第5章

数据库的完整性

数据库的完整性与数据库的安全性是两个完全不同的概念，可以从其定义及防范对象两方面加以区分。

数据库的完整性是防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据，其防范对象是不合语义的、不正确的数据。

数据库的安全性是指保护数据库，防止恶意的破坏和非法的存取，其防范对象是非法用户和非法操作。

数据的完整性可以从正确性及相容性两方面来刻画。数据的正确性是指数据是符合现实世界语义，反映了当前实际状况的；数据的相容性是指数据库同一对象在不同关系表中的数据是符合逻辑的。

例如，学生的学号必须唯一；学生所选课程必须是学校开设的课程，学生所在的院系必须是学校已成立的院系；学生的性别只能是男或女；本科学生年龄的取值范围为 14～50 的整数等。

为维护数据库的完整性，DBMS 必须完成以下三个工作。

(1) 提供定义完整性约束条件的机制。

完整性约束条件也称为完整性规则，是数据库中的数据必须满足的语义约束条件。SQL 标准使用了一系列概念来描述完整性，包括关系模型的实体完整性、参照完整性和用户定义完整性。

(2) 提供完整性检查的方法。

DBMS 中检查数据是否满足完整性约束条件的机制称为完整性检查。

完整性检查一般在 INSERT、UPDATE、DELETE 语句执行后开始检查，也可以在事务提交时检查。

(3) 违约处理。数据库管理系统若发现用户的操作违背了完整性约束条件，就采取一定的动作：

拒绝(NO ACTION)执行该操作；

级联(CASCADE)执行其他操作。

在 MySQL 中，各种完整性约束是数据库关系模式定义的一部分，可通过 CREATE TABLE 或 ALTER TABLE 语句来定义。一旦定义了完整性约束，MySQL 服务器会随时检测处于更新状态的数据库内容是否符合相关的完整性约束，从而保证数据的一致性与正确性。完整性约束机制既能有效地防止对数据库的意外破坏，又能提高完整性检测的效率，还能减轻数据库编程人员的工作负担。

定义新表时可以定义完整性。

语法格式：

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table _name
( [ column_ definition ],...
| [CONSTRAINT [symbol]] PRIMARY KEY[index_type] (key_part,...)
[index_option] ...
| [CONSTRAINT [symbol]] UNIQUE [INDEX| KEY]
[index_name] [index_type] (key_part,...)[index_option] ...
| [CONSTRAINT [symbol]] FOREIGN KEY[index_name] (col_name,...)
reference_definition
| check_CONSTRAINT_definition )
```

修改旧表时也可以定义完整性。

语法格式：

```
ALTER TABLE tbl_name
[alter_option [, alter_option] ...]

alter_option: {
ADD [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (key_part,...)
[index_option]...
|ADD [CONSTRAINT [symbol]] UNIQUE [INDEX| KEY]
[index_name] [index_type] (key_part,...)[index_option] ...
|ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (col_name...)
reference_definition
|ADD [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
|DROP {CHECK| CONSTRAINT} symbol
|ALTER {CHECK| CONSTRAINT} symbol [NOT] ENFORCED
}

reference_definition:
[ON DELETE reference_option]
[ON UPDATE reference_option]
reference_option:
RESTRICT| CASCADE| SET NULL| NO ACTION| SET DEFAULT
```

5.1 实体完整性

实体完整性强制表标识列的完整性，可通过约束、唯一约束、主码约束或标识列属性来实施实体完整性。

码是一个属性集，可唯一地确定一个元组。既然是属性集，那么有可能由单属性构成，也有可能由多属性构成。

单属性构成的码有两种声明方法：定义为表级完整性约束和列级完整性约束。对多个属性构成的码只有一种声明方法：定义为表级约束条件。

本节中的例子，表名之后都加了后缀“_integrity”，它们与没有后缀的表定义相同。

【例 5-1】 将 tbStuInfo 表中的 sNo 属性定义为码。

(1) 在列级定义主码。

