数据库期末复习笔记

目录

1 第一题：数据库的创建

1.1 题目

学生选课管理数据库经过一段时间的使用后，随着数据量的不断增大，引起数据库空间不足。

1. 现增加一个数据文件存储在 D:\，数据文件的逻辑名称为 Stu_Data2，物理文件名为 Stu_data2.ndf，初始大小为 10MB，最大尺寸为 2GB，增长速度为 10MB。
2. 现在增加一个事务日志文件，存储在 D:\中，日志文件的逻辑名称为 Stu_log2，物理文件名为 Stu_log2.ldf，初始大小为 10MB，最大尺寸为 500MB，增长速率为 15%。

1.2 解析

这里给出答案：

```
alter database 学生选课
add file
(
    name = Stu_Data2,
    filename = 'D:\Stu_data2.ndf',
    size = 10mb,
    maxsize = 2gb,
    filegrowth = 15%
)

alter database 学生选课
add log file
(
    name = Stu_log2,
    filename = 'D:\Stu_log2.ldf',
    size = 10mb,
    maxsize = 500mb,
    filegrowth = 15%
)
```

创建数据库模板：

```
/*创建数据库*/
create database [这里填数据库名称]
on
(
```

```
        name = [这里填文件的逻辑名称],
        filename = [这里填文件的存储位置以及文件名称],
        size = [这里填文件的初始大小],
        maxsize = [这里填限制文件的最大存储大小
                （如果要求无限大填unlimited）],
        filegrowth = [这里填文件大小的增长速度]
    )
    /*接下来为log日志文件的建立*/
    log on
    (
        /*格式与创建数据库部分一致*/
    )
```

需要注意的是，数据库文件名的格式为 `*.ndf`，而日志文件名的格式为 `*.ldf`，注意区别文件后缀。

添加文件的格式与创建数据库的格式是一样的：

```
    /*为数据库添加文件*/
    alter database [数据库名称]
    add [这里填文件类型，如果要添加数据库文件就填file，
        如果要添加日志文件就填log file]
    (
        /*这部分格式与创建数据库部分一致*/
    )
```

- `add file` 添加数据库文件
- `add log file` 添加日志文件

2 第二题：视图与索引

2.1 题目

视图与索引

1. 创建带加密选课视图 view_sc，列名依次显示为学号，姓名，年龄
2. 在学生表中的 sname 上创建非聚集索引 (Sname_ind)
3. 在给课程名创建唯一非聚集索引（Cname_ind）

2.2 解析

三个问题，分步进行解析。

2.2.1 第一问

题目 创建带加密选课视图 *view_sc*，列名依次显示为学号，姓名，年龄。

分析一下创建视图的模板：

```
create view [视图名称]([列名显示：name1,name2,...])
/*列名显示部分设置的列名一一对应select选出来的列*/
[with encryption] /*如果要求创建带加密的视图，需要添加这一句*/
as
/*下面为需要展示的内容的select语句*/
select [col1,col2,...]
...
```

根据创建视图的模板可以得到答案：

```
create view view_sc(学号,姓名,年龄)
with encryption
as
select sno,sname,sage
from student
```

2.2.2 第二问

题目 在学生表中的 *sname* 上创建非聚集索引 *(Sname_ind)*

分析一下创建索引的模板：

```
create [unique 唯一索引] [clustered 聚集索引| nonclustered 非聚集索引]
/*如果不填默认为非聚集索引 nonclustered*/
index [填索引名称]
on [表名 | 视图名](这个表或视图中的列名：[col1,col2,...])
```

根据模板可以得到答案：

```
create index Sname_ind
on Student(sname)
```

2.2.3 第三问

题目 在给课程名创建唯一非聚集索引（*Cname_ind*)

根据创建索引的模板2.2.2可以得到答案：

```
create unique nonclustered /*题目要求唯一，需要加unique*/
index Cname_ind
on Course(Cname)
```

2.3 知识点拓展

2.3.1 索引类型及特征

聚集索引（Clustered Index）

- **特征**：数据行的物理存储顺序与索引键值的逻辑顺序相同
- **限制**：每个表只能有一个聚集索引
- **作用**：提高范围查询和排序操作的效率
- **适用场景**：经常进行范围查询的列，如主键、日期列

非聚集索引（Non-clustered Index）

- **特征**：索引的逻辑顺序与数据行的物理存储顺序无关
- **限制**：每个表可以有多个非聚集索引（最多 999 个）
- **作用**：提高特定列的查询速度，但不改变数据的物理存储
- **适用场景**：经常用于 WHERE 子句、JOIN 条件的列

