数据库系统第二次实验设计报告

数据库系统第二次实验设计报告

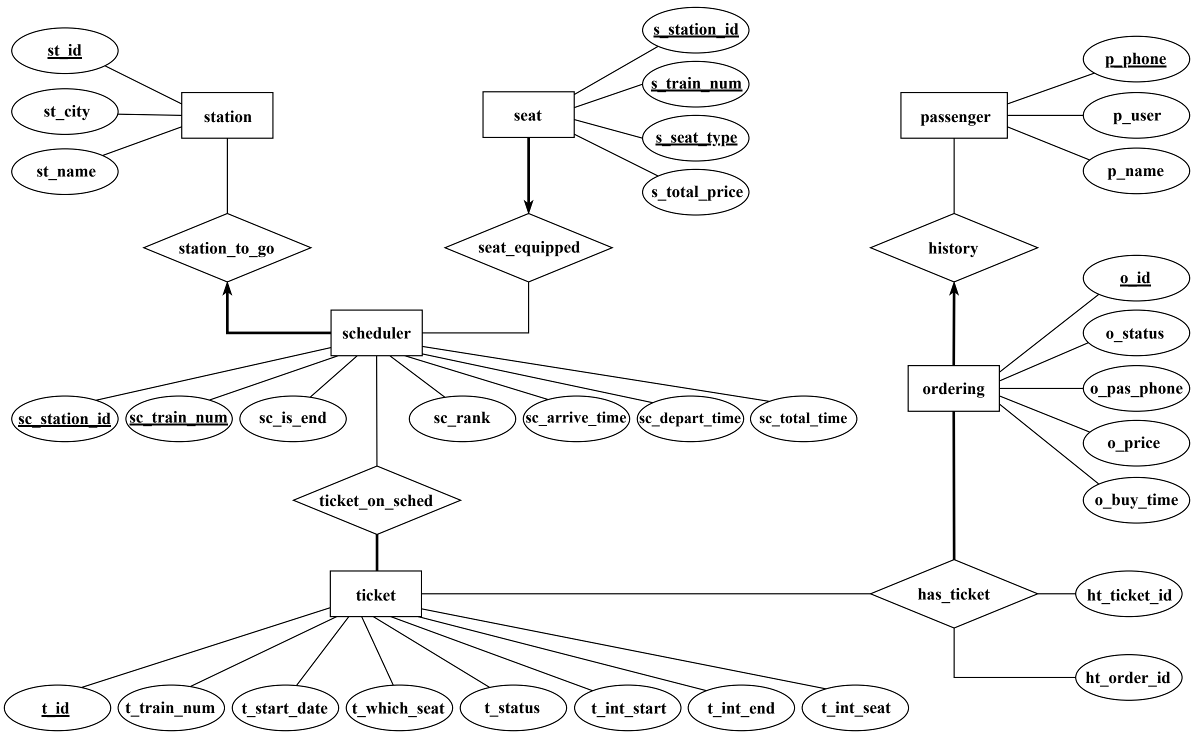
成员

组长：张玉龙；学号：2019K8009929036

组员：代瀚堃；学号：2019K8009929051

分工：张玉龙主要负责实现每个需求的SQL语句和相关操作，特别是查询和更新，以及实验设计中文字内容的主要书写；代瀚堃主要负责ER图的设计，关系模式的基本设计，以及CREATE TABLE语句的书写。两人合作讨论了关系模式的设计思路、范式细化分析、以及需求实现的设计思路。

ER图



关系模式

station

列属性	类型	描述
st_id	integer	车站-城市 编号
st_name	varchar(20)	车站名
st_city	varchar(20)	车站所在城市名

scheduler

列属性	类型	描述
sc_station_id	integer	车站编号，与station表中的st_id相对应
sc_train_num	varchar(5)	车次
sc_arrive_time	time	该车次到达该车站的时间
sc_depart_time	time	该车次离开该车站的时间
sc_total_time	interval	该车次到达该车站时总运行时间
sc_rank	integer	该车站在该车次中是第几站
sc_is_end	enum	该车站在该车次中是否是终点站

注：对于由于是始发站或终点站等原因，缺失到达或离开时间之一的记录，将sc_arrive_time和sc_depart_time中缺失的设置为与另一个时间相等。

seat

列属性	类型	描述
s_station_id	integer	车站编号，与station表中的st_id相对应
s_train_num	varchar(5)	车次
s_seat_type	enum	车座类型
s_total_price	decimal	从该车次的始发站到该站，乘坐该类型座位的总花费

注：如果票价缺失，则s_total_price为0.0

ticket

列属性	类型	描述
t_id	integer	车票编号
t_train_num	varchar(5)	车次
t_start_date	date	车票对应车次始发站的发车日期
t_which_seat	enum	该车票对应的车座编号(一种座位类型下有A ~ E五个编号的座位)
t_status	enum	该车票的状态
t_int_start	integer	该车票的起始站id
t_int_end	integer	该车票的终点站id
t_int_seat	enum	该车票对应的车座类型

注：与其说ticket中的一条记录描述了一张车票，不如说其中的一条记录描述了一段乘车区间的可售状态。

has_ticket

列属性	类型	描述
ht_ticket_id	integer	车票编号，对应ticket中的t_id
ht_order_id	integer	订单编号，对应ordering中的o_id

ordering

列属性	类型	描述
o_id	integer	订单编号
o_pas_phone	char(11)	订单对应用户的手机号，对应passenger中的p_phone
o_price	decimal	订单总价
o_buy_time	timestamp	下单时间
o_status	enum	订单状态

passenger

列属性	类型	描述
p_phone	char(11)	用户手机号
p_user	varchar(20)	用户名
p_name	varchar(20)	用户真实姓名

