实验名称

一、 实验目的

1. 熟悉通过 SQL 进行数据完整性控制的方法。

二、 实验环境

1. 数据库管理系统：MySQL

三、 实验流程

1. 定义数据表

首先需要建立包含了主键声明、check 等语句的表。选择数据库后，执行如图 3.1.1 中的两条创建表指令。执行后可见左侧 Navigator 中出现了新建的表。

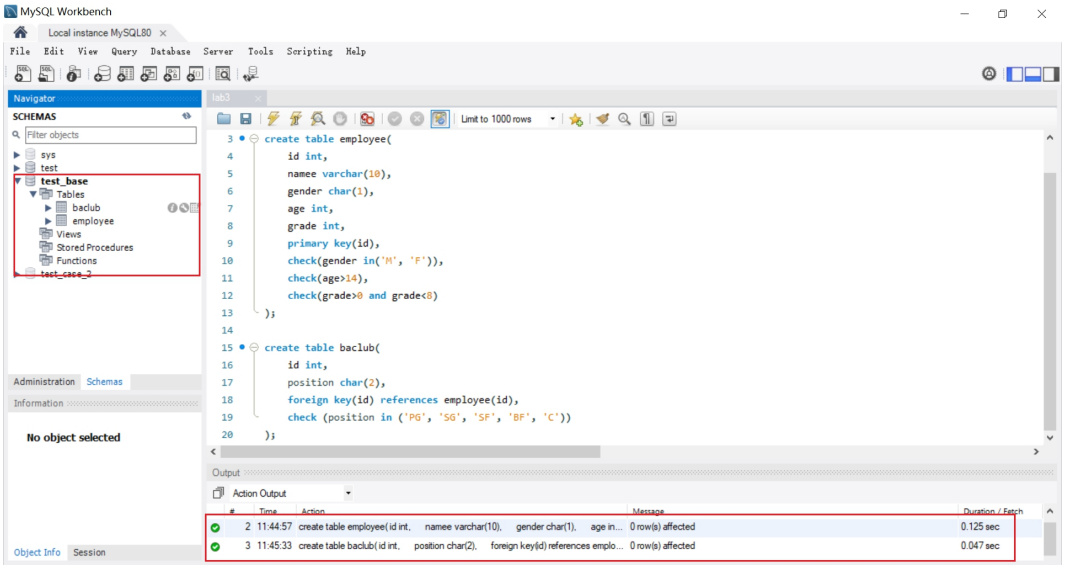


图 3.1.1 新建数据库表

其中，表 *employee* 是包含了全体员工数据的表，而表 *baclub* 是包含了篮球俱乐部信息的表。

2. 考察主键控制实体完整性

如图 3.2.1，向表 *employee* 中插入一些员工信息。

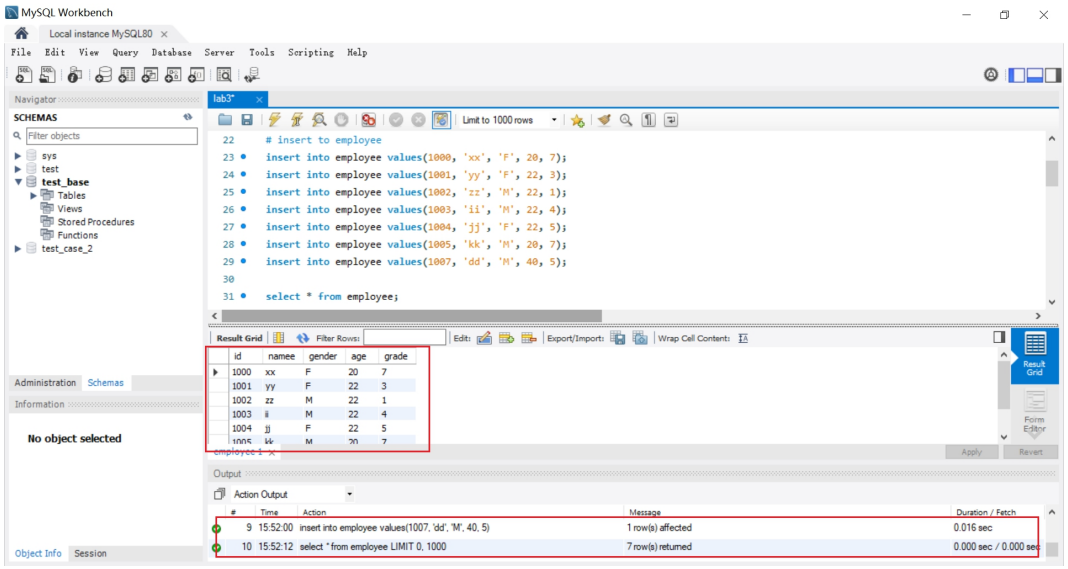


图 3.2.1 插入数据

此后，向表中插入一条带有重复 id 的员工信息，可以看见数据库提示报错，如图 3.2.2，报错原因为 Duplicate entry，主键不可重复。由此可知，主键控制了实体完整性。