```
DROP TABLE IF EXISTS tbStuInfo_integrity;

CREATE TABLE tbStuInfo_ integrity
(sNo INT UNSIGNED PRIMARY KEY,
sName VARCHAR(20),
sSex CHAR(1) ,
sClass INT UNSIGNED,
sDept CHAR(2) REFERENCES tbdepartment(dNo),
sBirthDate DATE,
sBirthPlace VARCHAR(20),
sPwd VARCHAR(50)
);
```

(2) 在表级定义主码。

```
DROP TABLE IF EXISTS tbStuInfo_integrity;

CREATE TABLE tbStuInfo_integrity
(sNo INT UNSIGNED,
sName VARCHAR(20),
sSex CHAR(1) ,
sClass INT UNSIGNED,
sDept CHAR(2) REFERENCES tbdepartment(dNo),
sBirthDate DATE,
sBirthPlace VARCHAR(20),
sPwd VARCHAR(50),
PRIMARY KEY (sNo)
);

mysql>SHOW CREATE TABLE tbStuInfo_integrity\G
*************************** 1. row ***************************
       Table: tbStuInfo_integrity
Create Table: CREATE TABLE 'tbstuinfo_integrity' (
  'sNo' INT unsigned NOT NULL,
  'sName' VARCHAR(20) DEFAULT NULL,
  'sSex' CHAR(1) DEFAULT NULL,
  'sClass' INT UNSIGNED DEFAULT NULL,
  'sDept' CHAR(2) DEFAULT NULL,
  'sBirthDate' DATE DEFAULT NULL,
  'sBirthPlace' VARCHAR(20) DEFAULT NULL,
  'sPwd' VARCHAR(50) DEFAULT NULL,
  PRIMARY KEY ('sNo')
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

学生表 tbStuInfo 中,学号 sNo 是一个单属性码。在例 5-1 的第一种定义方法中,学号 sNo 是在列级进行 PRIMARY KEY 声明的,PRIMARY KEY 紧跟在 sNo 的定义之后;在第二种定义中,采用的是表级声明方法,在所有属性定义完成后,在表尾加上 PRIMARY

KEY 的声明。可通过 workbench 查看主码定义的情况，如图 5-1 所示。

【例 5-2】 将 tbSC 表中的 sNo，cNo 属性组定义为码。

```
DROP TABLE IF EXISTS tbSC_integrity;

CREATE TABLE tbSC_integrity
(
sNo   INT UNSIGNED   COMMENT '学号'  ,
cNo   CHAR(4)   COMMENT '课程号'  ,
grade   DECIMAL(4,1)   COMMENT '成绩',
PRIMARY KEY (sNo,cNo)              /* 只能在表级定义主码 */
);
```

在学生选课关系 tbSC 中，组合属性 sNo，cNo 是码，只能在表级进行定义。也就是说，需要在所有的属性都定义完成后，才能进行主码 PRIMARY KEY 的声明，如图 5-2 所示。

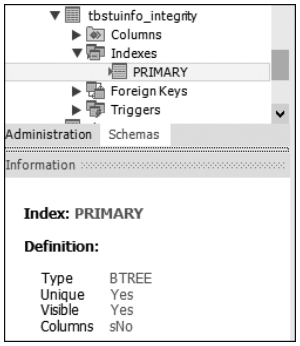


图 5-1　定义表之后从 workbench 查看 tbstuinfo_integrity 主码

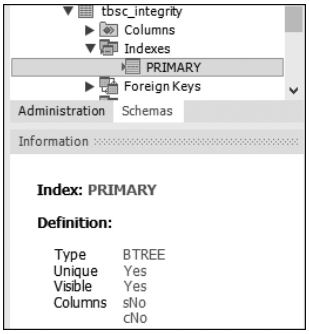


图 5-2　定义表之后查看 tbSC_integrity 主码

在进行表级主码声明时，无论是单属性，还是多属性，主码都需要用括号括起来，跟在关键字 PRIMARY KEY 之后。

向关系中插入新元组或对主码列进行更新操作时，关系数据库管理系统(relational database management system，RDBMS)按照实体完整性规则自动进行检查，包括：