唯一索引（Unique Index）

- **特征**：确保索引键值的唯一性，不允许重复值
- **作用**：既提高查询性能，又保证数据完整性
- **适用场景**：需要保证唯一性的列，如身份证号、学号等

复合索引（Composite Index）

- **特征**：基于多个列创建的索引
- **作用**：提高多列组合查询的效率
- **注意**：遵循” 最左前缀” 原则，索引列的顺序很重要

2.3.2 索引使用原则

1. **选择性原则**：在选择性高（重复值少）的列上创建索引效果更好
2. **频率原则**：在经常用于查询条件的列上创建索引
3. **维护成本**：索引会增加 INSERT、UPDATE、DELETE 操作的开销
4. **存储空间**：索引需要额外的存储空间

> **小贴士**：索引是一把双刃剑，能够显著提高查询性能，但也会增加数据修改的开销和存储空间的占用。因此需要根据实际的查询需求和数据更新频率来合理设计索引策略。

2.3.3 索引的管理方式

1. 删除索引

```
drop index [表名.索引名 | 视图名.索引名]

/*一个简单的例子：删除第二问中的Sname_ind索引*/
drop index Student.Sname_ind
```

注意：表在创建时如果对某一列进行了主键约束会自动自动创建一个索引，这个索引是无法用 `drop index` 进行删除的。

2. 查看索引

可以使用 SSMS 软件进行查看，也可以用代码进行查看。这里介绍一下代码怎么查看。

```
use [数据库名]
go
sp_helpindex [表名]
go
```

2.3.4 重命名索引

利用 sp_rename 可以重命名索引

```
use [数据库名]
go
sp_rename '表名.原索引名', '新索引名'
go

/*举一个例子*/
use 学生选课
go
```

```
sp_rename 'student.Sname_index', 'index_Sname'
go
```

3 第三题：触发器

3.1 题目

1. 通过创建一个后触发型触发器 tr_sc_del，限制删除‘计算机系’学生的信息，并给出信息“不能删除计算机系学生选课信息!”（通过多表连接，不要用别名）
2. 通过创建一个前触发型触发器 tr_stu_del，限制删除有选课的学生信息。(注意：通过内连接实现，不取别名，临时表在前)
3. 创建一个用于防止用户删除学生选课数据库中任何数据表的触发器 tr_droptable。
4. 为了必须删除一个选课记录（学号 95001，课程号 001），请先抑制触发器 tr_sc_del，删除后，再恢复触发器。

3.2 解析

3.2.1 问题一

题目 通过创建一个后触发型触发器 *tr_sc_del*，限制删除‘计算机系’学生的信息，并给出信息“不能删除计算机系学生选课信息!”(通过多表连接，不要用别名)

先来介绍一下触发器。

1. DML 触发器

DML 触发器分为三类：AFTER 触发器、INSTEAD OF 触发器和 CLR 触发器。

- **AFTER 触发器**：在触发事件（INSERT、UPDATE、DELETE）完成后执行，是最常用的触发器类型
- **INSTEAD OF 触发器**：替代触发事件执行，主要用于视图上的数据修改操作
- **CLR 触发器**：使用.NET Framework 公共语言运行时创建的触发器，允许使用托管代码编写

2. DDL 触发器

DDL 触发器是在数据库结构发生变化时触发的，如创建、修改或删除表、索引、视图等操作。与 DML 触发器不同，DDL 触发器主要用于数据库管理和安全控制。

- **触发事件**：CREATE、ALTER、DROP 等数据定义语言操作
- **作用范围**：可以在数据库级别或服务器级别设置
- **主要用途**：防止误操作、记录结构变更日志、实施安全策略

来重点讲一下 DML 触发器

1. INSERTED 表和 DELETED 表

在 DML 触发器中，系统会自动创建两张临时表：INSERTED 表和 DELETED 表，用于存储触发器执行前后的数据变化。

- **INSERTED 表**：存储新插入或更新后的数据
- **DELETED 表**：存储被删除或更新前的数据

不同操作中的表状态：

1. **INSERT 操作**：
   - INSERTED 表：包含新插入的行数据
   - DELETED 表：为空
2. **DELETE 操作**：
   - INSERTED 表：为空
   - DELETED 表：包含被删除的行数据
3. **UPDATE 操作**：
   - INSERTED 表：包含更新后的新数据
   - DELETED 表：包含更新前的旧数据