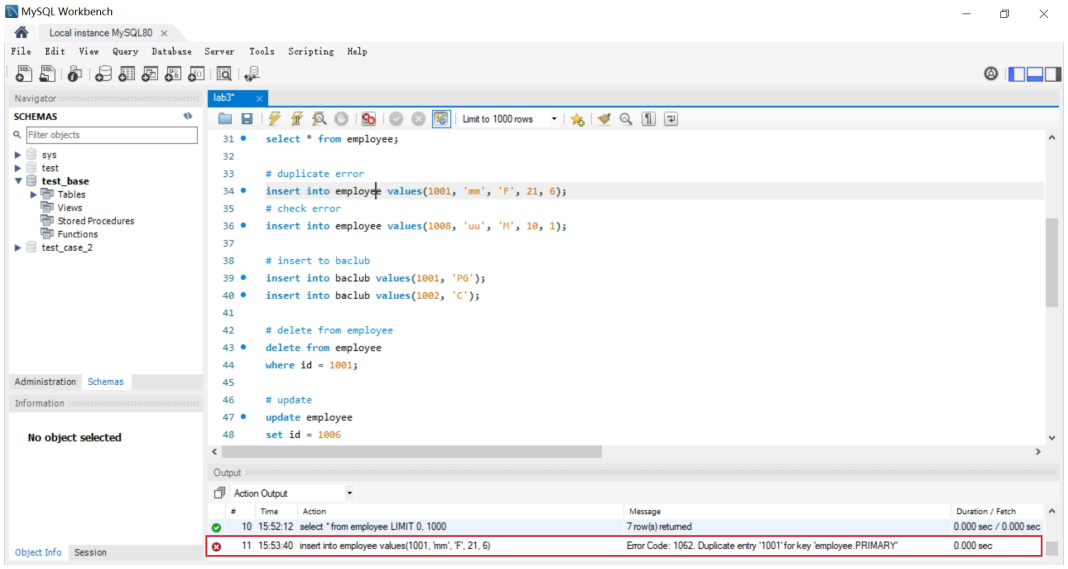


图 3.2.2 插入重复数据

3. 考察外键的 on delete 子句控制参照完整性

如图 3.3.1，首先向表 *baclub* 中插入一些成员数据。

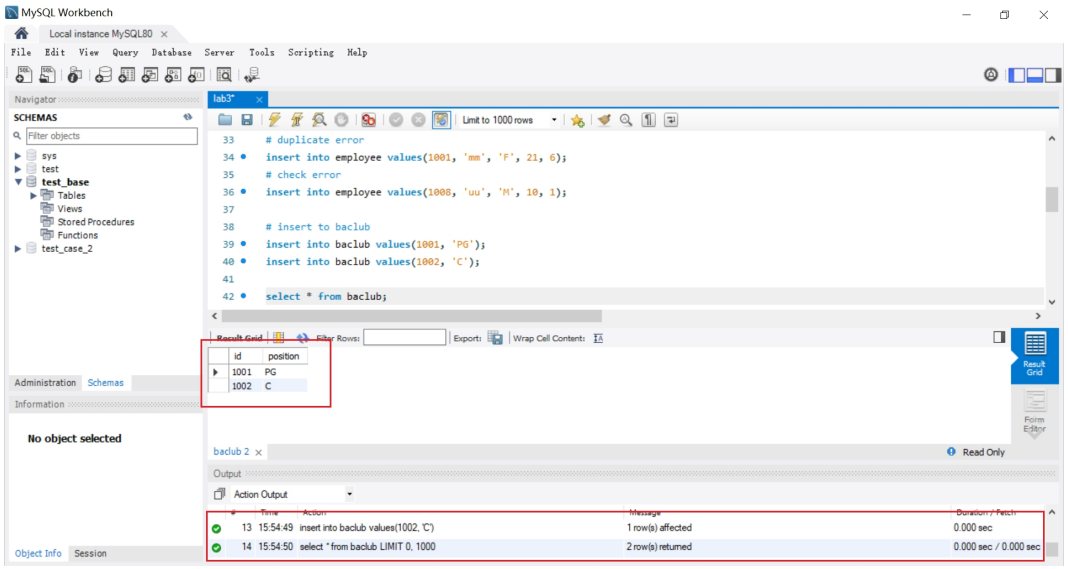


图 3.3.1 插入成员数据

此时，*baclub* 中的两位成员与 *employee* 的两位员工已经产生连接。如图 3.3.2，如果此时尝试删除 *employee* 表中的两位员工的数据，系统会报错。报错原因为 Cannot delete or update a parent row。