数据库系统原理作业#4

一、视图设计与操作题

1、针对12306余票查询系统，再结合UI的数据需求，设计视图如下：

I-针对用户的账号和密码，创建行列子集视图view1如下：

CREATE VIEW view1

AS

SELECT \*

FROM account;

II-针对用户是否为VIP用户，创建带表达式的视图view2如下：

CREATE VIEW view2(id,name,VIP)

AS

SELECT id,name, 'VIP'

FROM user

WHERE idcode='360734198003040001';

III-对从不同站点出发的列车分组并统计其平均票价，创建分组视图view3如下所示：

CREATE VIEW view3(originating\_station,AVGPrice)

AS

SELECT originating\_station,AVG(price)

FROM information

GROUP BY originating\_station;

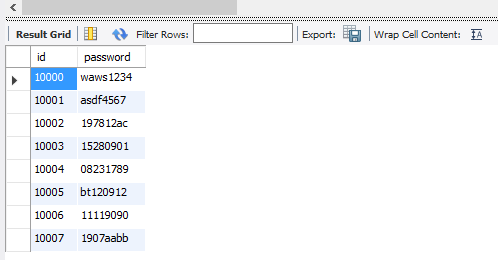
**（详情请参见DBlab3\_16281262.sql）**

2、针对上述视图，一一进行查询测试：

对于行列子集视图view1,通过下列语句进行测试：

SELECT \* FROM view1;

测试截图如下：

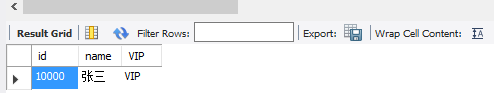


发现其与直接查询account表所得的结果等价。

对于带表达式的视图view2，通过下列语句进行测试：

SELECT \* FROM view2;

测试截图如下：



测试表明，只有idcode='360734198003040001'的张三用户才是VIP,符合视图的定义。

对于分组视图view3，通过下列语句进行测试：

SELECT \* FROM view3;

测试截图如下所示：

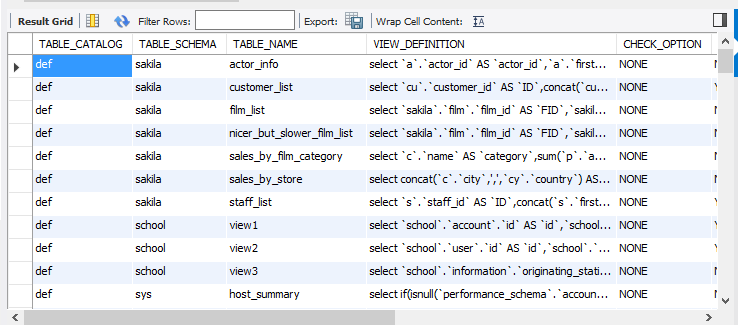
测试表明，该视图的查询结果符合预期。



其实，判断视图有没有成功建立，也可以直接数据字典，查询语句如下：

select \* from information\_schema.views;

测试截图如下所示：



发现school数据库中的view1、view2、view3视图确实已在数据字典中。

3、尝试对上述3个视图做更新操作：

**测试行列子集视图view1：**

对于行列子集视图view1，插入一个元组，插入语句如下：

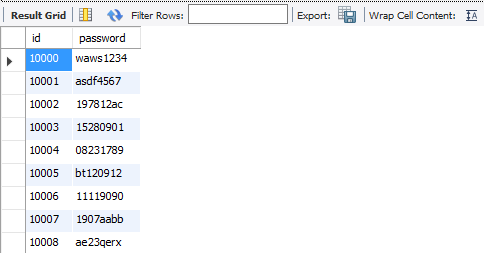
INSERT

INTO view1

VALUES(10008,'ae23qerx');

再对view1作查询，结果如下：

发现此元组确实已成功插入。



对行列子集视图view1,把10000账户的登录密码置为10000000.