范式细化分析

station

该表只有一个主键 (st_id)。

由于该表唯一的主键里键属性只有一个，因此对于任何函数确定的关系$X \rightarrow A$，其中X集合如果包含键属性，则X一定是超键。因此不存在部分依赖、非键传递依赖、以及对于键属性的函数依赖。

综上，station的模式是**BCNF**的。

scheduler

该表包含一个主键 (sc_station_id, sc_train_num)，和一个候选键 (sc_train_num, sc_rank)。

对于部分依赖的情形：如果存在函数确定的关系$X \rightarrow A$，X只可能是 sc_station_id, sc_train_num, sc_rank中的一个属性的集合，而这三个属性之一都不能函数确定其他的非键属性，因此不存在部分依赖。

对于非键传递依赖：X集合如果包含 sc_station_id 或 sc_rank 或二者兼有，那么X集合仅有车站的信息，在缺少车次的信息的情况下，即使结合其他非键属性，也不能函数确定另外的非键属性。而如果X集合包含 sc_train_num，那么X集合仅包含车次的信息，其他非键属性不能完全包含车站的相关信息(比如sc_is_end不包含中间站和始发站的信息)，这样X也不能函数确定其他非键属性，因此不存在非键传递依赖。

对于键属性的函数依赖：X集合只可能包含一个键属性，而无论是车次还是车站、车站序号的信息，都不能函数确定其他键属性，因此不存在对于键属性的函数依赖。

综上，scheduler的模式是**BCNF**的。

另：最初的设计中，scheduler的模式还包含sc_is_start以及sc_stay_time两个属性，但是这两个属性中，sc_is_start可以由sc_rank函数确定，sc_stay_time可以由sc_arrive_time和sc_depart_time函数确定，会造成冗余，因此在scheduler的模式设计中最后删除了这两个属性。

seat

该表包含一个主键 (s_station_id, s_train_num, s_seat_type)。

对于部分依赖的情形：如果存在函数确定的关系$X \to A$，由于只存在一个非键属性s_total_price，因此A只能是s_total_price。那么任何一个键属性的集合X，如果不能包含主键中的全部属性，都不能函数确定s_total_price，因为这个价格跟车次、车站、车座类型都是有关系的。因此不存在部分依赖。

对于非键传递依赖：属性A只能是s_total_price，而集合X没有其他的非键属性可以包含，因此不存在非键传递依赖。

对于键属性的函数依赖：属性A可以是上述三个键属性之一。而集合X仅有三个键属性的一部分，则不能函数确定另外的键属性，因为三个键属性互相是独立的。即使集合X包含了非键属性s_total_price，由于座位价格的确定存在随机性，和三个键属性之间不存在确定的函数关系，所以X也不能函数确定另外的键属性A。因此不存在对于键属性的函数依赖。

综上，seat的模式是**BCNF**的。

ticket

该表包含一个主键 (t_id)，除了t_id的其余属性构成的元组是另一个候选键。

对于部分依赖和非键传递依赖：由于不存在非键属性，因此不存在部分依赖和非键传递依赖。

对于键属性的函数依赖：由于t_id和其余的属性是独立的，因此其余的属性不能函数确定t_id。属性A只能是除了t_id的其余键属性之一。而除了t_id的其余属性在ticket中是彼此独立的，所以也不能互相函数确定，即不存在键属性的集合X函数确定另外的键属性A。因此不存在对于键属性的函数依赖。

综上，ticket的模式是**BCNF**的。

has_ticket

该表包含一个主键 (ht_ticket_id, ht_order_id)。

对于部分依赖和非键传递依赖：由于不存在非键属性，因此不存在部分依赖和非键传递依赖。

对于键属性的函数依赖：由于一个ht_order_id可能对应一个或两个ht_ticket_id，因此ht_order_id不能函数确定ht_ticket_id，而一个ht_ticket_id也可能对应多个ht_order_id，即出现同一张票被退订之后又预定的情况，就会出现多个订单号与同一个车票的编号对应。因此这两个键属性互相不能函数确定彼此。因此不存在对于键属性的函数依赖。