即，要想删除该行必须先删除子表中引用该行的数据。可以看出，外键的 on delete 子句保持了参照完整性。

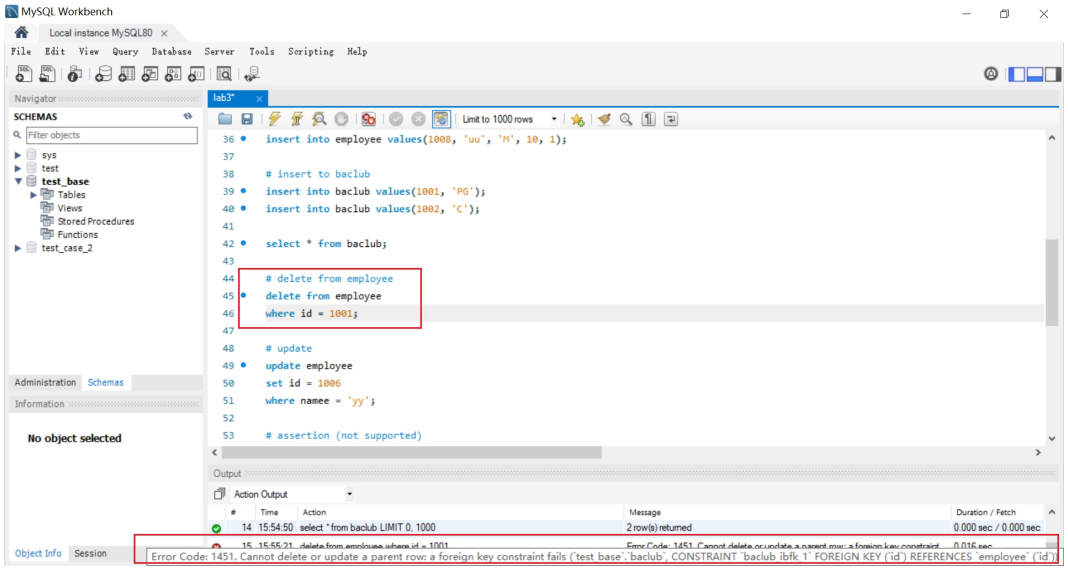


图 3.3.2 删除主表数据

4. 考察外键的 on update 子句控制参照完整性

如图 3.4.1，直接尝试更新表 *employee* 中被引用的行。由于 *baclub* 中的两位成员与 *employee* 的两位员工已经产生连接，系统会报错。报错原因为 Cannot delete or update a parent row。即，要想更新该行必须先删除子表中引用该行的数据。可以看出，外键的 on update 子句保持了参照完整性。

