实验1.1 实体完整性

1. 在数据库School中建立表Stu_Union，进行主键约束，在没有违反实体完整性的前提下插入并更新一条记录

```sql
create table Stu_Union(
SUid char(5) primary key,
SUname char(10)
)
insert into Stu_Union(SUid,SUname)values('1','李明');
insert into Stu_Union(SUid,SUname)values('2','张三');
```

	SUid	SUname
1	1	李明
2	2	张三

2. 演示违反实体完整性的插入操作

- 关系的主属性不能取空值

```sql
insert into Stu_Union(SUname)values('李四');
```

```
消息 515，级别 16，状态 2，第 16 行
不能将值 NULL 插入列 'SUid'，表 'School.dbo.Stu_Union'；列不允许有 Null 值。INSERT 失败。
语句已终止。
```

- 尝试插入已经存在的主键值

```sql
insert into Stu_Union(SUid,SUname)values('1','李华');
```

```
消息 2627，级别 14，状态 1，第 18 行
违反了 PRIMARY KEY 约束“PK__Stu_Unio__A6480A5963F668F6”。不能在对象“dbo.Stu_Union”中插入重复键。重复键值为 (1)。
语句已终止。
```

3. 演示违反实体完整性的更新操作

- 更新一个不存在的记录

```sql
update Stu_Union
set SUname='王力'
where SUid='3'
```

结果是0行受影响，并不违反实体完整性

- 超过属性长度限制

```sql
update Stu_Union
set SUname='王力2222222222222222'
where SUid='2'
```

```
消息 8152，级别 16，状态 30，第 27 行
将截断字符串或二进制数据。
语句已终止。
```

4. 演示事务的处理，包括事务的建立，处理以及出错时的事物回滚

提示：SQL2005相关语句为

```sql
BEGIN TRAN
ROLLBACK TRAN
COMMIT TRAN
```

可以这样演示：新建一个包含两条语句的事务，使第一条成功而第二条失败，然后查看整个事务是否回滚。

重要提示： SQL默认只回滚出错的语句，要回滚整个事务，需要预先执行以下语句：`SET XACT_ABORT ON`

事务执行前的Stu_Union表

	SUid	SUname
1	1	李明
2	2	张三

- 部分回滚

```sql
select * from Stu_Union
--事物的建立
begin tran
--在事物中执行一系列操作
insert into Stu_Union(SUid,SUname)values('3','张强');

update Stu_Union
set SUname='黄波'
where SUid='3';
--模拟出错
update Stu_Union
set SUname='王力2222222222222222'
where SUid='2'
--回滚
commit tran
select * from Stu_Union
```

使用回滚错误语句机制的事务执行后的Stu_Union表，可以看到出错的语句被回滚

	SUid	SUname
1	1	李明
2	2	张三
3	3	黄波

- 全部回滚

```sql
--回滚整个事务
select * from Stu_Union
--事物的建立
begin tran
--在事物中执行一系列操作
insert into Stu_Union(SUid,SUname)values('3','张强');

update Stu_Union
set SUname='黄波'
where SUid='3';
--模拟出错
update Stu_Union
set SUname='王力2222222222222222'
where SUid='2'
--回滚
rollback tran
select * from Stu_Union
```

使用回滚整个事务机制的事务执行后Stu_Union表，可以看到整个事务被回滚到事务执行前的状态

	SUid	SUname
1	1	李明
2	2	张三

2. 通过建立Scholarship表，插入一些数据。演示当与现有的数据环境不等时，无法建立实体完整性以及参照完整性。

```sql
create table Scholarship(
ID int primary key,
SUid char(5),
foreign key(SUid) references Stu_Union(SUid)
)
--无法建立实体完整性
insert into Scholarship(SUid)values('2')
--无法建立参照完整性
insert into Scholarship(ID,SUid)values(1,'5')
```

```
消息 515，级别 16，状态 2，第 82 行
不能将值 NULL 插入列 'ID'，表 'School.dbo.Scholarship'；列不允许有 Null 值。INSERT 失败。
语句已终止。
```

```
消息 547，级别 16，状态 0，第 84 行
INSERT 语句与 FOREIGN KEY 约束"FK__Scholarshi__SUid__634EBE90"冲突。该冲突发生于数据库"School"，表"dbo.Stu_Union"，column 'SU
语句已终止。
```