综上，has_ticket的模式是**BCNF**的。

ordering

该表包含一个主键 (o_id)，如果不考虑并发 (同时下两个订单)，那么还包含一个候选键 (o_buy_time)。

由于任意候选键只包含一个键属性，因此若存在函数确定的关系$X \to A$，那么集合X一定是超键，因此不存在部分依赖、非键传递依赖和对于键属性的函数依赖。

综上，ordering的模式是**BCNF**的。

passenger

该表包含一个主键(p_phone)和一个候选键(p_user)。

由于任意候选键只包含一个键属性，因此若存在函数确定的关系$X \to A$，那么集合X一定是超键，因此不存在部分依赖、非键传递依赖和对于键属性的函数依赖。

综上，passenger的模式是**BCNF**的。

每个需求对应的SQL语句或操作

需求1

建表

主要是 scheduler、seat、station 表的建立

```
-- 存储列车行程的相关信息，包括车次、经停站、以及各站点的抵离时间、运行时间
-- 站点次序、终点站的标识等
CREATE TABLE scheduler (
    sc_station_id INTEGER NOT NULL,
    sc_train_num VARCHAR(5) NOT NULL,
    sc_arrive_time TIME NOT NULL,
    sc_depart_time TIME NOT NULL,
    sc_total_time INTERVAL NOT NULL,
    sc_rank INTEGER NOT NULL,
    sc_is_end ENUM ('True', 'False') NOT NULL,
    PRIMARY KEY (sc_station_id, sc_train_num),
    FOREIGN KEY (sc_station_id) REFERENCES station(st_id),
    CHECK (sc_rank >= 1)
);

-- 存储座位的相关信息
-- 包括座位类型、票价、对应车次车站等
CREATE TABLE seat (
    s_station_id INTEGER NOT NULL,
    s_train_num VARCHAR(5) NOT NULL,
    s_seat_type ENUM ('H', 'S', 'HU', 'HM', 'HL', 'SU', 'SL') NOT NULL,
    s_total_price DECIMAL(7, 2) NOT NULL,
    PRIMARY KEY (s_station_id, s_train_num, s_seat_type),
    FOREIGN KEY (s_station_id, s_train_num) REFERENCES
    scheduler(sc_station_id, sc_train_num),
    FOREIGN KEY (s_station_id) REFERENCES station(st_id),
    CHECK (s_total_price >= 0.00)
);

-- 存储车站-城市对应关系
CREATE TABLE station (
    st_id  INTEGER PRIMARY KEY,
    st_name VARCHAR(20) NOT NULL,
    st_city VARCHAR(20) NOT NULL
);
```

需求2

建表

主要是 ticket 表的建立

```
-- 存储车票(车票和座位绑定)
-- 包括车票编号、车次、发车日期、座位编号、车票状态、抵离站点、座位类型等信息
CREATE TABLE ticket (
    t_id INTEGER PRIMARY KEY,
    t_train_num VARCHAR(5) NOT NULL,
    t_start_date DATE NOT NULL,
    t_which_seat ENUM ('A', 'B', 'C', 'D', 'E') NOT NULL,
```

```sql
    t_status ENUM ('Available', 'Sold', 'Unavailable') NOT NULL,
    t_int_start INTEGER NOT NULL,
    t_int_end INTEGER NOT NULL,
    t_int_seat ENUM ('H', 'S', 'HU', 'HM', 'HL', 'SU', 'SL') NOT NULL,
    FOREIGN KEY (t_int_start, t_train_num) REFERENCES scheduler(sc_station_id,
sc_train_num),
    FOREIGN KEY (t_int_end, t_train_num) REFERENCES scheduler(sc_station_id,
sc_train_num),
    FOREIGN KEY (t_int_start) REFERENCES station(st_id),
    FOREIGN KEY (t_int_end) REFERENCES station(st_id)
);
```

查询、插入 ticket 初始数据

```sql
-- 车次、对应车次起始站的station_id
CREATE VIEW start_station_id AS
SELECT
    sc.sc_train_num AS train_num,
    sc.sc_station_id AS station_id
FROM
    scheduler AS sc
WHERE
    sc.sc_rank=1;   -- rank为1表示起始站

-- 车次，对应车次终点站的station_id
CREATE VIEW end_station_id AS
SELECT
    sc.sc_train_num AS train_num,
    sc.sc_station_id AS station_id
FROM
    scheduler AS sc
WHERE
    sc.sc_is_end="True";    -- is_end为True表示终点站

-- 插入所有车次从起始站到终点站的车票
-- 下列插入应做 10(-5天 <-> +5天) * 7(7种座位类型) * 5(每天每种座位5个) 次
-- 分别遍历 t_start_date : 从当前时间算起，过去五天到未来五天
-- t_int_seat : H,S,HU,HM,HL,SU,SL 7种座位类型
-- t_which_seat : A,B,C,D,E 五个座位编号
-- t_id可以最后统一更新为行号，保证是唯一的(主键)
INSERT INTO ticket(t_train_num, t_int_start, t_int_end, t_status)
SELECT
    stid.train_num,
    stid.station_id,
    edid.station_id,
    "Avaliable"
FROM
    start_station_id AS stid,
    end_station_id AS edid
WHERE
    stid.train_num=edid.train_num;
```

需求3

建表

主要是passenger、ordering、has_ticket表的建立

```
-- 存储用户注册信息，包括用户名、手机、真实姓名
CREATE TABLE passenger (
    p_phone CHAR(11) PRIMARY KEY,
    p_user VARCHAR(20) UNIQUE,
    p_name VARCHAR(20) NOT NULL
);

-- 存储订单的基本信息，包括订单号、所属手机号(索引到用户)、订单总价、下单时间、订单状态
CREATE TABLE ordering (
    o_id INTEGER PRIMARY KEY,
    o_pas_phone CHAR(11) NOT NULL,
    o_price DECIMAL(7, 2) NOT NULL,
    o_buy_time TIMESTAMP NOT NULL,
    o_status ENUM ('NORMAL', 'CANCELED') NOT NULL,
    FOREIGN KEY (o_pas_phone) REFERENCES passenger(p_phone),
    CHECK (o_price >= 0.00)
);

-- 存储订单和车票的对应关系，从而确定一笔订单对应的行程信息
-- 包括订单号、车票编号
CREATE TABLE has_ticket (
    ht_ticket_id INTEGER NOT NULL,
    ht_order_id INTEGER NOT NULL,
    PRIMARY KEY (ht_ticket_id, ht_order_id),
    FOREIGN KEY ht_ticket_id REFERENCES ticket(t_id),
    FOREIGN KEY ht_order_id REFERENCES ordering(o_id)
);
```

乘客注册时插入记录

```
-- 输入：手机号，姓名，用户名
INPUT: phone, name, user

-- 用户名、手机号登录
INSERT INTO passenger(p_phone, p_name, p_user)
VALUES(INPUT.phone, INPUT.name, INPUT.user);
```

乘客登录时查询手机号是否匹配

```
-- 输入：用户名，手机号
INPUT: user, phone

-- 查询记录为0则不匹配，为1则匹配
SELECT count(*)
FROM passenger
WHERE p_user=INPUT.user AND p_phone=INPUT.phone;
```