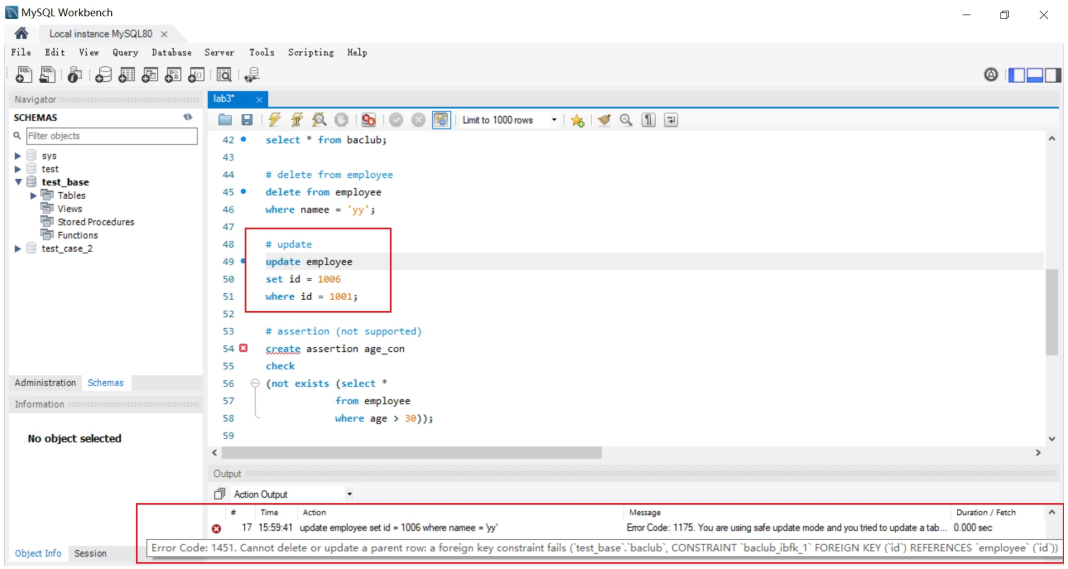


图 3.4.1 更新主表数据

5. 考察 check 子句控制数据完整性

如图 3.5.1，尝试插入一条违反了 check 条件的数据。系统报错原因为：Check constraint ‘employee_chk_2’ is violated。意思是该表的第二个 check 约束（员工年龄要大于 14 岁）被违反了，插入数据失败。可以看出，check 子句保持了数据完整性。