实验1.2参照完整性

1. 为演示参照完整性，建立表Course，令cno为其主键，并在Stu_Union中插入数据。为下面的实验步骤做预先准备。

```sql
create table Course(
cno int primary key
)
select *
from Stu_Union
```

2. 建立表sc，另sno和cno分别为参照Stu_Union表以及Course表的外键，设定为级连删除，并令(sno, cno)为其主键。在不违反参照完整性的前提下，插入数据。

- 建立表sc

```sql
create table Course(
cno int primary key
)
create table Stu_Union(
sno int primary key
)
create table sc(
sno int,
cno int,
score float,
constraint PK primary key (sno,cno),--令(sno,cno)为主键
constraint FK1 foreign key(sno) references Stu_Union(sno) on delete cascade,--将sno设为参照Stu_Union的外键，用constraint FK1将约束命名为FK1；用on delete cascade设置为级连删除
constraint FK2 foreign key(cno) references Course(cno) on delete cascade
)
```

- 查看sc约束

```sql
exec sp_helpconstraint sc
```

	Object Name
1	sc

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	FOREIGN KEY	FK1	Cascade	No Action	Enabled	Is_For_Replication	sno
2							REFERENCES School.dbo.Stu_Union (sno)
3	FOREIGN KEY	FK2	Cascade	No Action	Enabled	Is_For_Replication	cno
4							REFERENCES School.dbo.Course (cno)
5	PRIMARY KEY ...	PK	(n/a)	(n/a)	(n/a)	(n/a)	sno, cno

- 插入数据

```sql
insert into Course(cno) values(2000);
insert into Stu_Union(sno) values(1000);
insert into sc(sno,cno,score) values(1000,2000,99.5);
```

	sno	cno	score
1	1000	2000	99.5

3. 演示违反参照完整性的插入数据

	cno
1	2000

	sno
1	1000

```sql
insert into sc(sno,cno,score) values(1,2,99.5);
```

消息 547，级别 16，状态 0，第 121 行
INSERT 语句与 FOREIGN KEY 约束"FK1"冲突。该冲突发生于数据库"School"，表"dbo.Stu_Union"，column 'sno'。
语句已终止。

4. 在Stu_Union中删除数据，演示级连删除。

```sql
delete Stu_Union
where sno=1000;
```

	sno
1	1000

	sno	cno	score
1	1000	2000	99.5

	sno

	sno	cno	score

5. Course中删除数据，演示级连删除。

```sql
delete Course
where cno=2000;
```

	cno
1	2000

	sno	cno	score
1	1000	2000	99.5

	cno

	sno	cno	score

6. 为了演示多重级连删除，建立Stu_Card表，令stu_id为参照Stu_Union表的外键，令card_id为其主键，并插入数据。

- 建立Stu_Card表

```sql
create table Stu_Union(
stu_id int primary key
);
create table Stu_Card(
stu_id int,
card_id int primary key,
grade int,
constraint FK1 foreign key (stu_id) references Stu_Union(stu_id) on
delete cascade
)
```

- 插入数据

```sql
insert into Stu_Union(stu_id)values(1);
insert into Stu_Union(stu_id)values(2);

insert into Stu_Card(stu_id,card_id,grade)values(1,2017,3);
insert into Stu_Card(stu_id,card_id)values(2,2018);
```

	stu_id	card_id	grade
1	1	2017	3
2	2	2018	NULL

7. 为了演示多重级连删除，建立ICBC_Card表，令stu_card_id为参照Stu_Card表的外键，令bank_id为其主键，并插入数据。

- 建立ICBC_Card表

```sql
create table Stu_Card(
stu_id int,
card_id int primary key,
grade int,
constraint FK1 foreign key (stu_id) references Stu_Union(stu_id) on
delete cascade
)
create table ICBC_Card(
stu_card_id int,
bank_id int,
grade  int,
constraint PK primary key (bank_id),
constraint FK2 foreign key(stu_card_id) references Stu_Card(card_id)
on delete cascade
)
```