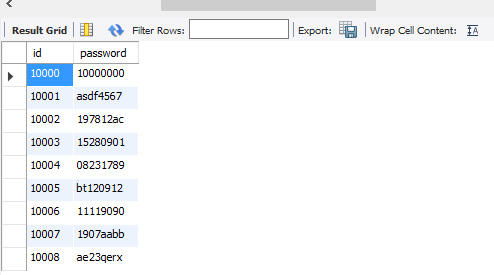
即执行如下更新操作：

UPDATE view1

SET password='10000000'

WHERE id=10000;

查看结果：

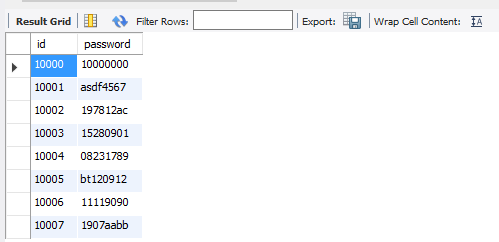


发现可以对行列子集视图进行修改操作。

对行列子集视图view1，看能否删除其中一个元组，执行如下代码：

DELETE FROM view1 WHERE id=10008;

再度查看view1的内容：



发现可以对view1做删除操作。

**测试带表达式的视图view2：**

对带表达式的视图view2,插入一个元组，

INSERT INTO view2 VALUES(10001,'李四','VIP');

查看运行结果：



此时返回错误代码如上图所示。

可见，对带表达式的视图view2插入失败。

对带表达式的视图view2做更新操作，

UPDATE view2 SET id=10002;

查看运行结果：



同上，更新失败。

再尝试对其作删除操作：

DELETE FROM view2;

查看运行结果：



同上，删除失败。

**测试分组视图view3：**

对分组视图作更新操作：

UPDATE view3

SET AVGPrice=700

WHERE originating\_station='北京';

测试结果如下：



可见，更新失败。

对分组视图view3作删除操作：

DELETE FROM view3；

测试结果如下：



可见，删除失败。

**比较分析：**

经过测试表明，在3种视图中，只有行列子集视图view1可以做更新操作（插入、修改、删除），带表达式视图view2和分组视图view3都不可以被更新。

原因：对于行列子集视图view1，对它的更新可以唯一地有意义地转换为对相应基本表（此处为account表）的更新，因此是可以被更新的，更新视图就相当于更新基本表；而对于带表达式的视图和分组视图，某些属性列是基本表中没有的，因此不能唯一地有意义地转换为对基本表的更新，因此对它们做更新都会被拒。

4、视图的主要作用：

①简单性。视图不仅可以简化用户对数据的理解，也可以简化它们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

②安全性。通过用户视图只能查询和修改他们所能见到的数据。数据库中的其它数据则既看不到也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上，但不能授权到数据库特定的行和特定的列上。通过视图，用户可以被限制在数据的不同子集上，使用权限可被限制在另一视图的一个子集上，或是一些视图和基表合并后的子集上。

③逻辑数据独立性。视图可以帮助用户屏蔽真实表结构变化带来的影响。

二、用户口令管理设计题

1、用java实现不可逆加密算法如下：

（1）BASE64（严格地说，属于编码格式，而非加密算法）

**介绍**：Base64是网络上最常见的用于传输8Bit字节代码的编码方式之一，可用于在HTTP环境下传递较长的标识信息。采用Base64编码具有不可读性， 即所编码的数据不会被人用肉眼所直接看到。常见于邮件、http加密，截取http信息，你就会发现登录操作的用户名、密码字段通过BASE64加密的。