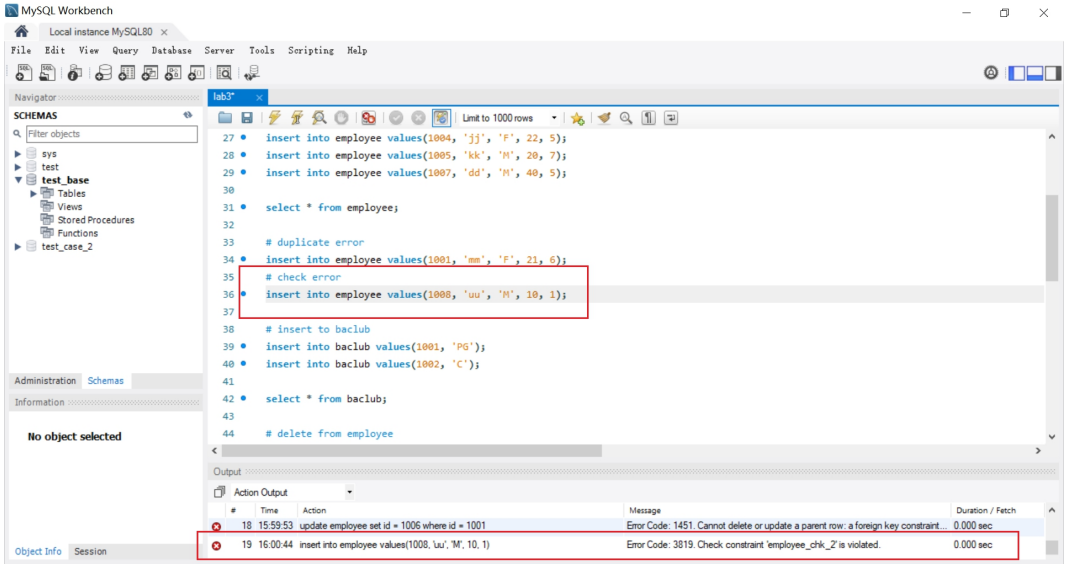


图 3.5.1 插入违法数据

6. 考察断言控制数据完整性

MySQL 不支持断言，如图 3.6.1，输入创建断言指令，系统显示指令错误，无法执行。

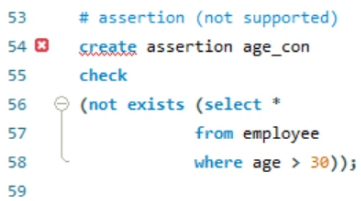
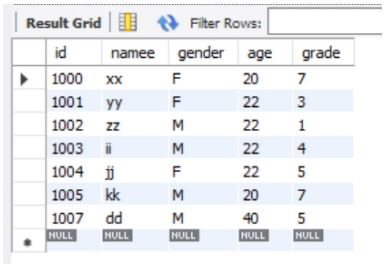


图 3.6.1 断言指令报错

7. 考察触发器的作用

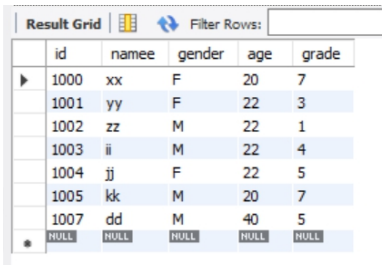
如图 3.7.1 和 3.7.2 为修改数据前的表 employee 与表 baclub。



Result Grid | Filter Rows:

id	namee	gender	age	grade
1000	xx	F	20	7
1001	yy	F	22	3
1002	zz	M	22	1
1003	ii	M	22	4
1004	jj	F	22	5
1005	kk	M	20	7
1007	dd	M	40	5