- 插入数据

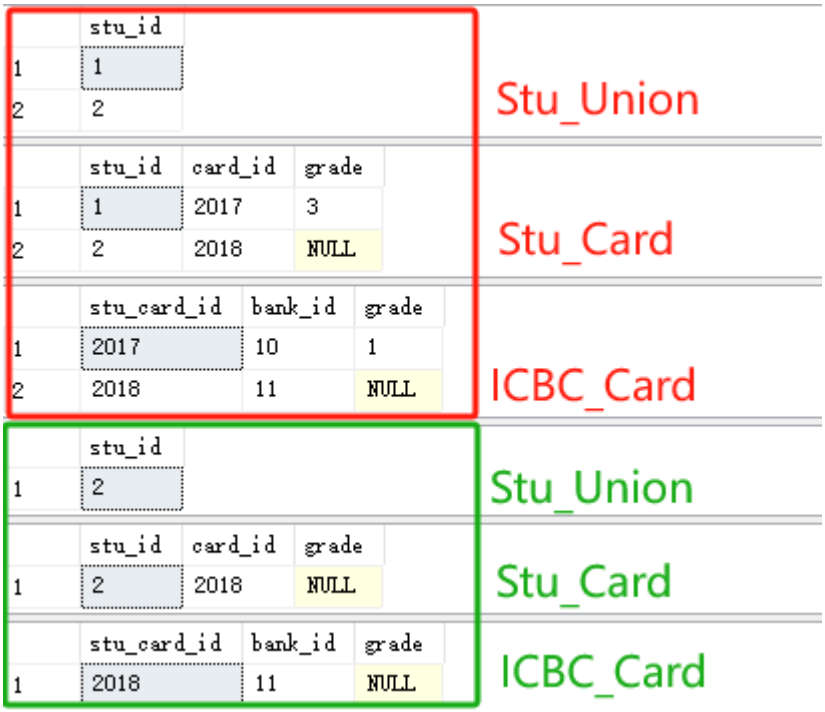
```sql
--insert into Stu_Card(stu_id,card_id,grade)values(1,2017,3);
--insert into Stu_Card(stu_id,card_id)values(2,2018);
insert into ICBC_Card(stu_card_id,bank_id,grade)values(2017,10,1);
insert into ICBC_Card(stu_card_id,bank_id)values(2018,11);
```

	stu_card_id	bank_id	grade
1	2017	10	1
2	2018	11	NULL

8. 通过删除Stu_Union表中的一条记录，演示三个表的多重级连删除。

```sql
delete Stu_Union
where stu_id=1;
```

结果如图所示，其中红框为删前，绿框为删



9. 演示事务中进行多重级连删除失败的处理。修改ICBC_Card表的外键属性，使其变为On delete No action, 演示事务中通过删除Stu_Union表中的一条记录，多重级连删除失败，整个事务回滚到事务的初始状态。

- 创建事务

```sql
begin tran
alter table ICBC_Card
drop constraint FK2;
alter table ICBC_Card
add constraint FK2 foreign key(stu_card_id) references
Stu_Card(card_id) on delete no action;
delete Stu_Union
where stu_id=1;
rollback tran
```

- 查看事务执行结果

```sql
exec sp_helpconstraint ICBC_Card;
select *
from ICBC_Card;
```

