数据库系统原理作业#3

1.答：12306网站余票查询业务模型的数据模式如下所示：

用户的12306账户（账号ID、登录密码）；

用户（用户账号ID、姓名、性别、身份证号、籍贯、联系电话、邮箱）；

订票（订单号、订购者ID、订票时间）；

车票（订单号、车次、出发时间、到站时间、始发站、终点站、座位类型、票价、余票数目）；

列车（车次、行车路线、行驶里程、行驶时间）

将上述数据模式转成表，表结构描述如下：

<1>账户表（Account表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | | Account | | | |
| 数据库用户 | | root | | | |
| 主键 | | ID | | | |
| 外键 | | ID | | | |
| 排序字段 | | ID | | | |
| 索引字段 | | ID | | | |
| 字段名称 | 数据类型 | 允许为空 | 唯一 | 默认值 | 约束条件 |
| ID | int | N | Y |  | 主键、外键 |
| password | varchar | Y | Y |  |  |

<2>用户表（User表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | | User | | | |
| 数据库用户 | | root | | | |
| 主键 | | ID | | | |
| 外键 | |  | | | |
| 排序字段 | | ID | | | |
| 索引字段 | | ID | | | |
| 字段名称 | 数据类型 | 允许为空 | 唯一 | 默认值 | 约束条件 |
| ID | int | N | Y |  | 主键 |
| Name | varchar | N | N |  |  |
| Gender | char | Y | N |  | 男/女 |
| IDCode | varchar | N | N |  |  |
| Native\_place | varchar | Y | N |  |  |
| telephone | bigint | Y | N |  | 必须满足是11位数字长度 |
| mail | varchar | Y | N |  |  |

<3>订票表（Booking表）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | | Booking | | | |
| 数据库用户 | | root | | | |
| 主键 | | NO | | | |
| 外键 | | ID | | | |
| 排序字段 | | NO | | | |
| 索引字段 | | NO | | | |
| 字段名称 | 数据类型 | 允许为空 | 唯一 | 默认值 | 约束条件 |
| NO | varchar | N | Y |  | 主键 |
| ID | int | N | N |  | 外键 |
| bookingtime | timestamp | Y | N |  |  |

<4>车票信息表（Information）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | | information | | | |
| 数据库用户 | | root | | | |
| 主键 | | NO | | | |
| 外键 | | NO | | | |
| 排序字段 | | NO | | | |
| 索引字段 | | NO | | | |
| 字段名称 | 数据类型 | 允许为空 | 唯一 | 默认值 | 约束条件 |
| NO | varchar | N | Y |  | 主键、外键 |
| Trips | varchar | Y | N |  | 外键 |
| Departure\_time | timestamp | Y | N |  |  |
| Arrival\_time | timestamp | Y | N |  |  |
| Originating\_station | varchar | Y | N |  |  |
| Arrival\_station | varchar | Y | N |  |  |
| type | varchar | Y | N |  |  |
| price | int | Y | N |  |  |
| amount | int | Y | N |  |  |

附加说明：此处订单号NO是booking表的外键，而车次Trips是train表的外键。Price字段的单位为元。

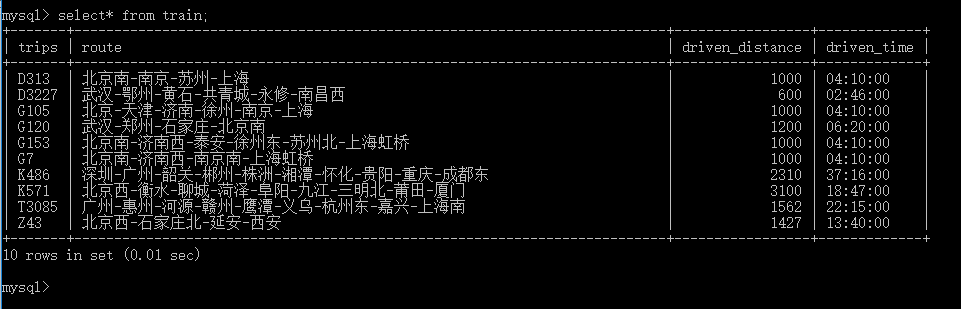
<5>列车表（train）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | | train | | | |
| 数据库用户 | | root | | | |
| 主键 | | trips | | | |
| 外键 | |  | | | |
| 排序字段 | | trips | | | |
| 索引字段 | | trips | | | |
| 字段名称 | 数据类型 | 允许为空 | 唯一 | 默认值 | 约束条件 |
| trips | varchar | N | Y |  | 主键 |
| route | text | Y | N |  |  |
| driven \_distance | int | Y | N |  |  |
| driven\_time | time | Y | N |  |  |