(1) 检查主码值是否唯一，如果不唯一则拒绝插入或修改；

(2) 检查主码的各个属性，即主属性是否为空，只要有一个为空就拒绝插入或修改。

检查记录中主码值是否唯一的一种方法是进行全表扫描，如图 5-3 所示，依次判断表中

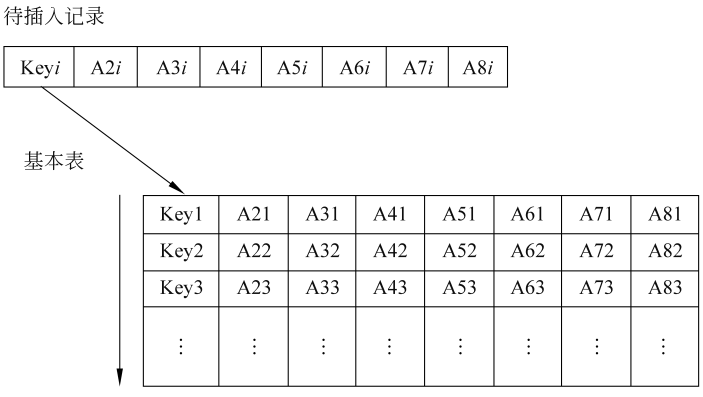


图 5-3　全表扫描示例

每一条记录的主码值与将插入记录上的主码值(或者修改的新主码值)是否相同,即到基本表中去查找,看是否包含关键字为 Key*i* 的记录。全表扫描的缺点是十分耗时。

为避免对基本表进行全表扫描,RDBMS 一般都在主码上自动建立一个索引。

例如,如图 5-4 所示,在主码上建的是一个 B+树索引,新插入记录的主码值是 80。

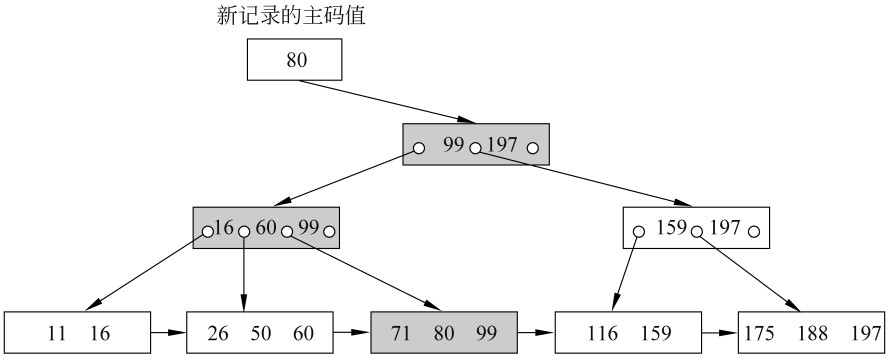


图 5-4 B+树查找关键字示例

(1) 通过主码索引,从 B+树的根结点开始查找。

(2) 读取 3 个结点:根结点(99 197)、中间结点(16 60 99)、叶结点(71 80 99)。

(3) 该主码值已经存在,不能插入这条记录。

5.2 参照完整性


视频讲解

在插入、修改和删除记录时,参照完整性保持表之间已定义的关系。它确保键值在所有表中的一致性,从而避免引用不存在的值。如果一个键值更改了,那么在整个数据库中,对该键值的引用要进行一致的更改。

关系模型的参照完整性定义通常通过表级声明来实现。在 CREATE TABLE 语句中使用 FOREIGN KEY 短语定义哪些列为外码,并通过 REFERENCES 短语指明这些外码参照哪些表的主码。