需求4

查询

```
-- 输入：车次、出发日期
INPUT: train_num, ddate

-- 查询该车次的所有车站、相应车站在车次中是第几站
CREATE VIEW train_stations AS
SELECT
    st.st_station AS station,
    sc.sc_rank AS rank
FROM
    scheduler AS sc,
    station AS st
WHERE
    sc.sc_train_num=INPUT.train_num AND
    sc.sc_station_id=st.st_id;

-- 车票是给定车次的，日期是给定日期的
-- 车票从起始站出发，到达该车次的所有站点
-- 车票的状态是可售
-- 按照站点顺序、座位类型依次排列
-- 余票数量可以对station, seat_type分组后求count(*)
SELECT
    ts.rank AS rank,
    ts.station AS station,
    sc3.sc_depart_time AS depart_time,
    sc3.sc_arrive_time AS arrive_time,
    s.s_seat_type AS seat_type,
    s.s_total_price AS price,
    t.t_id AS ticket_id,
    t.t_which_seat AS which_seat
FROM
    train_stations AS ts,
    scheduler AS sc1,
    scheduler AS sc2,
    scheduler AS sc3,
    ticket AS t,
    seat AS s
WHERE
    sc1.sc_station_id=t.t_int_start AND sc1.sc_train_num=t.t_train_num AND
    sc2.sc_station_id=t.t_int_end AND sc2.sc_train_num=t.t_train_num AND
    sc1.sc_rank=1 AND sc2.sc_rank>=ts.rank AND
    t.t_status="Avaliable" AND t.t_start_date=INPUT.ddate AND
    t.t_train_num=INPUT.train_num AND
-- 筛选合适的票
    sc3.sc_train_num=INPUT.train_num AND sc3.sc_rank=ts.rank AND
    sc3.t_train_num=s.s_train_num AND sc3.sc_station_id=s.s_station_id AND
    t.t_int_seat=s.s_seat_type
-- 确定票价
ORDER BY
    ts.rank ASC,
    s.s_seat_type DESC;
```

