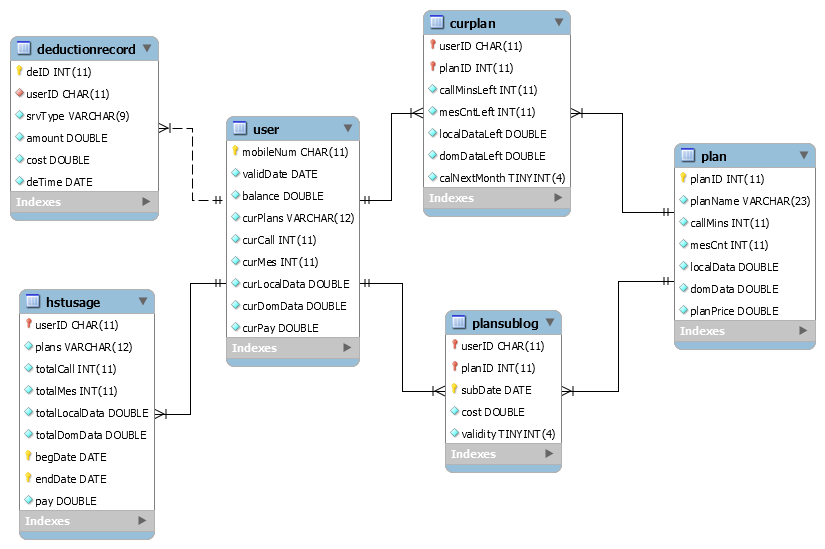
**移动运营商业务数据库设计与实现**

**说明文档**

**一、数据库的ER图及说明：**



1. user表将用户手机号（11位数字）作为主键，其余属性分别为月结日期（初始化时为下个月一号）、账户余额（可为负，即欠费）、当前套餐（数字字符串，用,分隔id）、本月目前的通话时长、发送短信数、本地/全国流量用量以及消费金额。
2. plan表属性分别为id、名字、含有的通话时长、短信数、本地/全国流量以及套餐价格。

测试用例中插入的五个套餐如下：

1. 话费套餐：月功能费20元，最多可拨打100分钟电话，超出时间按照0.5元/分钟计费。
2. 短信套餐：月功能费10元，最多可发送200条短信，超出条数按0.1元/条计费。
3. 本地流量套餐：月功能费20元，最多可获得2G流量，仅在本地使用，超出流量按2元/M计费。
4. 国内流量套餐：月功能费30元，最多可获得2G流量，超出流量按5元/M计费。
5. 多资费优惠类型套餐：月功能费40元，含50分钟电话，100条短信，1G本地流量以及1G国内流量
6. plansublog表存储套餐订购记录，当退订（立即生效）套餐时，不会删除记录，而是通过将validity属性改为0标记为失效。
7. curplan表存储用户当前套餐的详细信息（剩余通话时长、短信数、本地/全国流量），其中calNextMonth属性默认为0，每月1号定时器例程会进行套餐续订，可通过退订（次月生效）套餐将其更改为1，从而避免续订套餐。
8. deductionrecord表存储扣费记录，srvType属性目前有"call, mes, localData,domData"四种服务类型，amount表示用量，cost为使用套餐服务剩余量后的消费金额。
9. hstusage表记录用户的历史用量（套餐id串、通话时长、短信数、本地/全国流量、始末时间及消费金额[可用于账单生成]），由每月1号的定时器例程更新。

**二、数据库的操作及运行：**

**注1：项目采用spring boot+mybatis实现，限于时间没写View层，故通过测试验证正确性**

**用户**"13528677777"**由前置测试插入，供后面测试使用**

**注2：操作均加@Transactional(propagation = Propagation.REQUIRES\_NEW)事务注解，确保各个表数据的一致性，Spring Boot启动类加@EnableTransactionManagement注解**

1. 对某个用户进行套餐的查询（包括历史记录）、订购、退订（考虑立即生效和次月生效）操作。

套餐的查询（包括历史记录）直接查plansublog表即可

下面说说套餐订购和退订的实现思路：

订购：涉及对plansublog、user、curplan三个表的操作，plansublog表新增一条套餐订购记录，user修改balance、curPay及curPlans属性并更新到表中，curplan表以plan表中套餐详情新增一条记录

