

实验名称

一、 实验目的

- 熟悉通过 SQL 进行数据完整性控制的方法。

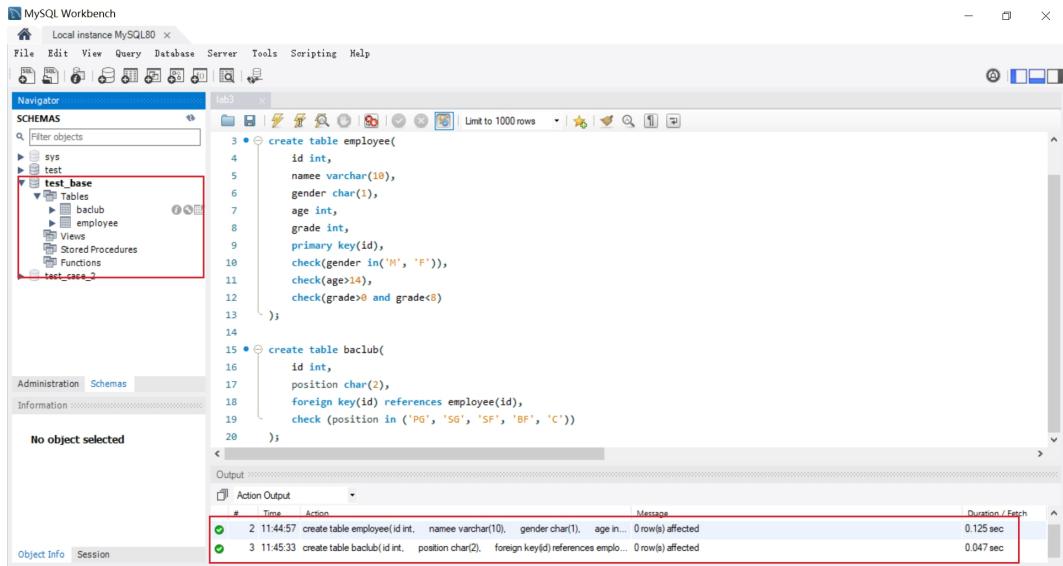
二、 实验环境

- 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。



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure. A red box highlights the "test_base" schema, which contains the "employee" and "baclub" tables.
- Query Editor (tab3):** Displays the SQL code for creating the tables:

```
3 *  create table employee(
4     id int,
5     name varchar(10),
6     gender char(1),
7     age int,
8     grade int,
9     primary key(id),
10    check(gender in('M', 'F')),
11    check(age>14),
12    check(grade>0 and grade<=5)
13 );
14
15 *  create table baclub(
16     id int,
17     position char(2),
18     foreign key(id) references employee(id),
19     check (position in ('PG', 'SG', 'SF', 'BF', 'C'))
20 );
```
- Output:** Shows the execution results:

Action	Message	Duration / Search
2 11:44:57	create table employee(id int, name varchar(10), gender char(1), age int, grade int); 0 rows affected	0.125 sec
3 11:45:33	create table baclub(id int, position char(2), foreign key(id) references employee(id)); 0 rows affected	0.047 sec

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。

The screenshot shows the MySQL Workbench interface with a query editor window titled 'lab3'. The code entered is:

```

22  # insert to employee
23  • insert into employee values(1000, 'xx', 'F', 20, 7);
24  • insert into employee values(1001, 'yy', 'F', 22, 3);
25  • insert into employee values(1002, 'zz', 'M', 22, 1);
26  • insert into employee values(1003, 'ii', 'M', 22, 4);
27  • insert into employee values(1004, 'jj', 'F', 22, 5);
28  • insert into employee values(1005, 'kk', 'M', 20, 7);
29  • insert into employee values(1007, 'dd', 'M', 40, 5);
30
31 • select * from employee;

```

The 'Result Grid' tab displays the inserted data:

	id	name	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

The 'Output' tab shows the execution log:

#	Time	Action	Message	Duration / Fetch
9	15:52:00	insert into employee values(1007, 'dd', 'M', 40, 5)	1 row(s) affected	0.016 sec
10	15:52:12	select * from employee LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec

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。

The screenshot shows the MySQL Workbench interface with a query editor window titled 'lab3'. The code entered includes a duplicate insert statement:

```

31 • select * from employee;
32
33 • # duplicate error
34 • insert into employee values(1001, 'mm', 'F', 21, 6);
35 • # check error
36 • insert into employee values(1008, 'uu', 'M', 10, 1);
37
38 • # insert to bacclub
39 • insert into bacclub values(1001, 'PG');
40 • insert into bacclub values(1002, 'C');
41
42 • # delete from employee
43 • delete from employee
44 where id = 1001;
45
46 • # update
47 • update employee
48 set id = 1006