如果外码是单属性列,也可以在列级直接使用 REFERENCES 短语指明该列参照哪个表的主码,这种方法隐式地定义了一个 FOREIGN KEY。

本节中的例子,表名之后都加了后缀"_foreign",它们与没有后缀的表定义相同。

【例 5-3】 将 tbSC 表中的 sNo、cNo 属性定义为外码。

```
DROP TABLE IF EXISTS tbSC_foreign;

CREATE TABLE tbSC_foreign (
sNo INT UNSIGNED,
cNo CHAR(4) ,
grade DECIMAL(4,1),
PRIMARY KEY(sNo,cNo),                                  /*在表级定义实体完整性*/
FOREIGN KEY (sNo) REFERENCES tbStuInfo(sNo) ,          /*在表级定义*/
FOREIGN KEY (cNo) REFERENCES tbCourse(cNo)             /*在表级定义*/
);
```

在关系 tbSC 中,(sNo,cNo)是主码,sNo、cNo 分别参照 tbStuInfo 关系的主码和

tbCourse 关系的主码，可以在表级定义这两个外码，在所有的属性定义之后加上：

```
FOREIGN KEY (sNo) REFERENCES tbStuInfo(sNo),
FOREIGN KEY (cNo) REFERENCES tbCourse(cNo),
```

前者表示 sNo 是外码，参照 tbStuInfo 关系的主码 sNo；后者表示 cNo 是外码，参照 tbCourse 关系的主码 cNo，如图 5-5 所示。

图 5-5　定义表之后查看 tbSC_foreign 外码

参照完整性将“被参照表/主表/父表”与“参照表/从表/子表”两个表中的相应元组联系起来；对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行检查。

如图 5-6 所示，对表 tbSC 和 tbStuInfo 有四种可能破坏参照完整性的情况。

(1) tbSC 表中增加一个元组。该元组的 sNo 属性值在 tbStuInfo 表中不存在与之相等的记录。

例如 tbSC 表中最后一个元组，sNo 为 2020082150，但在 tbStuInfo 表中不存在与之相等的记录，因此插入该元组破坏了参照完整性。

(2) 修改 tbSC 表中的一个元组。修改后该元组的 sNo 属性值在 tbStuInfo 表中不存在与之相等的记录。

tbSC

学号 sNo	课程号 cNo	成绩 grade
2020082101	CS01	79
2020082101	CS02	84
2020082101	CS03	79
2020082101	CS04	84
2020082101	CS05	79
2020082101	CS06	84
2020082101	CS07	79
2020082122	CS01	84
2020082131	CS01	95
2020082131	CS02	81
2020082131	CS03	84
2020082131	CS04	74
2020082150	CS01	59

tbStuInfo

学号 sNo	姓名 sName	性别 sSex	班级 sClass	系部号 sDept	出生日期 sBirthDate
2020072101	贺世娜	f	20200721	07	2002-9-12
2020072113	郭兰	f	20200721	07	2003-4-5
2020082101	应胜男	f	20200821	08	2002-11-8
2020082122	郑正星	m	20200821	08	2002-12-11
2020082131	吕建鸥	m	20200821	08	2003-1-6

图 5-6　对参照表(tbSC)及被参照表(tbStuInfo)进行增删改违约示例

例如 tbSC 表中第 8 个元组的 sNo 为 2020082122，将其修改为 2020082150，在 tbStuInfo 表中不存在与之相等的记录，因此修改该元组破坏了参照完整性。

(3) 删除 tbStuInfo 表中的一个元组。导致 tbSC 表中某些元组的 sNo 属性值在 tbStuInfo 表中不存在与之相等的记录。

例如删除 tbStuInfo 表中 sNo 为 2020082101 的元组，会造成 tbSC 中前 7 个元组的 sNo 值在 tbStuInfo 表中无法找到与之相等的记录，因此删除该元组破坏了参照完整性。