退订：分立即生效和次月生效两种。次月生效比较简单，直接将curplan表中相应记录calNextMonth属性设为1即可；立即生效的话，涉及对plansublog、user、curplan三个表的操作，将plansublog中对应记录validity属性改为0标记为失效，利用curplan中套餐剩余用量计算出用户使用量后将curplan相应记录删除，最后就是用户了，将curPlans属性中对应套餐id删除，balance的话先加上套餐退订返还金额，然后在通过DeductionSevice对退订套餐使用量进行扣费，更新用户到表，over！

public void C\_planOps(){  
 User user=userService.searchUser("13528677777");  
 try {  
 PlanSubItem planSubItem1=new PlanSubItem("13528677777",1,DateHandler.*transDate*("2018-9-1"),20.0,0);  
 PlanSubItem planSubItem2=new PlanSubItem("13528677777",2,DateHandler.*transDate*("2018-9-8"),10.0,0);  
 planSubMapper.addPlanSubItem(planSubItem1);  
 planSubMapper.addPlanSubItem(planSubItem2);  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 Double balance\_old=user.getBalance();  
 planService.subscribePlan("13528677777",3);  
 planService.subscribePlan("13528677777",4);  
 planService.cancalPlan("13528677777",3,true);  
 planService.cancalPlan("13528677777",4,false);  
 user=userService.searchUser("13528677777");  
 Double balance\_new=user.getBalance();  
 Assert.*assertTrue*(balance\_old-balance\_new==30.0);  
 ArrayList<PlanSubItem> subLogs=planSubMapper.getPlanSubItems("13528677777");  
 Assert.*assertTrue*(subLogs.size()==4);  
 for(PlanSubItem item:subLogs){  
 if(item.getPlanID()!=4){  
 Assert.*assertTrue*(item.getValidity()==0);  
 }else{  
 Assert.*assertTrue*(item.getValidity()==1);  
 }  
 }  
 Assert.*assertTrue*(user.getCurPlans().equals("4"));  
 ArrayList<DetailedPlan> plans=detailedPlanMapper.getDetailedPlans("13528677777");  
 Assert.*assertTrue*(plans.size()==1);  
 Assert.*assertTrue*(plans.get(0).getDomDataLeft()==2048.0);  
}

首先以历史时间插入两条套餐订购记录（validity均设为0），再通过planService订购两次套餐，分别以两种不同方式退订，然后开始测试：

a.套餐3退订立即生效，故只消费了套餐4，消费金额为30.0

b.套餐查询（包括历史记录）结果为4，且仅有套餐4的订购记录有效

c.用户当前套餐只有4，且剩余国内流量为2048.0

测试通过！！！

1. 某个用户在通话情况下的资费生成以及某个用户在使用流量情况下的资费生成。

这两个操作均通过DeductionService实现，思路相似，索性一起说明。

实现函数基本使用userID,amount两个参数（国内流量还有个Flag判明是否是本地流量溢出），涉及对deductionrecord、user、curplan三个表的操作，首先从curplan获取含用户当前套餐详细信息的数组，遍历并进行相应服务剩余量的修改，保存，直到amount为0或遍历完成，若各个套餐内剩余对应服务量大于消费量，则cost为0，否则溢出量根据基准资费计算cost，修改user的balance及curPay属性，更新存储到表，最后deductionrecord新增一条记录即可（本地流量溢出则有可能会增加两或三条记录[二次溢出…]）！