NULL	NULL	NULL	NULL	NULL

图 3.7.1 表 *employee*



Result Grid | Filter Rows:

id	namee	gender	age	grade
1000	xx	F	20	7
1001	yy	F	22	3
1002	zz	M	22	1
1003	ii	M	22	4
1004	jj	F	22	5
1005	kk	M	20	7
1007	dd	M	40	5
NULL	NULL	NULL	NULL	NULL

图 3.7.2 表 *baclub*

创建如图 3.7.3 所示的触发器，将年龄为 30 的俱乐部成员的位置自动设置为 BF。然后对主表 employee 中被关联的 id 为 1002 的列进行修改。

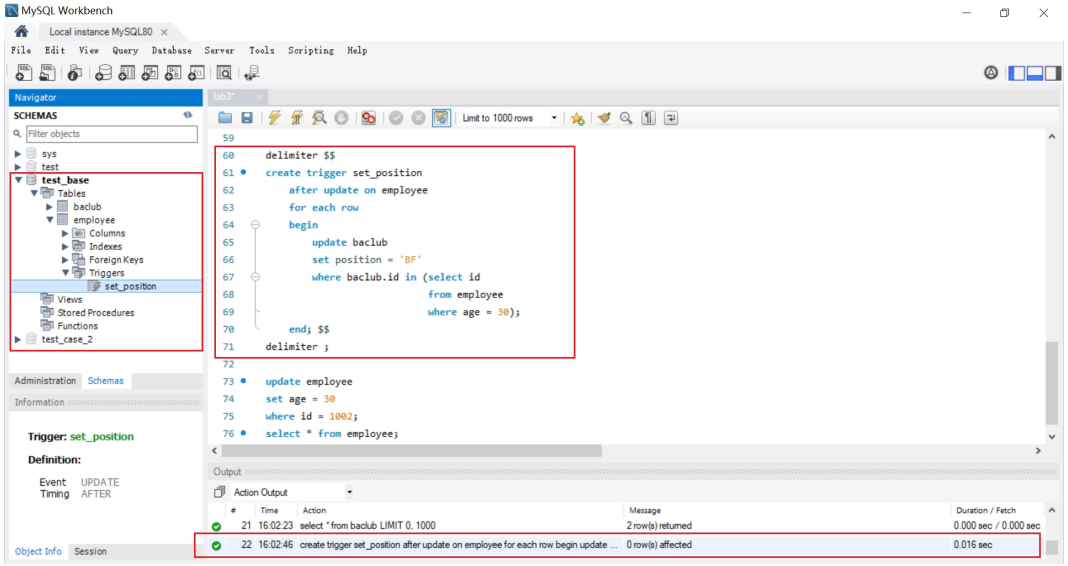


图 3.7.3 触发器

执行 update 语句后，再次查看两个数据表的内容。可见 *employee* 中 id 为 1002 的列，年龄变为 30，而 *baclub* 中其位置也自动变为 BF，如图 3.7.4 与图 3.7.5。