需求5

多次查询

```
-- 输入：出发城市、到达城市、出发日期、出发时间
INPUT: dcity, acity, ddate, dtime

-- 票id、起始站序号、到达站序号
-- 每张票是从该趟车的第几站到第几站的
CREATE VIEW ticket_rank AS
SELECT
    t.t_id AS ticket_id,
    sc1.sc_rank AS start_rank,
    sc2.sc_rank AS end_rank
FROM
    ticket AS t,
    scheduler AS sc1,
    scheduler AS sc2
WHERE
    t.t_int_start=sc1.sc_station_id AND t.t_int_end=sc2.sc_station_id AND
    t.t_train_num=sc1.sc_train_num AND t.t_train_num=sc2.sc_train_num;

-- 经过出发城市的车次、车站、车站ID(中间数据)
CREATE VIEW pass_start AS
SELECT
    sc.sc_train_num AS train_num,
    st.st_name AS station_name,
    st.st_id AS station_id
FROM
    station AS st,
    scheduler AS sc
WHERE
    st.st_id=sc.sc_station_id AND st.st_city=INPUT.dcity;

-- 经过到达城市的车次、车站、车站ID(中间数据)
CREATE VIEW pass_end AS
SELECT
    sc.sc_train_num AS train_num,
    st.st_name AS station_name,
    st.st_id AS station_id
FROM
    station AS st,
    scheduler AS sc
WHERE
    st.st_id=sc.sc_station_id AND st.st_city=INPUT.acity;

-- 上述两组车次中，经过相同城市的车次的组合
-- 第一趟车次，第二趟车次，总起始站ID，总终点站ID，第一趟车终点站ID，第二趟车起始站ID
-- 总起始站，总终点站，第一趟车终点站，第二趟车起始站，中间站所在城市(交互信息输出)
-- train1=train2, mstation1=mstation2，直达列车
-- train1!=train2, mstation1=mstation2，同站换乘
-- train1!=train2, mstation1!=mstation2，同城换乘但不同站
-- train1=train2, mstation1!=mstation2，一趟列车同城经过两个车站，也是直达，包含在情况
-- 1中。
CREATE VIEW train_same_city AS
SELECT
    sc1.sc_train_num AS train1,
```

```sql
    sc2.sc_train_num AS train2,
    pa1.station_id AS dstation_id,
    pa2.station_id AS astation_id,
    sc1.sc_station_id AS mstation_id1,
    sc2.sc_station_id AS mstation_id2,
    pa1.station_name AS dstation,
    pa2.station_name AS astation,
    st1.st_name AS mstation1,
    st2.st_name AS mstation2,
    st1.st_city AS mcity
FROM
    pass_start AS pa1,
    pass_end AS pa2,
    scheduler AS sc1,
    scheduler AS sc2,
    station AS st1,
    station AS st2
WHERE
    pa1.train_num=sc1.sc_train_num AND pa2.train_num=sc2.sc_train_num AND
    sc1.sc_station_id=st1.st_id AND sc2.sc_station_id=st2.st_id AND
    st1.st_city=st2.st_city;

-- 直达的车次、起始站、终点站
-- 起始站id、终点站id
-- 直达车次记录在train_same_city视图中不是唯一的，所以要Group By一下
CREATE VIEW direct_train AS
SELECT
    ts.train1 AS train,
    ts.dstation AS dstation,
    ts.astation AS astation,
    ts.dstation_id AS dstation_id,
    ts.astation_id AS astation_id
FROM
    train_same_city AS ts
WHERE
    ts.train1=ts.train2 AND ts.mstation1=ts.mstation2
GROUP BY
    ts.train1, ts.dstation, ts.astation;

-- 可行的直达车次
-- 车次、出发站、达到站、对应的id
-- 车座类型、票价
-- 出发时间、行程总时长
CREATE VIEW good_direct_train AS
SELECT
    dt.train AS train,
    dt.dstation AS dstation,
    dt.astation AS astation,
    dt.dstation_id AS dstation_id,
    dt.astation_id AS astation_id,
    s1.s_seat_type AS seat_type,
    s2.s_total_price-s1.s_total_price AS price,
    sc1.sc_depart_time AS depart_time,
    (CASE
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time>=0
        THEN sc2.sc_total_time-sc1.sc_total_time
            -(sc1.sc_arrive_time-sc1.sc_depart_time)
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time<0
        THEN sc2.sc_total_time-sc1.sc_total_time
             -(sc1.sc_arrive_time-sc1.sc_depart_time+INTERVAL '24:00:00')
     END
    ) AS total_time
FROM
    direct_train AS dt,
    scheduler AS sc1,
    scheduler AS sc2,
    seat AS s1,
    seat AS s2
WHERE
    dt.train=sc1.sc_train_num AND dt.dstation_id=sc1.sc_station_id AND
    dt.train=sc2.sc_train_num AND dt.astation_id=sc2.sc_station_id AND
    sc1.sc_depart_time>=INPUT.dtime AND
-- 发车时间晚于给定的出发时间
    sc1.sc_station_id=s1.s_station_id AND s1.s_train_num=sc1.sc_train_num AND
    (s1.s_total_price>0.0 OR sc1.sc_rank=1) AND
    sc2.sc_station_id=s2.s_station_id AND s2.s_train_num=sc2.sc_train_num AND
    (s2.s_total_price>0.0 OR sc2.sc_rank=1) AND
    s1.s_seat_type=s2.s_seat_type;
-- 起始站和到达站都必须能售票(有票价)，除非是列车始发站。

-- 订票时SQL查询ticket的日期，是所乘车次起始站(Rank=1)的发车日期
-- 而输入的出发日期是第一趟车的上车点的日期，因此需要进行换算
CREATE VIEW pre_ddate AS
SELECT
    sc.sc_train_num AS train_num,
    (CASE
        WHEN sc.sc_depart_time-sc.sc_arrive_time>=0
        THEN date(INPUT.ddate+sc.sc_arrive_time-sc.sc_total_time)
        WHEN sc.sc_depart_time-sc.sc_arrive_time<0
        THEN date(INPUT.ddate-1day+sc.sc_arrive_time-sc.sc_total_time)
     END
    ) AS ddate
FROM
    scheduler AS sc,
    good_direct_train AS gd
WHERE
    sc.sc_train_num=gd.train AND sc.sc_station_id=gd.dstation_id;

-- 直达：车次、起始终点站、发车时间、行程总时间、票价都已查询完毕
-- 只剩下关于车票的信息，即余票
CREATE VIEW direct_ticket AS
SELECT
    t.t_id AS ticket_id,
    gd.train AS train,
    gd.dstation AS dstation,
    gd.astation AS astation,
    gd.seat_type AS seat_type,
    gd.depart_time AS depart_time,
    gd.total_time AS total_time,
    t.t_which_seat AS which_seat,
    gd.price AS price
FROM
    good_direct_train AS gd,
    scheduler AS sc1,
    scheduler AS sc2,
    ticket_rank AS tr,
    ticket AS t,
    seat AS s1,
    seat AS s2,