总结过程： 表中被删除的数据会被转移到 DELETED 表，插入表中的新数据会被转移到 INSERTED 表。

使用示例：

```
-- 在触发器中访问这两张表
SELECT * FROM INSERTED;  -- 查看新数据
SELECT * FROM DELETED;   -- 查看旧数据

-- 常用于数据验证和日志记录
IF EXISTS (SELECT * FROM INSERTED WHERE salary < 0)
BEGIN
    RAISERROR('工资不能为负数', 16, 1)
    ROLLBACK TRANSACTION
END
```

2. 创建触发器

分析创建 DML 触发器的模板：

```
create trigger 触发器名
on 表名或视图名
{for | after | instead of} -- 这里是选择触发器的类型
                           -- 如果仅指定 for 关键字，则after为默认值
{insert | update | delete} -- 这里是指定哪种数据操作将激活触发器
[with encryption] -- 如果要求加密触发器需要加上这句
as
[if update (列名)] -- 判断指定的列（列名）是否进行了插入或更新操作
sql_statements -- 其他需要执行的sql语句
-- 注：单条语句时可省略BEGIN...END，多条语句时需要使用
```

3. 触发器执行顺序

当数据库操作发生时，触发器的执行遵循以下顺序：

1. **执行约束检查**：检查主键、外键、唯一约束等
2. **执行 INSTEAD OF 触发器**：如果存在，替代原始操作执行
3. **执行数据操作**：如果没有 INSTEAD OF 触发器，执行原始的 INSERT/UPDATE/DELETE 操作
4. **执行 AFTER 触发器**：在数据操作完成后执行
5. **提交或回滚事务**：根据触发器执行结果决定

重要提示：

- 如果触发器中执行了 `ROLLBACK TRANSACTION`，整个事务（包括原始操作）都会被回滚
- 多个触发器存在时，执行顺序可以通过 `sp_settriggerorder` 存储过程来设置
- 触发器中的错误会导致整个事务失败
- AFTER 触发器也叫做后触发型触发器
- INSTEAD OF 触发器也叫做前触发型触发器

现在可以得到这题的答案：

```
create trigger tr_sc_del
on SC
after update
as
if exists(
    select 1
```

```
    from delete
    inner join stu on delete.sno = stu.sno
    where stu.sdept = '计算机系'
    -- 查看被删除的信息中有没有计算机系的学生的信息
)
begin
    print '不能删除计算机系学生选课信息! '; -- 打印错误信息
    rollback transaction; -- 回滚事务
    return; -- 退出触发器
end
```

3.2.2 问题二

题目 通过创建一个前触发型触发器 *tr_stu_del*，限制删除有选课的学生信息。*(注意：通过内连接实现，不取别名，临时表在前)*

分析题目可以知道要使用前触发型触发器，在表被修改前进行操作。根据上一题的解析可以得到答案：

```
create trigger tr_stu_del
on stu
instead of delete
as
if exists(
    select 1
    from delete
    inner join sc on delete.sno = sc.sno
    -- 查找要删除的信息中是否存在有选课记录的学生
)
begin
    print '不能删除有选课记录的学生信息!';
    return;
end
else
-- INSTEAD型触发器会拦截删除操作，所以需要手动恢复
begin
    delete from stu
    where sno in (select sno from deleted);
end
```

重要提示：如何判断在哪张表上创建触发器

- **看操作对象**：题目中提到要限制对哪张表的操作，触发器就创建在那张表上
- **问题一**：限制删除” 计算机系学生的选课信息” → 操作的是 SC 表 → 在 SC 表上创建触发器
- **问题二**：限制删除” 有选课的学生信息” → 操作的是 Student 表 → 在 Student 表上创建触发器
- **一般规律**：
  - 如果要控制对表 A 的 INSERT/UPDATE/DELETE 操作，就在表 A 上创建触发器
  - 触发器会在对该表进行指定操作时自动执行
  - 通过 INSERTED/DELETED 表可以获取操作前后的数据进行判断

3.2.3 问题三

题目 *创建一个用于防止用户删除学生选课数据库中任何数据表的触发器 tr_droptable*

根据题目可以发现，触发器触发条件是删除数据表，由此可以判断需要创建 DDL 触发器。分析 DDL 触发器的创建模板：

```
create trigger 触发器名
on {all server | database}
[with encryption]
{for | after} {DDL事件} [,...n]
as
    sql_statements
```