由于使用的是rollback tran(全部回滚)的规则，因此ICBC_Card的外键约束仍然是on delete cascade的

	Object Name
1	ICBC_Card

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_k
1	FOREIGN KEY	FK2	Cascade	No Action	Enabled	Is_For_Replication	stu_card_id
2							REFERENCES S
3	PRIMARY KEY (clustered)	PK	(n/a)	(n/a)	(n/a)	(n/a)	bank_id

	stu_card_id	bank_id	grade
1	2017	10	1
2	2018	11	NULL

10. 演示互参照问题及其解决方法。建立教师授课和课程指定教师听课关系的两张表，规定一个教师可以授多门课，但是只能去听一门课。为两张表建立相互之间的参照关系，暂时不考虑听课教师和授课教师是否相同（有余力的同学可以尝试限定听课与授课教师不相同）。

提示： 教师授课表以课程号为主键，教师号引用听课表的主键，听课表以教师号为主键，课程号引用授课表的主键。

```sql
create table  Course_Teacher_Teaching (
CourseId int NOT NULL,
TeacherId int NOT NULL,
primary key (CourseId)
);
create table Teacher_Listening (
TeacherId int NOT NULL,
CourseId int NOT NULL,
primary key (TeacherId)
);
alter table Course_Teacher_Teaching
add constraint FK3 foreign key(TeacherId) references
Teacher_Listening(TeacherId)

alter table Teacher_Listening
add constraint FK4 foreign key(CourseId) references
Course_Teacher_Teaching(CourseId)
```

实验1.3 触发器的应用

重要提示： 在做以下练习前，先删除sc对stu_union的外键引用

1. 在表sc中演示触发器的insert操作，当学生成绩低于60分时，自动改为60，并在事先创建的记录表中插入一条学生成绩低于60的记录。

   提示：另外创建一个表记录成绩低于60分的学生的真实记录。

- 创建一个记录表

```sql
CREATE TABLE Student_Record
  (
     sno   CHAR(5),
     cno   CHAR(1),
     grade INT
  );
```

- 创建触发器

SQL

```sql
CREATE TRIGGER Triger_SC_Insert
ON sc
after INSERT
AS
  BEGIN
      DECLARE @sno CHAR(5);
      DECLARE @cno CHAR(1);
      DECLARE @grade INT;
      DECLARE cur1 CURSOR FOR
        SELECT *
        FROM   inserted;

      BEGIN
          OPEN cur1

          FETCH next FROM cur1 INTO @sno, @cno, @grade;

          WHILE( @@FETCH_STATUS = 0 )
            BEGIN
                IF ( @grade < 60 )
                  BEGIN
                      --将sc更新
                      UPDATE sc
                      SET    grade = 60
                      WHERE  sno = @sno
                             AND cno = @cno;

                      --将原值移入记录表
                      INSERT INTO Student_Record
                                  (sno,
                                   cno,
                                   grade)
                      VALUES     (@sno,
                                  @cno,
                                  @grade);
                  END

                FETCH next FROM cur1 INTO @sno, @cno, @grade;
            END
```

```
        END

        CLOSE cur1

        DEALLOCATE cur1
    END
```

- 测试

```SQL
insert into sc(sno,cno,grade)values(95003,6,51)
 select *
 from sc
 select *
 from Student_Record
```

结果 消息

	Sno	Cno	Grade
1	95001	1	92
2	95001	2	85
3	95001	3	88
4	95002	2	90
5	95002	3	80
6	95001	4	98
7	95001	5	97
8	95003	6	60
9	95003	6	60
10	95003	7	60
11	95003	6	60

	sno	cno	grade
1	95003	6	51

2. 在表stu_union中创建行级触发器，触发事件是UPDATE。当更新表stu_union的Sid时，同时更新sc中的选课记录。

> 提示：这个触发器的作用实际上相当于具有CASCADE参数的外键引用。

3. 在表stu_union中删除一学生的学号(演示触发器的delete 操作)，使他在sc中关的信息同时被删除。

- 建立触发器

```sql
CREATE TRIGGER Trigger_StuUnion_Delete
ON Stu_Union
after DELETE
AS
  BEGIN
      DECLARE @Sid INT;

      SELECT @Sid = Sid
      FROM   deleted;

      DELETE FROM sc
      WHERE  sc.sno = @Sid;
  END
```