    pre_ddate AS pd
WHERE
    sc1.sc_station_id=gd.dstation_id AND sc1.sc_train_num=gd.train AND
    sc2.sc_station_id=gd.astation_id AND sc2.sc_train_num=gd.train AND
    t.t_train_num=gd.train AND
    tr.ticket_id=t.t_id AND
    sc1.rank>=tr.start_rank AND sc2.rank<=tr.end_rank AND
    t.t_start_date=pd.ddate AND t.t_train_num=pd.train_num AND
    t.t_status="Avaliable" AND t.t_int_seat=gd.seat_type;

-- 余票数量
SELECT
    dt.train AS train,
    dt.seat_type AS seat_type,
    dt.price AS price
    count(*) AS number
FROM
    direct_ticket AS dt
GROUP BY dt.train, dt.dstation, dt.astation, dt.seat_type;
-- 对于直达列车，只需要综合上述关于直达的表的信息即可


-- 同站换乘的车次、车站等信息
CREATE VIEW transfer_train_same_station AS
SELECT *
FROM train_same_city
WHERE mstation1=mstation2 AND train1<>train2;

-- 可行的同站换乘车次
-- 车次1、车次2、总起始站、总终点站、中间站、以及它们各自的id
-- 第一趟车的车座类型、票价；第二趟车的车座类型、票价
-- 第一趟车发车时间，第一趟车行程总时长；第二趟车发车时间，第二趟车行程总时长；换乘时长
CREATE VIEW good_transfer_train_same_station AS
SELECT
    ts.train1 AS train1,
    ts.train2 AS train2,
    ts.dstation AS dstation,
    ts.astation AS astation,
    ts.mstation1 AS mstation,
    ts.dstation_id AS dstation_id,
    ts.astation_id AS astation_id,
    ts.mstation_id1 AS mstation_id,
    sc1.sc_depart_time AS depart_time1,
    s2.s_seat_type AS seat_type1,
    s2.s_price-s1.s_price AS price1,
    s3.s_seat_type AS seat_type2,
    s4.s_price-s3.s_price AS price2,
    (CASE
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time>=0
        THEN sc2.sc_total_time-sc1.sc_total_time
             -(sc1.sc_arrive_time-sc1.sc_depart_time)
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time<0
        THEN sc2.sc_total_time-sc1.sc_total_time
             -(sc1.sc_arrive_time-sc1.sc_depart_time+24:00:00)
     END
    ) AS total_time1,
    sc3.sc_depart_time AS depart_time2,
    (CASE
        WHEN sc3.sc_arrive_time-sc3.sc_depart_time>=0
        THEN sc4.sc_total_time-sc3.sc_total_time
             -(sc3.sc_arrive_time-sc3.sc_depart_time)
        WHEN sc3.sc_arrive_time-sc3.sc_depart_time<0
        THEN sc4.sc_total_time-sc3.sc_total_time
             -(sc3.sc_arrive_time-sc3.sc_depart_time+24:00:00)
     END
    ) AS total_time2,
    (CASE
        WHEN sc3.sc_depart_time-Sc2.sc_arrive_time>=1:00:00 AND
             sc3.sc_depart_time-sc2.sc_arrive_time<=4:00:00
        THEN sc3.sc_depart_time-sc2.sc_arrive_time
        WHEN sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00>=1:00:00 AND
             sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00<=4:00:00
        THEN sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00
     END
    ) AS transfer_time
FROM
    transfer_train_same_station AS ts,
    scheduler AS sc1,   -- 第一趟车上车站
    scheduler AS sc2,   -- 第一趟车下车站
    scheduler AS sc3,   -- 第二趟车上车站
    scheduler AS sc4,   -- 第二趟车下车站
    seat AS s1,
    seat AS s2,
    seat AS s3,
    seat AS s4
WHERE
    ts.train1=sc1.sc_train_num AND ts.dstation_id=sc1.sc_station_id AND
    ts.train1=sc2.sc_train_num AND ts.mstation_id1=sc2.sc_station_id AND
    ts.train2=sc3.sc_train_num AND ts.mstation_id2=sc3.sc_station_id AND
    ts.train2=sc4.sc_train_num AND ts.astation_id=sc4.sc_station_id AND
    ((sc3.sc_depart_time-sc2.sc_arrive_time>=1:00:00) AND
     (sc3.sc_depart_time-sc2.sc_arrive_time<=4:00:00) OR
     (sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00>=1:00:00) AND
     (sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00<=4:00:00)) AND
    sc1.sc_depart_time>=INPUT.dtime AND
    sc1.sc_train_num=s1.s_train_num AND sc1.sc_station_id=s1.s_station_id AND
    sc2.sc_train_num=s2.s_train_num AND sc2.sc_station_id=s2.s_station_id AND
    sc3.sc_train_num=s3.s_train_num AND sc3.sc_station_id=s3.s_station_id AND
    sc4.sc_train_num=s4.s_train_num AND sc4.sc_station_id=s4.s_station_id AND
    (s1.s_price>0.0 OR sc1.sc_rank=1) AND
    (s2.s_price>0.0 OR sc2.sc_rank=1) AND
    (s3.s_price>0.0 OR sc3.sc_rank=1) AND
    (s4.s_price>0.0 OR sc4.sc_rank=1) AND
    s2.s_seat_type=s1.s_seat_type AND s4.s_seat_type=s3.s_seat_type;

-- 预处理同站换乘的日期信息
CREATE VIEW pre_ddate_transfer_same_station AS
SELECT
    (CASE
        WHEN sc1.sc_depart_time-sc1.sc_arrive_time>=0
        THEN date(INPUT.ddate+sc1.sc_arrive_time-sc1.sc_total_time)
        WHEN sc1.sc_depart_time-sc1.sc_arrive_time<0
        THEN date(INPUT.ddate-1day+sc1.sc_arrive_time-sc1.sc_total_time)
```

```sql
     END
    ) AS ddate1,
    (CASE
        WHEN sc2.sc_depart_time-sc2.sc_arrive_time>=0
        THEN date(INPUT.ddate+sc1.sc_depart_time+gt.total_time1+
                  gt.transfer_time-(sc2.sc_depart_time-sc2.sc_arrive_time)
                  -sc2.sc_total_time)
        WHEN sc2.sc_depart_time-sc2.sc_arrive_time<0
        THEN date(INPUT.ddate+sc1.sc_depart_time+gt.total_time1+
                  gt.transfer_time-(24:00:00+sc2.sc_depart_time-
                  sc2.sc_arrive_time)-sc2.sc_total_time)