(4) 修改 tbStuInfo 表中一个元组的 sNo 属性，造成 tbSC 表中某些元组的 sNo 属性的值在 tbStuInfo 表中不存在与之相等的记录。

例如 tbStuInfo 表中第 5 个元组的 sNo 为 2020082131，将其修改为 2020082151，会造成 tbSC 表中第 9～12 个元组的 sNo 值在 tbStuInfo 表中无法找到与之相等的记录，因此修改该元组的 sNo 属性破坏了参照完整性。

如表 5-1 所示,对于从表(子表),即参照表 tbSC 进行插入或更新操作,可能会破坏参照完整性。对于此类操作,DBMS 会直接拒绝执行。

表 5-1　可能破坏参照完整性的情况及违约处理

被参照表(例如 tbStuInfo)	参照表(例如 tbSC)	违 约 处 理
可能破坏参照完整性 ←	插入元组	拒绝
可能破坏参照完整性 ←	修改外码值	拒绝
删除元组 →	可能破坏参照完整性	拒绝/级联删除/设置为空值
修改主码值 →	可能破坏参照完整性	拒绝/级联修改/设置为空值

对于主表(父表),即被参照表 tbStuInfo,对其进行删除元组或修改主码值的操作,可能会破坏参照完整性。

参照完整性违约处理有以下几种策略。

(1) 拒绝(NO ACTION|RISTRICT)执行。

不允许该操作执行。该策略一般设置为默认策略。

(2) 级联(CASCADE)操作。

当删除或修改被参照表的一个元组造成了与参照表的不一致,则删除或修改参照表中的所有造成不一致的元组。

(3) 设置为空值(SET NULL)。

当删除或修改被参照表的一个元组时造成了不一致,则将参照表中的所有造成不一致的元组的对应属性设置为空值。

(4) 设置为默认值(SET DEFAULT)。

当删除或修改被参照表的一个元组时造成了不一致,则将参照表中的所有造成不一致的元组的对应属性设置为默认值。

例如,有下面 2 个关系:

系部(系部编码, 简称, 全称);

学生(学号, 姓名, 性别, 班级, 系部编码, 出生日期)。

假设表 5-2 所示的系部关系中某个元组被删除,如系部编码为 07 的元组被删除。按照设置为空值的策略,就要把学生表中系部编码= 07 的所有元组的系部编码设置为空值,如表 5-3 所示。对应的实际语义为某个系部删除了,该系部的所有学生系部未定,等待重新分配系部。

表 5-2　系部关系

系部编码 dNo	简称 dName	全称 dComment	系部编码 dNo	简称 dName	全称 dComment
01	MC	马克思主义学院	09	CE	工学院
02	CC	商学院	25	ME	现代教育技术中心
03	LL	文学院	35	ST	科技处
07	SS	理学院	36	HS	人文社科处
08	IE	信息学院			

表 5-3　学生关系

学号 sNo	姓名 sName	性别 sSex	班级 sClass	系部编码 sDept	出生日期 sBirthDate
2020072101	贺世娜	f	20200721	NULL	2002-9-12
2020072113	郭兰	f	20200721	NULL	2003-4-5
2020082101	应胜男	f	20200821	08	2002-11-8
2020082122	郑正星	m	20200821	08	2002-12-11
2020082131	吕建鸥	m	20200821	08	2003-1-6
……	……	……	……	……	……

对于参照完整性，除了应该定义外码，还应定义外码列是否允许有相应的动作。

【例 5-4】 显式说明参照完整性的违约处理示例。

(1) 删除已有的 tbSC_foreign、tbStuInfo_foreign、tbCourse_foreign。注意删除顺序，先删除参照表，再删除被参照表。

```
DROP TABLE IF EXISTS tbSC_foreign;
DROP TABLE IF EXISTS tbStuInfo_foreign;
DROP TABLE IF EXISTS tbCourse_foreign;
```