**原理**： Base64算法主要是将给定的字符以与字符编码（如ASCII码，UTF-8码）对应的十进制数为基准，做编码操作：

①将给定的字符串以字符为单位，转换为对应的字符编码（如ASCII码）。

②将获得的字符编码转换为二进制码。

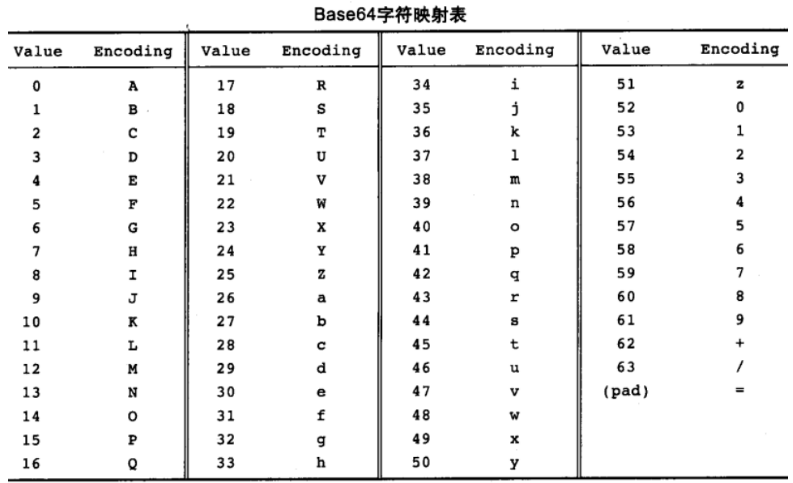
③对获得的二进制吗做分组转换操作，每3个8位二进制码为一组，转换为每4个6位二进制为一组（不足6位时地位补0）。这是一个分组变化的过程，3个8位二进制码和4个6位二进制码的长度都是24位（3\*8=4\*6=24）。

④对获得的4-6二进制码补位，向6位二进制码添加2位高位0，组成4个8位二进制码。

⑤将获得的4-8二进制码转换为十进制码。

⑥将获得的十进制码转换为Base64字符表中对应的字符。

**Base64字符映射表如下所示：**



**Java实现代码（具体参照Encrypt Algo工程- BASE64.java文件）：**

代码测试：

输入：中华人民共和国

输出：1tC7qsjLw/G5srrNufo=

（2）MD5(Message Digest algorithm 5，信息摘要算法) MD5是计算机广泛使用的的杂凑算法之一（又译摘要算法、哈希算法），广泛应用于 加密和解密技术，常用于文件校验。不管校验文件有多大，经过MD5后都能生成唯的MD5值。

MD5算法具有如下几个特点：

①压缩性：任意长度的数据，算出的MD5值长度都是固定的。

②容易计算：从原数据计算出MD5值很容易。

③抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

④弱抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

⑤强抗碰撞：想找到两个不同的数据，使它们具有相同的MD5值，是非常困难的。

因此，基于MD5算法的特性，可知用该算法对密码加密是很有效的。

**Java实现代码（具体参照Encrypt Algo工程- MD5.java文件）：**

代码测试：

输入：hello

输出：5d41402abc4b2a76b9719d911017c592

（3）SHA(Secure Hash Algorithm，安全散列算法)

SHA算法主要适用于数字签名标准（Digital Signature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA）。该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，并把它们转化为 长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。

2、答：设计用户修改密码的流程：

首先用户需要通过旧密码登录系统，再进入修改密码服务。

第一步：让用户填入旧密码；

第二步：让用户填入新密码（要求：6-16位的字母和数字的组合，密码区分大小写），若输入不合规，提示输入错误，重新输入；

第三步：再次输入新密码，输入的新密码应与第二步输入的密码相同，若不相同，提示输入错误，重新输入；

第四步：输入验证码（验证码可以随机生成）；

第五步：提交修改，若输入的旧密码不对，则提示：修改密码失败，请重新尝试！

若输入的验证码不对，则提示：验证码不对，请重新输入！

若数据准确无误，则提示：密码修改成功！

**具体过程请参照alterPasswd.vsdx.**

3、答：设计用户重置密码流程：

第一步：用户点击找回密码；

第二步：用户输入用户名或手机号，判断用户名是否存在，手机号码是否匹配；若用户名不存在或手机号码不匹配，则提示用户名不存在或手机号码错误；若无误，则进入第三步；

第三步：点击发送短信，用户会收到修改密码链接；

第四步：点击链接，输入新密码，完成重置密码流程。

**具体参见 resetPasswd.vsdx .**

三、用户授权操作题**（详情参见privileges.sql）**

1、创建用户

CREATE USER C1,C2,C3,C4,C5;