     END
    ) AS ddate2
FROM
    good_transfer_train_same_station AS gt,
    scheduler AS sc1,
    scheduler AS sc2
WHERE
    sc1.sc_train_num=gt.train1 AND sc1.sc_station_id=gt.dstation_id;
    sc2.sc_train_num=gt.train2 AND sc2.sc_station_id=gt.mstation_id;

-- 同站换乘的票价和余票信息可以看作两趟直达列车分别进行筛选
-- 最终结果对两趟列车的余票求最小值，票价求和
-- 行程总时间即total_time1+total_time2+transfer_time

-- 同城不同站换乘的车次、车站等信息
CREATE VIEW trainsfer_train_same_city AS
SELECT * FROM train_same_city
WHERE train1<>train2 AND mstation1<>mstation2;

-- 可行的同城不同站换乘车次
-- 类比同站换乘，只需要修改换乘经停时间的区间
CREATE VIEW good_transfer_train_same_station AS
SELECT
    ts.train1 AS train1,
    ts.train2 AS train2,
    ts.dstation AS dstation,
    ts.astation AS astation,
    ts.mstation1 AS mstation,
    ts.dstation_id AS dstation_id,
    ts.astation_id AS astation_id,
    ts.mstation_id1 AS mstation_id,
    sc1.sc_depart_time AS depart_time1,
    s2.s_seat_type AS seat_type1,
    s2.s_price-s1.s_price AS price1,
    s3.s_seat_type AS seat_type2,
    s4.s_price-s3.s_price AS price2,
    (CASE
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time>=0
        THEN sc2.sc_total_time-sc1.sc_total_time
             -(sc1.sc_arrive_time-sc1.sc_depart_time)
        WHEN sc1.sc_arrive_time-sc1.sc_depart_time<0
        THEN sc2.sc_total_time-sc1.sc_total_time
             -(sc1.sc_arrive_time-sc1.sc_depart_time+24:00:00)
     END
    ) AS total_time1,
    sc3.sc_depart_time AS depart_time2,
    (CASE
        WHEN sc3.sc_arrive_time-sc3.sc_depart_time>=0
        THEN sc4.sc_total_time-sc3.sc_total_time
             -(sc3.sc_arrive_time-sc3.sc_depart_time)
        WHEN sc3.sc_arrive_time-sc3.sc_depart_time<0
        THEN sc4.sc_total_time-sc3.sc_total_time
             -(sc3.sc_arrive_time-sc3.sc_depart_time+24:00:00)
     END
    ) AS total_time2,
    (CASE
        WHEN sc3.sc_depart_time-Sc2.sc_arrive_time>=2:00:00 AND
             sc3.sc_depart_time-sc2.sc_arrive_time<=4:00:00
        THEN sc3.sc_depart_time-sc2.sc_arrive_time
        WHEN sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00>=2:00:00 AND
             sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00<=4:00:00
        THEN sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00
     END
    ) AS transfer_time
FROM
    transfer_train_same_station AS ts,
    scheduler AS sc1,   -- 第一趟车上车站
    scheduler AS sc2,   -- 第一趟车下车站
    scheduler AS sc3,   -- 第二趟车上车站
    scheduler AS sc4,   -- 第二趟车下车站
    seat AS s1,
    seat AS s2,
    seat AS s3,
    seat AS s4
WHERE
    ts.train1=sc1.sc_train_num AND ts.dstation_id=sc1.sc_station_id AND
    ts.train1=sc2.sc_train_num AND ts.mstation_id1=sc2.sc_station_id AND
    ts.train2=sc3.sc_train_num AND ts.mstation_id2=sc3.sc_station_id AND
    ts.train2=sc4.sc_train_num AND ts.astation_id=sc4.sc_station_id AND
    ((sc3.sc_depart_time-sc2.sc_arrive_time>=2:00:00) AND
     (sc3.sc_depart_time-sc2.sc_arrive_time<=4:00:00) OR
     (sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00>=2:00:00) AND
     (sc3.sc_depart_time-sc2.sc_arrive_time+24:00:00<=4:00:00)) AND
    sc1.sc_depart_time>=INPUT.dtime AND
    sc1.sc_train_num=s1.s_train_num AND sc1.sc_station_id=s1.s_station_id AND
    sc2.sc_train_num=s2.s_train_num AND sc2.sc_station_id=s2.s_station_id AND
    sc3.sc_train_num=s3.s_train_num AND sc3.sc_station_id=s3.s_station_id AND
    sc4.sc_train_num=s4.s_train_num AND sc4.sc_station_id=s4.s_station_id AND
    (s1.s_price>0.0 OR sc1.sc_rank=1) AND
    (s2.s_price>0.0 OR sc2.sc_rank=1) AND
    (s3.s_price>0.0 OR sc3.sc_rank=1) AND
    (s4.s_price>0.0 OR sc4.sc_rank=1) AND
    s2.s_seat_type=s1.s_seat_type AND s4.s_seat_type=s3.s_seat_type;

-- 预处理同城不同站换乘的日期信息
-- 参考同站换乘的处理，只需要修改第二趟车的上车点即可
CREATE VIEW pre_ddate_transfer_same_station AS
SELECT
    (CASE
        WHEN sc1.sc_depart_time-sc1.sc_arrive_time>=0
        THEN date(INPUT.ddate+sc1.sc_arrive_time-sc1.sc_total_time)
        WHEN sc1.sc_depart_time-sc1.sc_arrive_time<0
        THEN date(INPUT.ddate-1day+sc1.sc_arrive_time-sc1.sc_total_time)
     END
    ) AS ddate1,
    (CASE
        WHEN sc2.sc_depart_time-sc2.sc_arrive_time>=0
        THEN date(INPUT.ddate+sc1.sc_depart_time+gt.total_time1+
                  gt.transfer_time-(sc2.sc_depart_time-sc2.sc_arrive_time)
                  -sc2.sc_total_time)
        WHEN sc2.sc_depart_time-sc2.sc_arrive_time<0
        THEN date(INPUT.ddate+sc1.sc_depart_time+gt.total_time1+
                  gt.transfer_time-(24:00:00+sc2.sc_depart_time-
                  sc2.sc_arrive_time)-sc2.sc_total_time)
     END
    ) AS ddate2
FROM
    good_transfer_train_same_station AS gt,
    scheduler AS sc1,
    scheduler AS sc2
WHERE
    sc1.sc_train_num=gt.train1 AND sc1.sc_station_id=gt.dstation_id;
    sc2.sc_train_num=gt.train2 AND sc2.sc_station_id=gt.mstation_id2;

-- 同城不同站的换乘，票价和余票信息也可以按照两趟直达列车分别进行筛选
-- 最后对票价求和，对余票求最小值
```