(2) 创建与现有 tbStuInfo 和 tbCourse 完全相同的表 tbStuInfo_foreign 和 tbCourse_foreign。

```
CREATE TABLE tbStuInfo_foreign AS
SELECT * FROM tbStuInfo;
ALTER TABLE tbStuInfo_foreign ADD PRIMARY KEY (sNo);

CREATE TABLE tbCourse_foreign AS
SELECT * FROM tbCourse;
ALTER TABLE tbCourse_foreign ADD PRIMARY KEY (cNo);
```

(3) 创建与 tbSC 结构类似的表 tbSC_foreign。

```
CREATE TABLE tbSC_foreign(
sNo INT UNSIGNED,
cNo CHAR(4) ,
grade DECIMAL(4,1),
PRIMARY KEY (sNo,cNo),
FOREIGN KEY (sNo) REFERENCES tbStuInfo_foreign(sNo)
    ON DELETE CASCADE                    /*级联删除 tbSC_foreign 表中相应的元组*/
    ON UPDATE CASCADE,                   /*级联更新 tbSC_foreign 表中相应的元组*/
FOREIGN KEY (cNo) REFERENCES tbCourse_foreign(cNo)
    ON DELETE NO ACTION
        /*当删除 tbCourse_foreign 表中的元组造成了与 tbSC_foreign 表不一致时拒绝删除*/
    ON UPDATE CASCADE
        /*当更新 tbCourse_foreign 表中的 cNo 时,级联更新 tbSC_foreign 表中相应的元
```

```
组 * /
);
```

(4) 为表 tbSC_foreign 添加数据。

```
TRUNCATE TABLE tbSC_foreign;
INSERT INTO tbSC_foreign(sNo,cNo,grade)
SELECT * FROM tbSC;
```

本例创建了一张学生成绩表 tbSC_foreign,外码 sNo 参照了学生表 tbStuInfo_foreign 中的 sNo,并对参照完整性的违约进行了显式说明:对于主表的 DELETE 和 UPDATE 操作,都会引起对从表 tbSC_foreign 的 CASCADE 操作。

外码 cNo 参照了课程表 tbCourse_foreign 中的 cNo:对于主表 tbCourse_foreign 的 DELETE 操作,当删除的元组造成了与 tbSC_foreign 表不一致时,拒绝删除(NO ACTION);而对主表 tbCourse_foreign 的更新操作,会引起对从表 tbSC_foreign 的 CASCADE 操作。

【例 5-5】 对主表 tbStuInfo_foreign 进行更新、删除操作,验证例 5-4 对参照完整性的显式说明。

(1) 更新 tbStuInfo 学生 2020082101 的学号为 2020082160。

```
#对主表 tbStuInfo_foreign 进行更新操作
UPDATE tbStuInfo_foreign SET sNo=2020082160 WHERE sNo=2020082101;
#查看从表成绩表
SELECT * FROM tbSC_foreign;
```

如图 5-7(a)的方框内容所示,可以看到学号 2020082101 已经级联修改成了 2020082160。

```
#恢复对主表的更新操作
#对主表 tbStuInfo_foreign 进行更新操作
UPDATE tbStuInfo_foreign SET sNo=2020082101 WHERE sNo=2020082160;
#查看从表成绩表
SELECT * FROM tbSC_foreign;
```