可以看出触发器保证了两个表满足一定的约束关系。

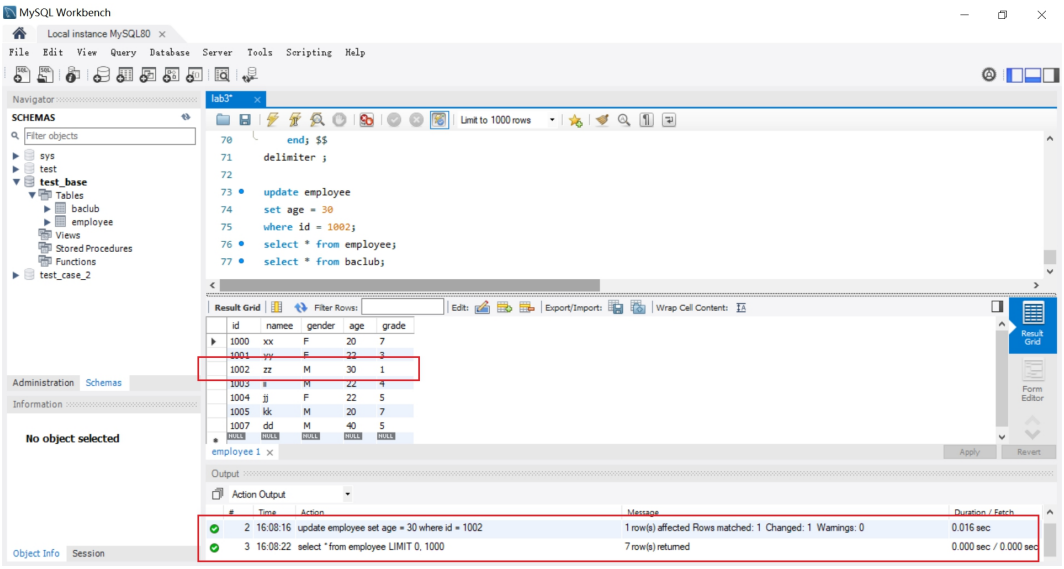


图 3.7.4 修改后的表 *employee*

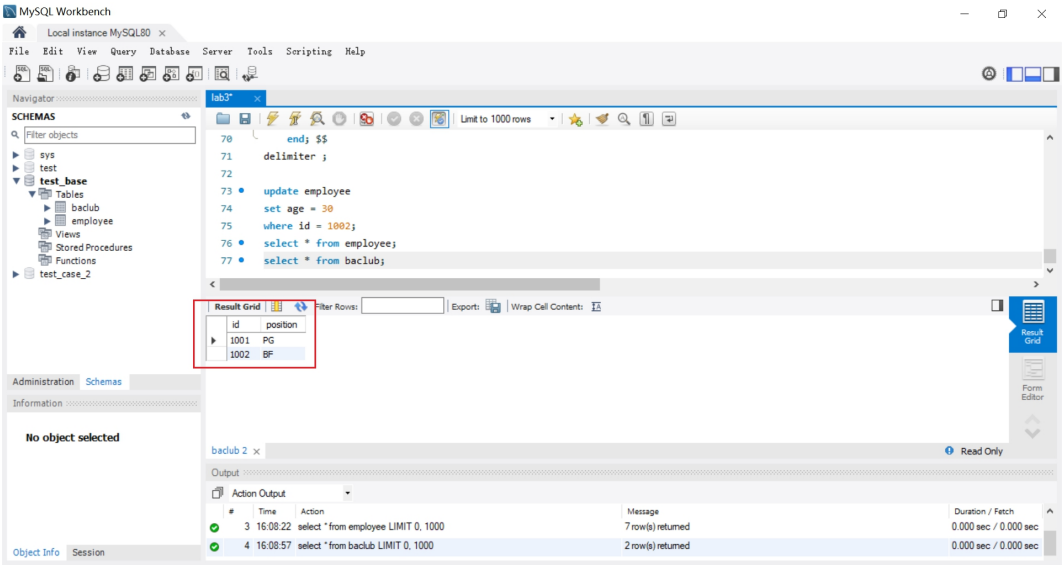


图 3.7.5 修改后的表 *baclub*

四、 遇到的问题及解决方法

1. 关于 safe updates

Workbench 默认开启 safe updates 模式，这意味着用户如果想要修改一个数据表中的数据，必须通过一个对表的键进行判断的 where 子句（这一点从 error 1175 的提示信息也可以看出）。因此，如图 3.3.2，如果删除表 *employee* 内的数据时，where 子句内是对主键 id 属性做判断，系统会提示一个关于 parent row 的报错。即，系统认为这个指令违背了 safe updates 的初衷，只是因为这一行被子表引用了所以不可删除。如图 4.1，将被删除的行改为未被引用的 id 为 1005 的行，成功删除。

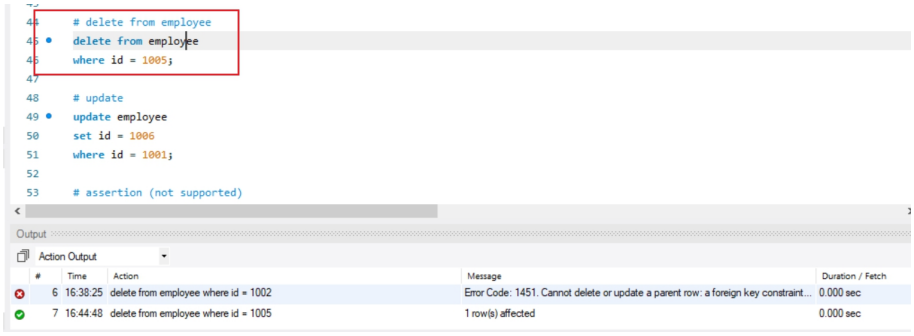


图 4.1 安全更新模式下删除数据

而如下图所示，如果删除表 *employee* 内的数据时，where 子句内是对非主键的 namee 属性做判断，系统会提示一个关于 safe updates 的报错，认为该指令违背了安全更新的初衷（防止全表更新，因为 namee 属性的值可以重复）。