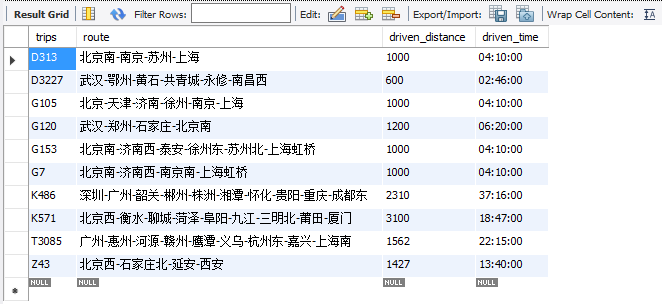
附加说明：route字段为列车的行车路线，driven\_distance为行车距离，单位为km。

3、答：执行数据批量插入脚本、窗口界面表格式手工录入、命令行交互式录入均可成功建库及建表。

使用数据批量插入脚本后成功界面截图（对train表进行查询）：



使用命令行交互式录入成功界面截图（对train表进行查询）：



结论：实际填入测试数据之后，发现所有数据表的数据插入并无问题，以证实数据模式设计合理，且插入的数据均满足完整性约束的条件。

4、答：通过插入数据之后查询的结果来看，数据模式并无问题。

5、答：

设计1：查询用户编号为10001的姓名、身份证号、联系方式以及登录密码。

|-确定所需表：user：用来查询用户编号、姓名、身份证号、联系方式

account：用来查询登录密码

|-确定关联字段：user表的id与account表的id关联：user.id=account.id

查询语句如下：

SELECT s.id AS id,s.name AS name,s.idcode AS idcode,s.telephone AS telephone,a.password AS password /\*AS后面均为别名\*/

FROM user s, account a /\*s和a分别是user表和account表的别名\*/

WHERE s.id=a.id AND s.id=10001; /\*限制条件\*/

截图如下：



设计2：查询用户编号为10000的用户所订车票的始发站、终点站以及行车路线。

|-确定所需表：user:用来查询用户编号

Information：用来查询始发站、终点站

train：用来查询行车路线

booking：ID字段与user表进行关联

|-确定所需的关联字段：

|-user表与booking表通过ID进行关联：user.id=booking.id;

|-booking表与information表通过no进行关联:booking.no=information.no

|-information表与train表通过trips进行关联：information.trips=train.trips

查询语句如下：

SELECT \*

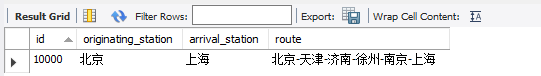
FROM (SELECT u.id,i.originating\_station AS originating\_station,i.arrival\_station AS arrival\_station,t.route