注：本地流量属于特殊情况，如果套餐内剩余量小于消费量，则会以溢出量调用国内流量消费，若国内流量依旧不够，溢出部分用本地流量资费计算

@Test  
public void D\_deductionCheck(){  
 planService.subscribePlan("13528677777",5);  
 User user=userService.searchUser("13528677777");  
 Double before\_balance=user.getBalance();  
 deductionService.callDeduction("13528677777",52);  
 deductionService.mesDeduction("13528677777",20);  
 deductionService.localDataDeduction("13528677777",1500.0);  
 deductionService.domDataDeduction("13528677777",2000.0,false);  
 ArrayList<DeductionRecord> deductionRecords=deductionMapper.getDeductions("13528677777");  
 Assert.*assertTrue*(deductionRecords.size()==5);  
 for(DeductionRecord d:deductionRecords){  
 if(d.getSrvType().equals("call")){  
 Assert.*assertTrue*(d.getAmount()==52);  
 Assert.*assertTrue*(d.getCost()==(52-50)\*0.5);  
 }else if(d.getSrvType().equals("mes")){  
 Assert.*assertTrue*(d.getAmount()==20);  
 Assert.*assertTrue*(d.getCost()==0.0);  
 }else if(d.getSrvType().equals("localData")){  
 Assert.*assertTrue*(d.getAmount()==1024.0);  
 Assert.*assertTrue*(d.getCost()==0);  
 }else{  
 if(d.getAmount()==2000.0)  
 Assert.*assertTrue*(d.getCost()==0.0);  
 }  
 }  
 DetailedPlan p=detailedPlanMapper.getDetailedPlan("13528677777",5);  
 Assert.*assertTrue*(p.getCallMinsLeft()==0);  
 Assert.*assertTrue*(p.getMesCntLeft()==80);  
 Assert.*assertTrue*(p.getLocalDataLeft()==0.0);  
 Assert.*assertTrue*(p.getDomDataLeft()==596.0);  
 user=userService.searchUser("13528677777");  
 Double after\_balance=user.getBalance();  
 Assert.*assertTrue*(before\_balance-after\_balance==1);  
}

首先在测试用户拥有套餐4（2G国内流量）的基础上再订购套餐5（50分钟通话，100条短信，1G本地流量及1G国内流量），接下来进行资费生成（52分钟通话，20条短信，1500M本地流量，2000M国内流量），好了，该测试了：

1. 产生5条扣费记录：

通话52分钟，超出套餐2分钟，扣费1RMB；

20条短信，属于套餐内，扣费0；

1024M本地流量，用完本地流量，扣费0；

476M本地流量，溢出部分，使用国内流量，扣费0；

2000M国内流量，未溢出（2G+1G,两个套餐），扣费0

1. 套餐5剩余服务量：

通话0，短信80，本地流量0，国内流量（3\*1024-1500-2000）=596M

1. 用户仅通话时长溢出两分钟，故仅消费1RMB

测试通过！！！

1. 某个用户月账单的生成

月账单的生成这块其实也挺简单的，通过BillReview提供的方法（分获取所有月账单和限制时间段两种）即可返回由Usage2Bill格式化HstUsage得到的String数组。那么，重点就在于hstusage这个表的更新了，这里采用Spring Boot的定时器实现。

Spring Boot启动类加@EnableScheduling注解，然后利用注解@Scheduled(cron = "0 0 0 1 \* ?")创建定时服务，上面的cron表达式表示每月1号00：00：00触发更新事件，将所有用户的用量存入hstusage表并刷新当前用量，续订套餐等……

具体代码如下：

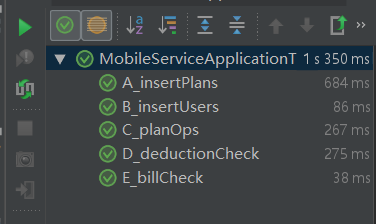
@Component  
public class ScheduleService {  
 @Autowired  
 private UserMapper userMapper;  
 @Autowired  
 private DetailedPlanMapper detailedPlanMapper;  
 @Autowired  
 private PlanMapper planMapper;  
 @Autowired  
 private PlanSubMapper planSubMapper;  
 @Autowired  
 private HstUsageMapper hstUsageMapper;  
  