2、给用户C1、C2、C3授予表级权限

①给用户C1、C2授予查询、插入、修改、删除user表的权限：

GRANT SELECT,INSERT,UPDATE,DELETE

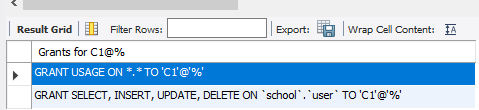
ON table user

TO C1,C2;

查看用户C1、C2的权限：（只拿C1为例）

SHOW grants for C1；

截图如下：



截图表明给用户C1、C2授权成功。

撤销用户C1、C2的权限：（拿C1为例）

REVOKE SELECT,INSERT,UPDATE,DELETE

ON table user

From C1,C2;

查看用户C1的权限，截图如下：



截图表明用户C1、C2的权限已被撤销。

②给用户C3授予操作user表的所有权限：

GRANT ALL privileges

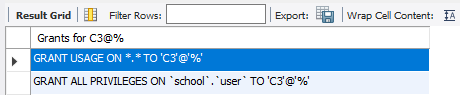
ON table user

to C3;

查看用户C3的权限：

SHOW GRANTS FOR C3;

截图如下：



收回用户的权限：

REVOKE ALL privileges

ON TABLE user

FROM C3;

查看用户C3的权限，截图如下：



截图表明用户C3的权限已被收回。

3、为用户C4授予和收回属性列级的权限：

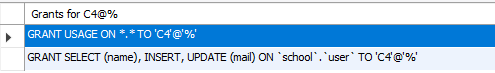
如：给用户C4授予查询user表中姓名、修改user表中mail属性以及给user表插入数据的权限，然后收回——

GRANT SELECT(name),UPDATE(mail),INSERT

ON table user

TO C4;

查看用户C4的权限，截图如下：



收回用户C4拥有的权限：

|  |
| --- |
| REVOKE SELECT(name),UPDATE(mail),INSERT  ON table user  FROM C4; |

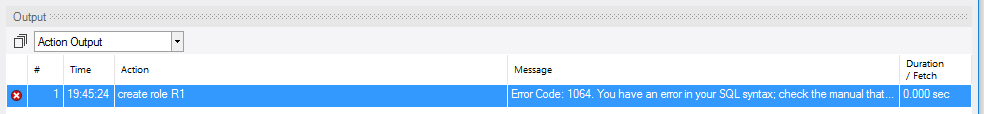
再次查看C4的权限，截图如下：



4、创建角色，将角色授予某个用户或收回用户的某个角色：

|  |
| --- |
| --创建角色  CREATE role R1;  --给角色授权  GRANT SELECT(name),UPDATE(mail),INSERT  ON TABLE user  TO R1;  --将角色授予用户C5  GRANT R1  TO C5  With admin option;  --收回角色的权限  REVOKE SELECT(name),UPDATE(mail),INSERT  ON table user  FROM R1; |

测试结果如下：



发现创建角色失败，错误代码：1064，即sql语句有语法错误。

原因：mysql中其实没有角色的概念，要想给用户授权，可以直接创建用户然后再通过grant语句给对应用户授权。

四、数据库完整性实验题

之前的数据模型已经符合实体完整性、参照完整性和内容的完整性了，现在完成对部分完整性约束进行命名，以及创建域以实现某些属性内容完整性。

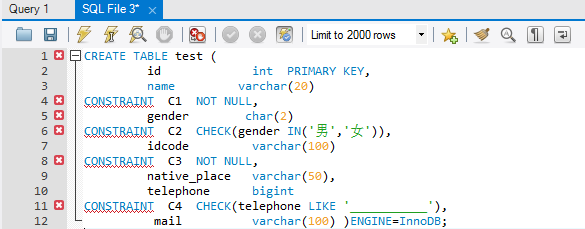
为表述方便，我另外创建一个测试表test。测试成功之后，再修改Lab3中的数据模式。