FROM user u,booking b,information i,train t

WHERE u.id=b.id AND b.no=i.no AND i.trips=t.trips) t

WHERE t.id=10000; /\*此例中用子查询的结果作为外层查询的数据源\*/

截图如下：



设计3：查询所有用户所订车票的车次、出发站、到达站、出发时间、到达时间、行车时间、座位类型、票价、余票数目，并按用户id进行升序排序。

|-确定所需表：user:查询用户编号

booking表：id字段与user表进行关联

information表：查询车次、出发站、到达站、出发时间、到达时间、座位类型、票价、余票数目

train表：查询行车时间

|-确定关联字段：

|-user表与booking表通过id字段进行关联：user.id=booking.id;

|-booking表与information表通过no进行关联，booking.no=information.no

|-information表与train表通过trips进行关联：information.trips=train.trips

查询语句如下：

SELECT \*

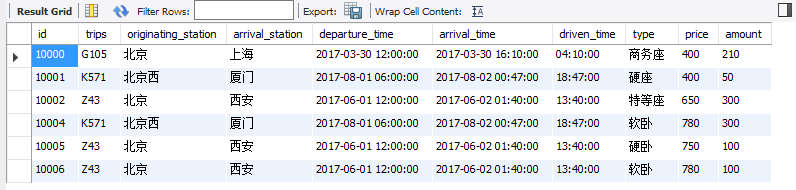
FROM (SELECT u.id,i.trips AS trips,i.originating\_station AS originating\_station,i.arrival\_station AS arrival\_station,i.departure\_time AS departure\_time,i.arrival\_time AS arrival\_time,t.driven\_time AS driven\_time,i.type AS type,i.price AS price,i.amount AS amount

FROM user u,booking b,information i,train t

WHERE u.id=b.id AND b.no=i.no AND i.trips=t.trips) t

ORDER BY t.id; /\*此例中用子查询的结果作为外层查询的数据源\*/

截图如下：



6、答：①给train表建立唯一索引：

CREATE UNIQUE INDEX train\_s ON train(trips);

建立索引后，显示train表的内容：



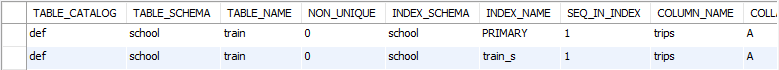
发现trips字段由原来的乱序到现在的升序排列，说明唯一索引已成功建立。

为证实索引确实已成功建立，可通过查询information\_schema数据库中的statistics表（数据字典），查询如下：

use information\_schema;

select \* from statistics；

查询结果如下：

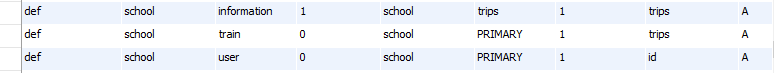


发现索引名为train\_s的索引已在数据字典当中。

当然，也可以直接通过show index from train语句查看索引。

接着删除该索引：drop index train\_s ON train;

同理，查询statistics表，结果如下：



发现索引名为train\_s的索引已被删除，不在数据字典中。

②创建聚簇索引：

MySQL InnoDB一定会建立聚簇索引，把实际数据行和相关的键值保存在一块，这也决定了一个表只能有一个聚簇索引，即MySQL不会一次把数据行保存在两个地方。

     1)  InnoDB通常根据主键值(primary key)进行聚簇；

     2) 如果没有创建主键，则会用一个唯一且不为空的索引列做为主键，成为此表的聚簇索引；

     3) 上面二个条件都不满足，InnoDB会自己创建一个虚拟的聚集索引。

因此可知，每个表均已根据各自主键进行了聚簇，无需再手动添加聚簇索引了。

7、答：对于train表而言，车次信息在实际生活中是以万计的。各种车型（如Z、D、G、T、K、C等）的车至少有几千种型号。

因此，在模拟实际列车表时，各种车型的车均需涉及，并可按一定的比例进行采样。

同时，车型命名也应该按照实际情况：单数号的车是由北往南开的，偶数号的车是由南往北开的。

对于user表而言，用户信息至少以十亿计，因此对于此类超大规模数据的表，必须进行仿真处理，不然用脚本添加是不合理的。

因此，我们可以对此进行抽样，各个属性不同取值的数据都需考虑，例如：性别不能只有男的而没有女的，籍贯不能都是北京的等等。

总而言之，对于大规模数据的表，在用脚本添加数据时应尽量满足实际情况，尽量全面地刻画实际信息。