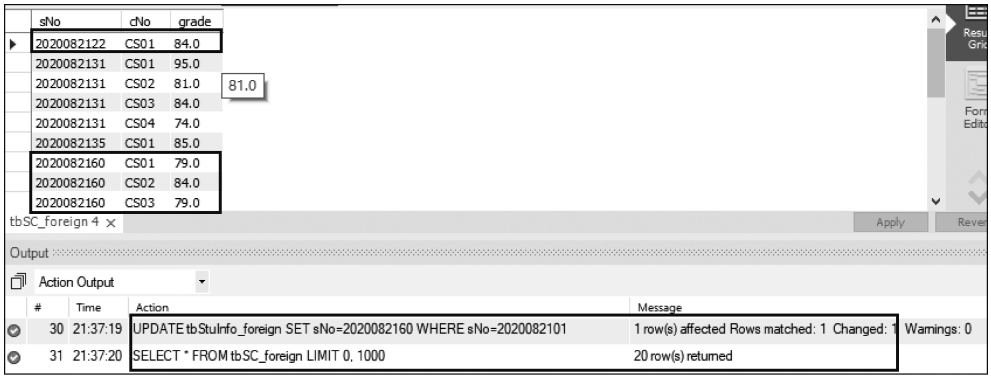
(2) 删除学生表 tbStuInfo_foreign 学号是 2020082122 的学生。

```
#对主表 tbStuInfo_foreign 进行删除操作
DELETE FROM tbStuInfo_foreign WHERE sNo=2020082122;
#查看从表成绩表
SELECT * FROM tbSC_foreign;
```

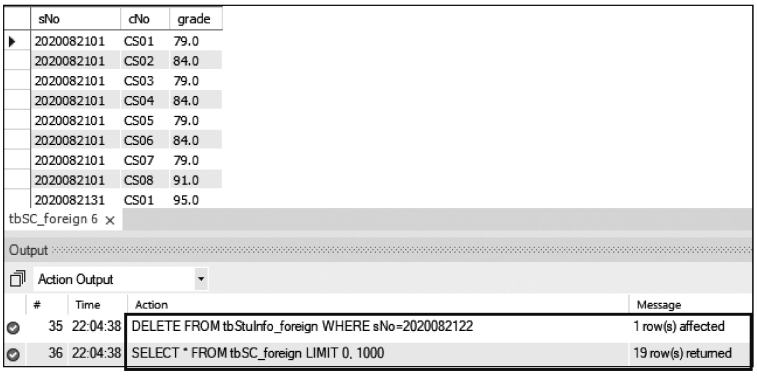
图 5-7(a)中成绩表 tbSC_foreign 框中标记的元组,该学生的成绩已经被级联删除,如图 5-7(b)中的方框内容所示,tbStuInfo_foreign 中删除了一条记录,同时 tbSC_foreign 中的记录也少了一条(由原来的 20 条变成了 19 条),说明被删除学生的成绩已经被级联删除。该结果也符合日常经验,一个学生退学之后,他的学习成绩也被同时清除。

(3) 恢复被参照表与参照表中刚刚被删除的信息。

```
INSERT INTO
tbStuInfo_foreign(sNo, sName, sSex, sClass, sDept, sBirthDate,
```



(a) 更新操作



(b) 删除操作

图 5-7　对被参照表(tbStuInfo_foreign)进行更新及删除操作示例

```
sBirthPlace)
VALUES
(2020082122,  '郑正星',  'm',  20200821,  '08',  '2002-12-11', '浙江杭州');
INSERT INTO tbSC_foreign(sNo,cNo,grade) VALUES (2020082122,'CS01',84);
```

【例 5-6】 对主表 tbCourse_foreign 进行更新、删除操作,验证例 5-4 对参照完整性的显式说明。

```
#对主表 tbCourse_foreign 进行更新操作
UPDATE tbCourse_foreign SET cNo='CS09' WHERE cNo='CS07';
#查看从表成绩表 tbSC_foreign
SELECT * FROM tbSC_foreign;
#恢复对主表 tbCourse_foreign 的更新操作
UPDATE tbCourse_foreign SET cNo='CS07' WHERE cNo='CS09';
```

如图 5-8 方框中所示,更新成功;学生成绩表 tbSC_foreign 中有一行的 cNo 值为 CS09。

```
#删除课程号是 CS01 的课程的课程号,失败
mysql>DELETE FROM tbCourse_foreign WHERE cNo='CS01';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign
key CONSTRAINT fails ('textbook_stu'.'tbsc_foreign', CONSTRAINT
```