```

The 'Output' tab shows the execution log, highlighting the error at step 34:

#	Time	Action	Message	Duration / Fetch
10	15:52:12	select * from employee LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
11	15:53:40	insert into employee values(1001, 'mm', 'F', 21, 6)	Error Code: 1062: Duplicate entry '1001' for key 'employee.PRIMARY'	0.000 sec

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 *bacclub* 中插入一些成员数据。

The screenshot shows the MySQL Workbench interface with a query editor window titled 'lab3'. The SQL code in the editor is:

```

33 # duplicate error
34 • insert into employee values(1001, 'mm', 'F', 21, 6);
35 # check error
36 • insert into employee values(1000, 'uu', 'M', 10, 1);
37
38 # insert to baclub
39 • insert into baclub values(1001, 'PG');
40 • insert into baclub values(1002, 'C');

41
42 • select * from baclub;

```

The results grid shows the following data:

id	position
1001	PG
1002	C

The 'Output' pane shows the execution log:

- 13 15:54:49 insert into baclub values(1002, 'C')
- 14 15:54:50 select * from baclub LIMIT 0, 1000

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。

The screenshot shows the MySQL Workbench interface with a query editor window titled 'lab3'. The SQL code in the editor is:

```

36 • insert into employee values(1008, 'uu', 'M', 10, 1);
37
38 # insert to baclub
39 • insert into baclub values(1001, 'PG');
40 • insert into baclub values(1002, 'C');

41
42 • select * from baclub;

43
44 # delete from employee
45 • delete from employee
46 where id = 1001;

47
48 # update
49 • update employee
50 set id = 1006
51 where namee = 'yy';

52
53 # assertion (not supported)

```

The results grid shows the following data:

id	namee	sex	age	position
1001	yy	M	10	PG
1002				C

The 'Output' pane shows the execution log and an error message:

- 14 15:54:50 select * from baclub LIMIT 0, 1000
- 15 15:55:21 delete from employee where id = 1001 Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('test_base`.`baclub', CONSTRAINT `baclub_ibfk_1` FOREIGN KEY (`id`) REFERENCES `employee` (`id`))

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。

```

42 • select * from baclub;
43
44 # delete from employee
45 • delete from employee
46 where namee = 'yy';
47
48 # update
49 update employee
50 set id = 1006
51 where id = 1001;
52
53 # assertion (not supported)
54 • create assertion age_con
55 check
56 (not exists (select *
57   from employee
58   where age > 30));
59

```

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('test_base`.`baclub', CONSTRAINT `baclub_ibfk_1` FOREIGN KEY (`id`) REFERENCES `employee` (`id`))

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。

```

27 • insert into employee values(1004, 'jj', 'F', 22, 5);
28 • insert into employee values(1005, 'kk', 'M', 20, 7);
29 • insert into employee values(1007, 'dd', 'M', 40, 5);
30
31 • select * from employees;
32
33 # duplicate error
34 • insert into employee values(1001, 'mm', 'F', 21, 6);
35 # check error
36 • insert into employee values(1008, 'uu', 'M', 10, 1);
37
38 # insert to baclub
39 • insert into baclub values(1001, 'P6');
40 • insert into baclub values(1002, 'C');
41
42 • select * from baclub;
43
44 # delete from employee

```

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('test_base`.`employee', CONSTRAINT `employee_ibfk_1` FOREIGN KEY (`id`) REFERENCES `baclub` (`id`))
Error Code: 3819. Check constraint 'employee_chk_2' is violated.

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。

```

53 # assertion (not supported)
54 • create assertion age_con
55 check
56 (not exists (select *
57   from employee
58   where age > 30));
59

```

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 bacclub。

	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
*	HULL	HULL	HULL	HULL	HULL

图 3.7.1 表 employee

	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
*	HULL	HULL	HULL	HULL	HULL

图 3.7.2 表 bacclub

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。

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** test_base is selected.
- Tables:** bacclub and employee are shown under test_base.
- Triggers:** A trigger named "set_position" is selected.
- Code Editor:** The trigger definition is displayed:

```
59
60 delimiter $$ 
61 • create trigger set_position
62     after update on employee
63     for each row
64     begin
65         update bacclub
66         set position = 'BF'
67         where bacclub.id in (select id
68                             from employee
69                             where age = 30);
70     end$$
71 delimiter ;
```
- Output:** The output pane shows two log entries:

#	Time	Action	Message	Duration / Fetch
21	16:02:23	select * from backup LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
22	16:02:46	create trigger set_position after update on employee for each row begin update ... 0 row(s) affected		0.016 sec

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 bacclub 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。

The screenshot shows the MySQL Workbench interface with the 'employee' table selected in the Result Grid. The table has columns: id, namee, gender, age, and grade. A red box highlights row 1002 (id=1002, namee='yy', gender='F', age=22, grade=3). The Action Output pane at the bottom shows two log entries:

- 2 16:08:16 update employee set age = 30 where id = 1002; 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 Duration / Fetch: 0.016 sec
- 3 16:08:22 select * from employee LIMIT 0, 1000 7 row(s) returned Duration / Fetch: 0.000 sec / 0.000 sec

图 3.7.4 修改后的表 *employee*

The screenshot shows the MySQL Workbench interface with the 'baclub' table selected in the Result Grid. The table has columns: id and position. A red box highlights row 1002 (id=1002, position='BF'). The Action Output pane at the bottom shows two log entries:

- 3 16:08:22 select * from employee LIMIT 0, 1000 7 row(s) returned Duration / Fetch: 0.000 sec / 0.000 sec
- 4 16:08:57 select * from baclub LIMIT 0, 1000 2 row(s) returned Duration / Fetch: 0.000 sec / 0.000 sec

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。

```

47 # delete from employee
48 • delete from employee
49 where id = 1005;
50
51 # assertion (not supported)
52
53 # update
54 • update employee
55 set id = 1006
56 where id = 1001;
57
58 # assertion (not supported)

```

The screenshot shows the MySQL Workbench interface with a SQL editor and an output pane. The SQL editor contains several statements, including a successful delete operation (line 48) and an update operation (line 54). The output pane shows two log entries: a delete operation at 16:38:25 and an update operation at 16:44:48. Both operations have a duration of 0.000 sec and affected 1 row(s).

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）。

```

36 • insert into employee values(1008, 'uu', 'M', 10, 1);
37
38 # insert to baclub
39 • insert into baclub values(1001, 'PG');
40 • insert into baclub values(1002, 'C');
41
42 • select * from baclub;
43
44 # delete from employee
45 • delete from employee
46 where namee = 'yy';
47
48 # update
49 • update employee
50 set id = 1006
51 where namee = 'yy';
52
53 # assertion (not supported)

```

The screenshot shows the MySQL Workbench interface with a SQL editor and an output pane. The SQL editor contains statements including a delete operation (line 45) with a condition on the non-primary key `namee`. The output pane shows two log entries: a delete operation at 15:55:21 and an update operation at 15:58:37. The delete operation fails with Error Code: 1451 (Cannot delete or update a parent row: a foreign key constraint...), while the update operation succeeds. A tooltip for the error message indicates: "Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint... Duration / Fetch 0.016 sec".

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。

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Local instance MySQL80' is selected. The left sidebar displays the 'Navigator' with 'SCHEMAS' expanded, showing 'test_base' and its tables: 'baclub' and 'employee'. The 'Tables' section for 'employee' includes columns, indexes, primary keys, foreign keys, triggers, stored procedures, and functions. A trigger named 'set_position' is defined:

```

60 delimiter $$ 
61 • create trigger set_position
62     after update on employee
63     for each row
64 begin
65     update baclub
66     set position = 'BF'
67     where baclub.id in (select id
68     from employee
69     where age = 30);
70 end$$
71 delimiter ;
72
73 • update employee
74     set age = 30
75     where id = 1002;
76 • select * from employee;
77 • select * from baclub;

```

The 'Information' panel shows an index named 'PRIMARY' with the following definition:

- Type: BTREE
- Unique: Yes
- Visible: Yes
- Columns: id

The 'Output' panel displays the results of the executed queries:

Action	Time	Message	Duration / Fetch
3	16:06:50	update employee set age = 30 where namee = 'zz'	Error Code: 1175. You are using safe update mode and you tried to update a tabl... 0.000 sec
4	16:07:02	update employee set age = 30 where id = 1002	Error Code: 1175. You are using safe update mode and you tried to update a tabl... 0.016 sec

Below the Workbench window, a terminal window shows the MySQL command-line interface:

```

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 column.
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。