- 使用事务做测试

```sql
BEGIN TRAN

SELECT *
FROM   Stu_Union;
SELECT *
FROM   sc;

DELETE FROM Stu_Union
WHERE  Sid = 95001;

SELECT *
FROM   Stu_Union;
SELECT *
FROM   sc;
```

结果 消息

	Sid
1	95001
2	95002
3	95003

	Sno	Cno	Grade
1	95001	1	92
2	95001	2	85
3	95001	3	88
4	95002	2	90
5	95002	3	80

	Sid
1	95002
2	95003

	Sno	Cno	Grade
1	95002	2	90
2	95002	3	80
3	95003	6	60
4	95003	6	60
5	95003	7	60
6	95003	6	60

4. 演示触发器删除操作。

提示：SQL2005创建触发器的语法

```SQL
CREATE TRIGGER Trigger_StuUnion_Delete
ON Stu_Union
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM sc
    WHERE sno IN (SELECT deleted.Sid FROM deleted);
END;
```

实验1.4索引的建立和作用

实验目的

学会在SQL SERVER中建立索引
通过本实验体会覆盖索引的作用，在以后的实践中，能适时地使用覆盖索引来提高数据库的性能。
通过实验体会聚簇索引的优缺点，学会根据具体情况创建聚簇索引

实验内容

1. STUDENTS(sid,sname,email,grade)在sname上建立聚簇索引，grade上建立非聚簇索引，并分析所遇到的问题

- 建立STUDENTS表

```SQL
CREATE TABLE STUDENTS (
sno VARCHAR(10) PRIMARY KEY,
sname VARCHAR(50),
email VARCHAR(50),
grade INT
);
```

- 建立索引

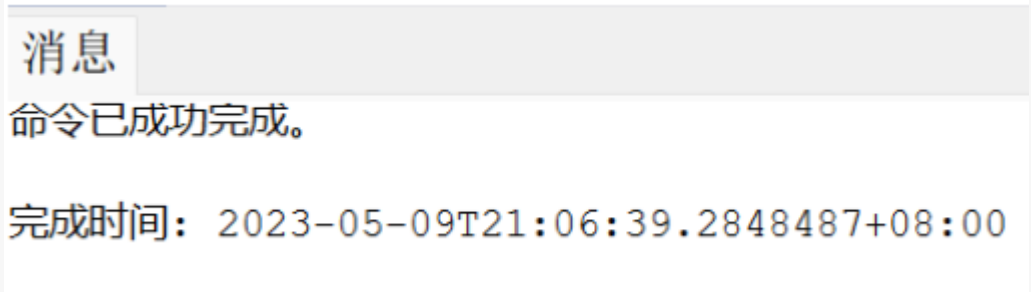
```SQL
create clustered index idx_clus_sname on STUDENT(sname);
create index idx_grade on STUDENT(grade);
```