**I-对部分完整性约束命名**

以test表为例，给某些约束取别名 **(测试代码见下)** ：

|  |
| --- |
| --创建experiment数据库  CREATE database experiment;  --创建测试表test  CREATE TABLE test (  id int PRIMARY KEY,  name varchar(20)  CONSTRAINT C1 NOT NULL,  gender char  CONSTRAINT C2 CHECK(gender IN('男','女')),  idcode varchar(100)  CONSTRAINT C3 NOT NULL,  native\_place varchar(50),  telephone bigint  CONSTRAINT C4 CHECK(telephone LIKE '\_\_\_\_\_\_\_\_\_\_\_'),  mail varchar(100) )ENGINE=InnoDB; |

测试上述脚本（实验截图如下）：





发现数据库系统识别不出constraint，因此mysql中不能实现给约束取别名的操作。

**II-创建域**

|  |
| --- |
| --创建一个性别域，并声明性别域的取值范围  CREATE DOMAIN GenderDomain CHAR(2)  CHECK(VALUE IN('男','女')); |

测试上述代码，截图如下：





发现DB也识别不出domain，因此，在mysql中，不能实现域的创建。

**III-利用违约语句调试约束（测试user表中的约束）**

1、测试实体完整性

在原表基础上，再插入一个id=10000的另外一个元组：

INSERT INTO user VALUES(10000,'刘鹏','男','219807123465781209','四川',12398090956,'1234567819@qq.com');

测试结果如下：

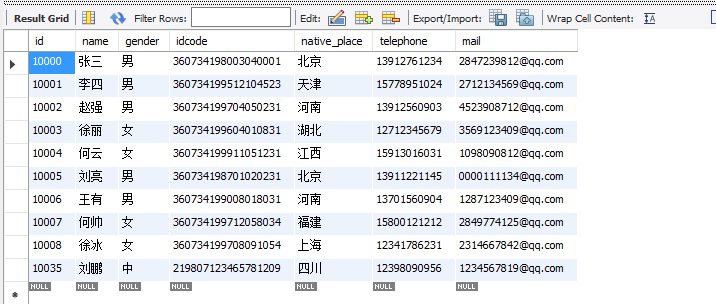


此时会出错，因为主键不允许出现重复的。

2、测试内容完整性

|  |
| --- |
| --插入一个id=10035的用户，测试此时性别列（男/女）能否成功  INSERT INTO user VALUES(10035,'刘鹏','中','219807123465781209','四川',12398090956,'1234567819@qq.com');  --查看user表的内容  Select \* from user; |

测试结果截图如下：



发现（刘鹏，中，……）已成功插入。

因此，mysql中check无效。

改进方法：改成枚举类型，对男/女记为（0,1）即可实现。

3、测试参照完整性

由于account表和user表通过id字段关联，此时尝试删除user表，看能否成功。

|  |
| --- |
| Drop table user; |

测试结果如下：



由于外键的原因，因此user表不能被成功删除。

五、实验中遇到的问题小结

|  |  |  |  |
| --- | --- | --- | --- |
| NO | Question | Why | Resolution |
| 1 | 通过sql语句查询用户权限时：show grants for C1,C2,C3;出错 | Error Code:1064,语法错误 | 把对3个用户的权限查询语句分开写：show grants for C1;show grants for C2…… |
| 2 | 创建角色语句报错：  create role R1; | Error Code:1064,  Mysql不支持角色的创建 | 使用Oracle进行角色的创建 |
| 3 | 给约束命名语句报错：  idcode varchar(100)  CONSTRAINT C3 NOT NULL,…… | ErrorCode:1064,  Mysql不支持constraint语句，不支持对约束命名 | 尝试使用Oracle来测试constraint语句 |
| 4 | 域的定义报错：  CREATE DOMAIN GenderDomain CHAR(2)  CHECK(VALUE IN('男','女')); | Error Code：1064；语法错误 | 使用Oracle |
| 5 | Check子句无效 | Mysql不支持check | 使用枚举类型来表示 |