- 实验目的：
- 实验平台：
- 实验内容和要求：
- 实验过程：
  - 1. 定义若干表，其中包括primary key, foreign key和check的定义。
  - 2. 让表中插入数据，考察primary key如何控制实体完整性。
  - 3. 删除被引用表中的行，考察foreign key中on delete子句如何控制参照完整性。
  - 4. 修改被引用表中的行的primary key，考察foreign key中on update子句如何控制参照完整性。
  - 5. 修改或插入表中数据，考察check子句如何控制校验完整性。
  - 6. 定义一个assertion，并通过修改表中数据考察断言如何控制数据完整性。
  - 7. 定义一个trigger，并通过修改表中数据考察触发器如何起作用。

实验3：SQL数据完整性

实验目的：

1. 熟悉通过SQL进行数据完整性控制的方法。

实验平台：

1. 数据库管理系统：MySQL

实验内容和要求：

1. 定义若干表，其中包括primary key, foreign key和check的定义。
2. 让表中插入数据，考察primary key如何控制实体完整性。
3. 删除被引用表中的行，考察foreign key中on delete子句如何控制参照完整性。
4. 修改被引用表中的行的primary key，考察foreign key中on update子句如何控制参照完整性。
5. 修改或插入表中数据，考察check子句如何控制校验完整性。
6. 定义一个assertion，并通过修改表中数据考察断言如何控制数据完整性。
7. 定义一个trigger，并通过修改表中数据考察触发器如何起作用。

实验过程：

1. 定义若干表，其中包括**primary key, foreign key**和**check**的定义。

- 我们建立3个表，分别代表人物/武器/圣遗物；并且key均指向人物的名称，并对其中的某些元素进行限定：

```sql
create database genshin;
use genshin;
create table person(
    name varchar(40),
    attr varchar(40),
    person_level int,
    person_artifact int,
    primary key(name),
    check(attr in('风', '雷', '火', '水', '岩', '草', '冰')),
    check(person_level > 0 AND person_level <= 90),
    check(person_artifact >= 0 AND person_artifact <= 6)
);

create table weapon(
    name varchar(40),
    weapon_name varchar(40),
    weapon_level int,
    weapon_artifact int,
    foreign key(name) references person(name),
    check(weapon_level > 0 AND weapon_level <= 90)
);

create table relics(
    name varchar(40),
    relics_name varchar(40),
    relics_number int,
    foreign key(name) references person(name),
    check(relics_number >= 0 AND relics_number <= 4)
);
```

- 效果如下：

2. 让表中插入数据，考察**primary key**如何控制实体完整性。

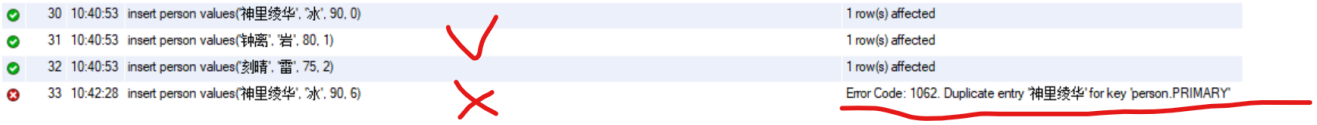
- 先插入几个数据：

```
insert person values('神里绫华', '冰', 90, 0);
insert person values('钟离', '岩', 80, 1);
insert person values('刻晴', '雷', 75, 2);
```

- 接下来，我们再插入一次“神里绫华”的状态：

```
insert person values('神里绫华', '冰', 90, 6);
```

- 运行后发现报错：这是因为primary key重复了，同一个表中不会存在两个相同的primary key：



3. 删除被引用表中的行，考察**foreign key**中**on delete**子句如何控制参照完整性。

- 我们插入一条关于“神里绫华”武器状态的描述：

```
insert weapon values('神里绫华', '雾切之回光', 90, 0);
```

- 然后删除person表中的“神里绫华”：

```
delete from person
where name = "神里绫华";
```

注：在实验的时候并没有出现“安全模式”的意外状况，因此在这里只粘贴本实验需要证明的图片：

- 发现报错，这是因为weapon表的foreign key指向了person表，因此只有在weapon表中的数据删除后，才能够删除person表中对应的数据。

4. 修改被引用表中的行的**primary key**，考察**foreign key**中**on update**子句如何控制参照完整性。

- 我们试图将作为primary key的“名称”更改；将属性为冰的角色名改为“甘雨”：

```
update person
set name = '甘雨'
where attr = '冰';
```

- 结果报错：

5. 修改或插入表中数据，考察**check**子句如何控制校验完整性。

注：此处与**MySQL**例中不同，**check**子句确实起到了作用：

- 我们试图插入错误数据，令“纳西妲”的命之座为7（check设定必须小于等于6）：

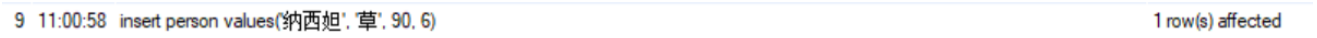
```
insert person values('纳西妲', '草', 90, 7);
```

- 结果报错：（提示：第三个check子句没有通过，符合预期）

- 我们再把命之座数据换成6，发现又能够插入了：

```
insert person values('纳西妲', '草', 90, 6);
```

- 效果如下：



6. 定义一个**assertion**，并通过修改表中数据考察断言如何控制数据完整性。

注：我使用的**MySQL**经查证后发现不支持**assertion**，因此在这里将语句内报错的图片粘贴在此处：

```
create assertion level_range
check
(not exists(select* from person)
where level > 90
);
```

7. 定义一个trigger，并通过修改表中数据考察触发器如何起作用。

- 我们先更新下weapon表中的内容，设定如下：

name	weapon_name	weapon_level	weapon_artifact
神里绫华	雾切之回光	90	0
神里绫华	无锋剑	10	1
刻晴	雾切之回光	20	0
钟离	黑缨枪	30	0

- 我们设置下面这个触发器：

```
 delimiter //
create trigger level_present
    after update on person
    for each row
begin
    update weapon set weapon_artifact = 6
    where weapon.name in (select name from person where person_level < 80);
end; //
 delimiter ;
```

它的意思是：

- 当我们对person更新时：
- 对weapon表中的name, 如果name在person中的等级小于80
- 那么对weapon表中的这个name，设置它武器的精炼值为6.
- 然后我们尝试改变person表中的人物，看看会发生什么：

```
update person set person_level = 30
where name = '神里绫华';
```

- 效果如下：

name	weapon_name	weapon_level	weapon_artifact
神里绫华	雾切之回光	90	6
神里绫华	无锋剑	10	6
刻晴	雾切之回光	20	6
钟离	黑缨枪	30	0

我们看到，现在等级更新为**30**的“神里绫华”，以及等级一直为**75**的“刻晴”，她们武器的精炼值都更新为**6**。这说明触发器起到了作用。