每一个 DDL 事件都对应一个 Transact-SQL 语句。例如，DROP_TABLE 事件对应 DROP TABLE 语句，CREATE_TABLE 事件对应 CREATE TABLE 语句，ALTER_TABLE 事件对应 ALTER TABLE 语句等。

根据题目要求，我们需要防止删除数据表，所以使用 DROP_TABLE 事件。答案如下：

```
create trigger tr_droptable
on database
for drop_table
as
begin
    print '禁止删除数据库中的表! ';
    rollback transaction;
```

```
end
```

3.2.4 问题四

题目 为了必须删除一个选课记录（学号 *95001*，课程号 *001*），请先抑制触发器 *tr_sc_del*，删除后，再恢复触发器。

分析题目可知，要删除一个值需要先禁用触发器，删除后再启用这个触发器，考察触发器的禁用和启用。

1. 禁用触发器

禁用触发器的模板：

```
disable trigger {all | 触发器名 [,...n]} -- 哪个触发器
on {object_name | database | all server} -- 在哪个位置的触发器
```

2. 启用触发器

启用触发器的模板：

```
enable trigger {all | 触发器名 [,...n]} -- 哪个触发器
on {object_name | database | all server} -- 在哪个位置的触发器
```

根据题目要求，完整的操作步骤如下：

```
-- 第一步：禁用触发器
disable trigger tr_sc_del on sc;

-- 第二步：删除指定的选课记录
delete from sc
where sno = '95001' and cno = '001';

-- 第三步：启用触发器
enable trigger tr_sc_del on sc;
```

3.3 知识点拓展

3.3.1 查看触发器

可以通过以下几种方式查看触发器：

1. **查看数据库中的所有触发器**：

   ```
   SELECT * FROM sys.triggers;
   ```

2. **查看特定表上的触发器**：

```
SELECT * FROM sys.triggers
WHERE parent_id = OBJECT_ID('表名');
```

3. **查看触发器的详细信息**:

```
EXEC sp_helptext '触发器名';
```

3.3.2 删除触发器

删除触发器的语法很简单:

```
DROP TRIGGER 触发器名;
```

示例:

```
-- 删除DML触发器
DROP TRIGGER tr_sc_del;

-- 删除DDL触发器
DROP TRIGGER tr_droptable ON DATABASE;
```

注意: 删除触发器后，该触发器的所有功能将永久失效，请谨慎操作。

4 第四题: 数据表的创建与修改

4.1 题目

1. 创建下述 course 表

列名	数据类型	宽度	为空性
cno	char	6	
cname	char	20	
credit	tinyint		√

2. 在 course 表中添加一列 semester，整数型，非空
3. 将 cno 设置成主键 (主键名字为 pk_seme)
4. 在 sc 表中将 cno 设置为外键（外键名字为 fk_cno）

4.2 解析

4.2.1 问题一: 创建表

问题 *创建 course 表*

分析创建表的模板:

```
CREATE TABLE 表名
(
    列名1 数据类型(长度) [约束条件],
    列名2 数据类型(长度) [约束条件],
    ...
    列名n 数据类型(长度) [约束条件],
    [表级约束]
)
```

常用数据类型：

- **char(n)**：固定长度字符串
- **varchar(n)**：可变长度字符串
- **int**：整数型
- **tinyint**：小整数型（0-255）
- **decimal(p,s)**：定点数
- **datetime**：日期时间型

约束条件：

- **NOT NULL**：非空约束
- **NULL**：允许为空（默认）
- **PRIMARY KEY**：主键约束
- **UNIQUE**：唯一约束
- **DEFAULT 值**：默认值约束

根据题目要求，创建 course 表的答案如下：

```
CREATE TABLE course
(
    cno CHAR(6) NOT NULL,
    cname CHAR(20) NOT NULL,
    credit TINYINT NULL
)
```

4.2.2 问题二：添加列

问题 *在 course 表中添加一列 semester，整数型，非空*

分析添加列的模板：

```
ALTER TABLE 表名
ADD 列名 数据类型(长度) [约束条件]
```

说明：

- **ALTER TABLE**：修改表结构的关键字
- **ADD**：添加列的操作
- 可以同时添加多列，用逗号分隔
- 添加的列默认为 NULL，如需非空需显式指定 NOT NULL