建立聚簇索引出错，因为默认为主键建立聚簇索引，要先删除主键的聚簇索引才能建索引。
2. 数据库SCHOOL的选课表CHOICES有如下结构：

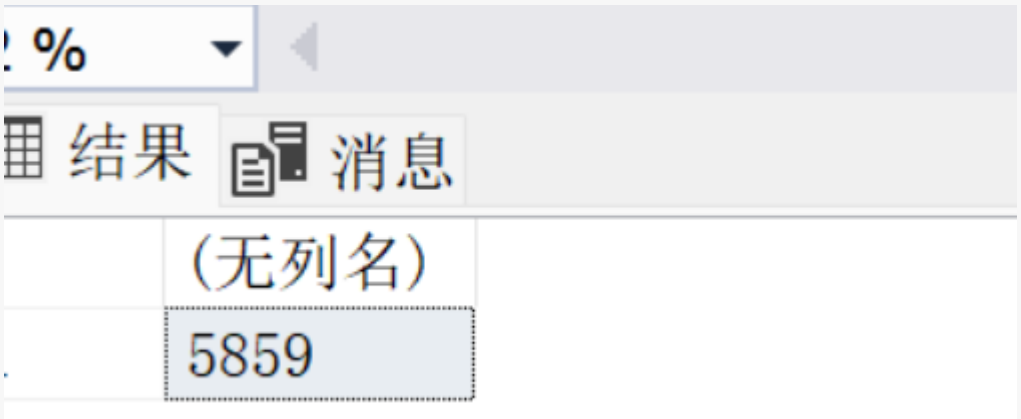
```
>CHOICES(no, sid, tid, cid, score)
>假设选课表集中用于查询分析，经常执行统计某课程修读的学生人数查询访问要求：
>1.  首先执行没有索引的实验（设数据库CHOICES表在cid列上没有索引）
>2. 然后做有索引的实验
>3. 对比试验结果，并进行分析
```

```sql
-- 创建 CHOICES 表
CREATE TABLE CHOICES
  (
     no    INT IDENTITY(1, 1) PRIMARY KEY,
     sid   VARCHAR(10),
     tid   VARCHAR(10),
     cid   VARCHAR(10),
     score INT
  );

-- 为 cid 字段添加索引
SELECT Count(*)
FROM   CHOICES
WHERE  cid = '101';

-- 为 cid 字段添加非聚簇索引
CREATE NONCLUSTERED INDEX idx_cid
  ON CHOICES(cid);

-- 重新运行查询
SELECT Count(*)
FROM   CHOICES
WHERE  cid = '101';
```

无索引:



有索引:



当对大型数据集进行查询时,使用索引可以显著提高查询性能。然而过多的索引可能会对数据库性能产生负面影响。因此，在创建索引时必须在性能和查询效率之间找到平衡点。

3. 以数据库SCHOOL中CHOICES表为例，设建表时考虑到以后经常有一个用sid查询此学生所有选课信息的查询，考虑到一般学生不止选一门课，且要询问这些记录的所有信息，故在sid上建立索引，使相同sid的记录存在一起，取数据页面时能一起取出来，减少数据页面的存取次数要求:

* 首先执行没有任何索引的情况
* 在sid上建有非聚簇索引的情况
* 在sid上建有聚簇索引的情况
* 对比实验结果，并进行分析

```sql
-- 为 sid 字段添加非聚簇索引

CREATE INDEX idx_sid ON CHOICES(sid);

-- 为 sid 字段添加聚簇索引

CREATE CLUSTERED INDEX idx_sid ON CHOICES(sid);

-- 运行查询

SELECT * FROM CHOICES WHERE sid = 'S01';
```

无索引：

消息

命令已成功完成。

完成时间：2023-05-09T21:15:43.3369330+08:00

有非聚簇索引：

1	500250857	800554358	262175339	10037	91
2	512254732	800554358	283495419	10015	62
3	515656273	800554358	244897158	10047	NULL
4	562223324	800554358	281785755	10043	79
5	588918248	800554358	224600699	10003	51

有聚簇索引：

1	515656273	800554358	244897158	10047	NULL
2	588918248	800554358	224600699	10003	51
3	500250857	800554358	262175339	10037	91
4	512254732	800554358	283495419	10015	62
5	562223324	800554358	281785755	10043	79

通过比对实验结果并进行分析，可以得出以下结论：

- 在没有索引的情况下，每次查询都需要扫描整个表，效率较低，随着数据量的增加，查询语句的执行时间会逐渐变长。因此，在表的大小较大时，没有索引会显著影响查询性能。
- 使用聚集索引进行查询时，它首先找到符合查询条件的第一条记录，然后根据索引中定义的列值顺序去查找下一条记录，直到找到满足所有查询条件的所有记录为止。由于聚簇索引是按照行的物理存储顺序排列的，因此查询时磁盘的读取次数会减少，从而提高查询速度。
- 使用非聚簇索引进行查询时，它首先根据索引列的值查找对应的记录，然后使用指针定位实际行数据，最后返回符合查询条件的结果。与聚簇索引不同，使用非聚簇索引进行查询时需要进行额外的磁盘读取，因此查询速度可能会受到一定的影响。