需求6

查询

交换需求5中的出发地和到达地城市输入

需求7

插入、查询、更新

```
-- 需求5查询到总时长，出发日期和时间给定，求出到达日期和到达时间
INPUT: train_num, ddate, dtime, dstation, astation, adate(calculated),
atime(calculated), seattype, price, phone(known), ustatus(known)

-- 订票费直接判断，总票价直接求和

-- 在订单表中插入这笔订单的相关信息(都已给出)
INSERT INTO ordering(o_id, o_pas_phone, o_price, o_buy_time, o_status)
VALUES(rownum, INPUT.phone, CALULATED.total_price, now(), "NORMAL");

-- 在需求5的Direct_ticket视图中已经查询到可以预定的座位以及Ticket编号
-- 由用户选定Ticket编号来预定哪一张票
-- 根据情况选择是否插入新的票
CREATE TRIGGER insert_ticket
AFTER UPDATE ON ticket
referencing
    old row AS oldtuple
    new row AS newtuple
FOR EACH ROW
WHEN (oldtuple.t_int_start<newtuple.t_int_start)
    INSERT INTO ticket(t_id, t_train_num, t_which_seat,
                       t_status, t_int_start, t_int_end, t_int_seat)
    VALUES(rownum, oldtuple.t_train_num, oldtuple.t_which_seat, "Avaliable",
           oldtuple.t_int_start, newtuple.t_int_start, oldtuple.t_int_seat)
WHEN (oldtuple.t_int_end>newtuple.t_int_end)
    INSERT INTO ticket(t_id, t_train_num, t_which_seat,
                        t_status, t_int_start, t_int_end, t_int_seat)
    VALUES(rownum, oldtuple.t_train_num, oldtuple.t_which_seat, "Avaliable",
            newtuple.t_int_end, oldtuple.t_int_end, oldtuple.t_int_seat);

-- 更改选定的这张票(或几张票，其中Gt.dstation_id, Gt.astation_id从视图中筛选)
UPDATE ticket
SET t_status="Sold", t_int_start=Gt.dstation_id, t_int_end=Gt.astation_id
WHERE t_id=SELECTED.id;

-- 更新has_ticket表
-- 订单号插入订单的时候会知道，车票编号下订单的时候会选定
INSERT INTO has_ticket(ht_order_id, ht_ticket_id)
VALUES(SELECTED.oid, SELECTED.tid);
```

需求8

查询给定日期范围的订单信息

```
INPUT: date1, date2

-- 订单从has_ticket连接到ticket表中
-- ticket可以从scheduler连接到station从而输出站点名字
SELECT
    o.o_id AS order_id,
    t.t_start_date AS depart_date,
    st1.name AS depart_station,
    st2.name AS arrive_station,
    o.price AS price,
    o.status AS order_status
FROM
    ordering AS o,
    has_ticket AS ht,
    ticket AS t,
    scheduler AS sc1,
    scheduler AS sc2,
    station AS st1,
    station AS st2
WHERE
    o.o_id=ht.order_id AND ht.ticket_id=t.id AND
    t.t_start_date>=date1 AND t.t_start_date<=date2 AND
    t.t_int_start=sc1.sc_station_id AND t.t_train_num=sc1.sc_train_num AND
    sc1.sc_station_id=st1.st_id AND
    t.t_int_start=sc2.sc_station_id AND t.t_train_num=sc2.sc_train_num AND
    sc2.sc_station_id=st2.st_id;

-- 按照车次查询信息和需求4的操作类似
```

更新

```
INPUT: id

-- 更新订单状态
UPDATE ordering
SET o_status="canceled"
WHERE id=INPUT.id;

-- 取消订单时，需要合并车票
```

```
CREATE VIEW need_aggregate AS
SELECT
    t1.t_id AS ticket_id0,
    t2.t_id AS ticket_id1,
    t3.t_id AS ticket_id2,
    t2.t_int_start AS new_int_start,
    t3.t_int_end AS new_int_end
FROM
    ticket AS t1,
    ticket AS t2,
    ticket AS t3,
    has_ticket AS ht
WHERE
    ht.ht_order_id=INPUT.id AND ht.ht_ticket_id=t1.t_id AND
    t2.t_int_end=t1.t_int_start AND t2.t_status="Avaliable" AND
    t3.t_int_start=t1.t_int_end AND t3.t_status="Avaliable";

-- 废弃车票
UPDATE ticket
SET t_status="Unavaliable"
WHERE t_id=(SELECT ticket_id1 FROM need_aggregate;);

UPDATE ticket
SET t_status="Unavaliable"
WHERE t_id=(SELECT ticket_id2 FROM need_aggregate;);

-- 合并车票
UPDATE ticket
SET t_int_start=(SELECT new_int_start FROM need_aggregate;),
    t_int_end=(SELECT new_int_end FROM need_aggregate;)
WHERE t_id=(SELECT ticket_id0 FROM need_aggregate;);

-- 实现的时候可以考虑实现一个trigger
```

需求9

查询

```
-- 总订单数
SELECT count(*) FROM ordering WHERE o_status="Normal";

-- 总票价
SELECT sum(o_price) FROM ordering WHERE o_status="Normal";

-- 热点车次排序
SELECT o_train_num, count(*) AS number
FROM ordering
WHERE o_status="Noraml"
GROUP BY o_train_num
ORDER BY number DESC
LIMIT 10;

-- 注册用户列表
SELECT * FROM passenger;

-- 查看每个用户订单
SELECT *
```

```sql
FROM ordering
ORDER BY o_pas_phone ASC;
-- 或者查看特定用户的订单
SELECT *
FROM ordering
WHERE o_pas_phone=INPUT.phone;
```