train_bt_cities_directly(from_city_id, src_city[1], q_date, q_time))
        loop
            for j in
                (select *
                        from get_train_bt_cities_directly(src_city[1], to_city_id, q_date, q_time + r.durance + interval '1 hour'))
                loop
                    if r.station_to_id = j.station_from_id then
                        select *
                            from get_actual_interval_bt_time(r.arrive_time, j.leave_time, 0)
                            into transfer_interval;
                        if transfer_interval >= interval '1 hour'
                            and transfer_interval <= interval '4 hours'
                        then
                            return next r;
                            return next j;
                        end if;
                    else
                        if transfer_interval >= interval '2 hours'
                            and transfer_interval <= interval '4 hours'
                        then
                            return next r;
                            return next j;
                        end if;
                    end if;
                end loop;
        end loop;
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。

```
/* try_occupy_seats */
/* @param: train_id */
/*        : order_date */
/*        : station_from_id */
/*        : station_to_id */
/*        : seat_type */
/*        : seat_num */
/* @return: succeed */
/*          : left_seat */
begin
    select query_station_order_from_tid_sid__tfi__(train_id, station_from_id) into station_start_order;
    station_order_ptr = station_start_order;
    select query_station_order_from_tid_sid__tfi__(train_id, station_to_id) into station_end_order;
    -- find min_seat --
    select get_min_seat.seat_num
        into min_seat
        from get_min_seats(train_id, order_date, station_from_id, station_to_id, array [seat_type]) get_min_seat
        where in_order = 1;
    -- check satisfiability --
    if min_seat < seat_num then
        succeed := false;
        left_seat := min_seat;
    else
        succeed := true;
        left_seat := 5 - min_seat;
        -- second loop, update station tickets --
        while station_order_ptr != station_end_order
            loop
                update station_tickets
                set stt_num = (select array_set(stt_num, seat_type, stt_num[seat_type] - seat_num))
                    where stt_train_id = train_id
                      and stt_station_id = station_id_ptr
                      and stt_date = order_date;
                station_order_ptr := station_order_ptr + 1;
                select query_station_id_from_tid_so__tfi__(train_id, station_order_ptr) into station_id_ptr;
            end loop;
    end if;
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。

```
/* pre_order_train */
/* @param: train_id */
/*        : station_from_id */
/*        : station_to_id */
/*        : seat_type */
/*        : seat_num */
/*        : order_date */
/* @return: succeed */
/*         : seat_id */
/*         : order_id */
begin
    select succeed, left_seat
        into succeed, seat_id
        from try_occupy_seats(train_id, order_date, station_from_id, station_to_id, seat_type, seat_num);
    if succeed then
        insert into orders (o_train_id, o_date, o_start_station, o_end_station, o_seat_type, o_seat_id,
                                    o_status, o_effect_time)
        select train_id,
               order_date,
               station_from_id,
               station_to_id,
               seat_type,
               seat_id,
               'PRE_ORDERED',
               now();
        select currval(pg_get_serial_sequence('orders', 'o_oid')) into order_id;
    else
        order_id := 0;
    end if;
end;

/* order_train_seats */
/* @param: order_id */
/*        : uid */
/* @return: succeed */
begin
    update orders
    set (o_uid, o_status) = (uid, 'ORDERED')
        where o_oid = order_id;
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。

```
select o_oid as order_id,
    t_train_name as train_name,
    o_train_id as train_id,
    s_start.s_station_name as station_leave,
    o_start_station as station_id,
    s_arrive.s_station_name as station_arrive,
    tfi_start.tfi_leave_time as start_time,
    tfi_end.tfi_arrive_time as arrive_time,
    (select * from get_actual_interval_bt_time(tfi_start.tfi_leave_time, tfi_end.tfi_arrive_time,
        tfi_start.tfi_day_from_departure - tfi_end.tfi_day_from_departure)) as durance,
    tfi_end.tfi_distance - tfi_start.tfi_distance as distance,
    o_seat_type as seat_type,
    o_seat_id as seat_id,
    o_status as status,
    tfi_end.tfi_price[o_seat_type] - tfi_start.tfi_price[o_seat_type] + 5 as price
    from orders
            left join station_list s_start on orders.o_start_station = s_start.s_station_id
            left join station_list s_arrive on orders.o_end_station = s_arrive.s_station_id
            left join train on orders.o_train_id = train.t_train_id
            left join train_full_info tfi_start on s_start.s_station_id = tfi_start.tfi_station_id
        and orders.o_train_id = tfi_start.tfi_train_id
            left join train_full_info tfi_end on s_start.s_station_id = tfi_end.tfi_station_id
        and orders.o_train_id = tfi_end.tfi_train_id
    where o_uid = uid
    and o_date >= start_query_date
    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：

```
/* user_cancel sql */
/* @param: order_id */
/* @return: succeed */
begin
    select o_train_id, o_date, o_start_station, o_end_station, o_seat_type
        into train_id, order_date, start_station, end_station, seat_type
        from orders
        where o_oid = order_id
          and o_status = 'COMPLETE';
    select release_seats(train_id, order_date, start_station, end_station, seat_type, 1);
    update orders
    set o_status = 'CANCELED'
        where o_oid = order_id;
end;

/* remove_outdated_order */
delete
    from orders
    where (select * from get_actual_interval_bt_time(orders.o_effect_time, now(), 0))
              > interval '30 minutes'
        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。

```
select p_pid as uid,
       u_user_name as uname,
       array(
            select o_oid from orders where o_uid = p_pid
       ) as orders
    from passengers
        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

```
-- 2 views to select top 10 train --
create or replace view top_10_train_tickets(train_id, count_num)
as
select o_train_id as train_id, count(*) as count_num
    from orders
    where o_status = 'COMPLETE'
    group by o_train_id
    order by count_num
    limit 10;

create or replace view top_10_train_ids(train_id)
as
select train_id
    from top_10_train_tickets;


/* admin_query_orders */
/* @return: total_order_num */
/*        : total_price */
/*        : hot_trains */
begin
    select count(*),
           sum(tfi_end.tfi_price[o_seat_type] - tfi_start.tfi_price[o_seat_type] + 5)
        into total_order_num, total_price
        from orders
                 left join train_full_info tfi_start on o_start_station = tfi_start.tfi_station_id
            and orders.o_train_id = tfi_start.tfi_train_id
                 left join train_full_info tfi_end on o_end_station = tfi_end.tfi_station_id
            and orders.o_train_id = tfi_end.tfi_train_id
            and orders.o_status = 'COMPLETE';
    hot_trains := array(select t_train_name
                            from train
                                     left join top_10_train_ids on train.t_train_id = top_10_train_ids.train_id);
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。