  
 @Transactional(propagation = Propagation.*REQUIRES\_NEW*)  
 @Scheduled(cron = "0 0 0 1 \* ?")  
 public void refreshInfo(){  
 ArrayList<User> users=userMapper.getAllUsers();  
 for(User user:users){  
 hstUsageMapper.addHstUsage(new Usage(user.getMobileNum(),user.getCurPlans(),user.getCurCall(),user.getCurMes(),user.getCurLocalData(),user.getCurDomData(),DateHandler.*addAndSubtractMonthsByCalendar*(new Date(new java.util.Date().getTime()),-1),new Date(new java.util.Date().getTime()),user.getCurPay()));  
 user.setCurPay(0.0);  
 ArrayList<DetailedPlan> cur\_plans=detailedPlanMapper.getDetailedPlans(user.getMobileNum());  
 ArrayList<String> new\_plans=new ArrayList<>();  
 for(DetailedPlan p:cur\_plans){  
 if(p.getCalNextMonth()==1){  
 detailedPlanMapper.deleteDetailedPlan(p.getUserID(),p.getPlanID());  
 }else{  
 new\_plans.add(String.*valueOf*(p.getPlanID()));  
 Plan temp=planMapper.getPlan(p.getPlanID());  
 user.setBalance(user.getBalance()-temp.getPlanPrice());  
 user.setCurPay(user.getCurPay()+temp.getPlanPrice());  
 p.setCallMinsLeft(temp.getCallMins());  
 p.setMesCntLeft(temp.getMesCnt());  
 p.setLocalDataLeft(temp.getLocalData());  
 p.setDomDataLeft(temp.getDomData());  
 planSubMapper.addPlanSubItem(new PlanSubItem(user.getMobileNum(),p.getPlanID(),new Date(new java.util.Date().getTime()),temp.getPlanPrice(),1));  
 detailedPlanMapper.updateDetailedPlan(p);  
 }  
 }  
 String planList=String.*join*(",",new\_plans);  
 user.setCurPlans(planList);  
 user.setValidDate(DateHandler.*addAndSubtractMonthsByCalendar*(new Date(new java.util.Date().getTime()),1));  
 user.setCurCall(0);  
 user.setCurMes(0);  
 user.setCurLocalData(0.0);  
 user.setCurDomData(0.0);  
 userMapper.updateUser(user);  
 }  
 }  
}

Emmm……定时服务是基于系统时间进行的，实在不好测试，于是便只测试了对表hstusage的各项操作，测试代码如下，就不细说了：

//应用时会使用BillReview，其通过Usage2Biil将HstUsage转换为易于理解的字符串  
//测试时为方便，测试HstUsage的DAO层即可  
@Test  
public void E\_billCheck(){  
 try {  
 Date beg=DateHandler.*transDate*("2018-7-1");  
 Date end=DateHandler.*transDate*("2018-8-1");  
 Usage testU=new Usage("13528677777","3,5",52,20,2500.0,1000.0,beg,end,61.0);  
 hstUsageMapper.addHstUsage(testU);  
 ArrayList<Usage> usages=hstUsageMapper.getHstUsage("13528677777");  
 Assert.*assertTrue*(usages.size()==1);  
 Assert.*assertTrue*(usages.get(0).getPay()==61.0);  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
}

测试同样通过！！！

记录相应操作的时间……然而测试里面都执行了多次操作哎，不如就贴出测试运行时间吧☺



**三、数据库的可能优化及补充说明：**

**可能的优化思路：**

1. 数据库读写分离：

一般的系统“读”的压力远远大于“写”，因此可以通过实现数据库的读写分离来提高系统的性能。

1. 应用Enum类型：

就SrvType属性而言，其取值固定（"call, mes, localData,domData"），用VarChar存储不仅占更多的空间，查找效率还不如Enum枚举类型。

1. 外键联接：

在经常需要联接的表建立外键，从而提高查找效率。

**补充说明：**

1. 上传的.sql仅含建表相关，不含基础数据，MobileServiceApplicationTests中的A\_insertPlans和B\_insertUsers会完成plan表和测试用User的插入
2. MobileServiceApplicationTests不可重复运行，若要再次运行，可先通过truncate table curplan; truncate table deductionrecord; truncate table hstusage; truncate table plan; truncate table plansublog;truncate table user; 将各表中数据清空，否则会因为数据键值重复无法插入而导致测试失败
3. 以上☺