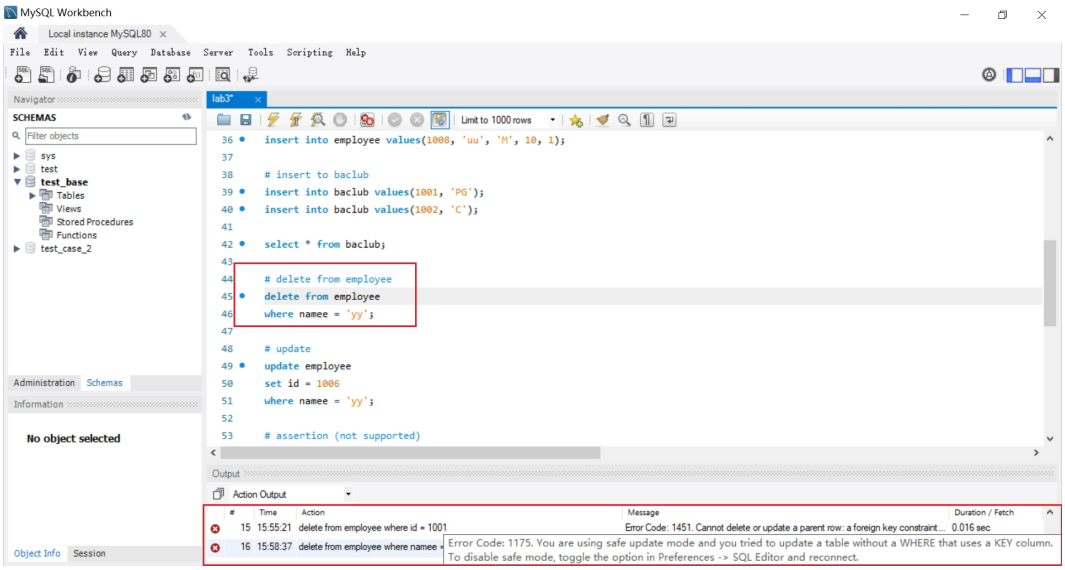
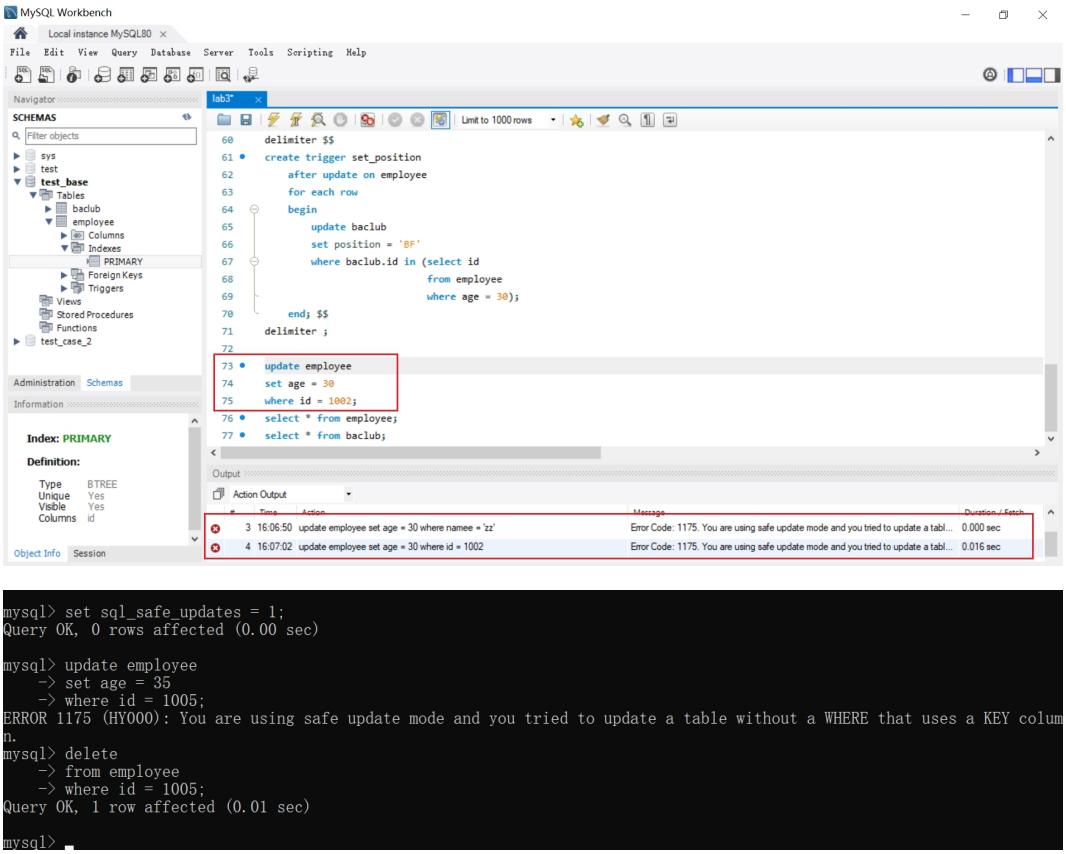


图 4.2 安全更新模式下删除失败

而奇怪的是，这一点特性似乎只适用于 delete 语句，并不适用于 update 语句。使用 update 语句时，不管 where 语句内是否对主键做了判断，系统都会提示关于 safe updates 的报错，在命令行下与 Workbench 内都如此，如下图。在查询许多网上资料并与助教讨论后，怀疑是设计缺陷。

若要使用 update 语句，似乎只能通过关闭 safe updates 来实现。




```
mysql> set sql_safe_updates = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> update employee
    -> set age = 35
    -> where id = 1005;
ERROR 1175 (HY000): You are using safe update mode and you tried to update a table without a WHERE that uses a KEY colum
n.
mysql> delete
    -> from employee
    -> where id = 1005;
Query OK, 1 row affected (0.01 sec)

mysql>
```

图 4.3 安全更新模式下更新失败

五、 总结

此次实验让我了解了许多控制数据完整、安全的方法，了解了许多数据管理时要注意的细节。