根据题目要求，添加 semester 列的答案如下：

```
ALTER TABLE course
ADD semester INT NOT NULL
```

4.2.3 问题三：设置主键

问题 *将 cno 设置成主键 (主键名字为 pk_seme)*

分析添加主键约束的模板：

```
ALTER TABLE 表名
ADD CONSTRAINT 约束名 PRIMARY KEY (列名)
```

说明：

- **ADD CONSTRAINT**：添加约束的关键字
- **约束名**：自定义的约束名称，便于管理
- **PRIMARY KEY**：主键约束类型
- 主键列必须为 NOT NULL，且值唯一
- 一个表只能有一个主键

根据题目要求，设置 cno 为主键的答案如下：

```
ALTER TABLE course
ADD CONSTRAINT pk_seme PRIMARY KEY (cno)
```

拓展说明

主键的作用：

- **唯一性标识**：确保表中每一行都有唯一的标识符，不允许重复值
- **非空约束**：主键列不能包含 NULL 值，保证数据完整性
- **建立索引**：系统自动为主键创建唯一聚集索引，提高查询性能
- **外键引用**：作为其他表外键的参照目标，建立表间关系
- **数据完整性**：防止插入重复或无效的数据，维护数据质量
- **优化存储**：聚集索引按主键顺序物理存储数据，提高 I/O 效率

4.2.4 问题四：设置外键

问题 *在 sc 表中将 cno 设置为外键（外键名字为 fk_cno）*

外键可以在创建表时定义，也可以在表创建后添加。

方式一：创建表时定义外键

```
CREATE TABLE 表名
(
    列名1 数据类型 约束条件,
    列名2 数据类型 约束条件,
    ...
    CONSTRAINT 外键名 FOREIGN KEY (列名)
        REFERENCES 参照表名(参照列名)
)
```

方式二：表创建后添加外键

```
ALTER TABLE 表名
ADD CONSTRAINT 外键名
    FOREIGN KEY (列名) REFERENCES 参照表名(参照列名)
```

说明：

- **FOREIGN KEY**：定义外键约束
- **REFERENCES**：指定参照的主表和主键
- 外键列的数据类型必须与参照列相同
- 参照列必须是主键或唯一键

根据题目要求，在 sc 表中设置 cno 为外键的答案如下：

```
ALTER TABLE sc
ADD CONSTRAINT fk_cno
    FOREIGN KEY (cno) REFERENCES course(cno)
```

外键的作用：

- **维护参照完整性**：确保子表中的外键值必须在父表的主键中存在
- **防止数据不一致**：避免插入无效的关联数据
- **级联操作**：可设置级联删除或更新，保持数据同步
- **建立表间关系**：明确表与表之间的逻辑关系
- **约束数据操作**：限制对父表主键的删除和修改操作
- **提高数据可靠性**：防止孤立记录的产生

4.3 知识点拓展

4.3.1 唯一约束

唯一约束确保列中的值是唯一的，但允许 NULL 值。

```
-- 创建表时定义
CREATE TABLE 表名
(
    列名 数据类型 UNIQUE,
    ...
)

-- 后续添加
ALTER TABLE 表名
ADD CONSTRAINT 约束名 UNIQUE (列名)
```

示例：

```
ALTER TABLE student
ADD CONSTRAINT uk_student_email UNIQUE (email);
```

4.3.2 检查约束

检查约束用于限制列中允许的值范围。

```
-- 创建表时定义
CREATE TABLE 表名
(
    列名 数据类型 CHECK (条件表达式),
    ...
)

-- 后续添加
ALTER TABLE 表名
ADD CONSTRAINT 约束名 CHECK (条件表达式)
```

示例:

```
ALTER TABLE student
ADD CONSTRAINT ck_student_age CHECK (age ⩾ 0 AND age ⩽ 150);
```

4.3.3 默认值约束

默认值约束为列提供默认值，在插入数据时如果未指定值则使用默认值。

```
-- 创建表时定义
CREATE TABLE 表名
(
    列名 数据类型 DEFAULT 默认值,
    ...
)

-- 后续添加
ALTER TABLE 表名
ADD CONSTRAINT 约束名 DEFAULT 默认值 FOR 列名
```

示例:

```
ALTER TABLE student
ADD CONSTRAINT df_student_status DEFAULT '在读' FOR status;
```

4.3.4 约束禁用和启用

可以临时禁用或启用约束，常用于数据导入或维护操作。

```
-- 禁用约束
ALTER TABLE 表名 NOCHECK CONSTRAINT 约束名;
```

```
-- 启用约束
ALTER TABLE 表名 CHECK CONSTRAINT 约束名;

-- 删除约束
ALTER TABLE 表名 DROP CONSTRAINT 约束名;
```

示例:

```
-- 禁用外键约束
ALTER TABLE sc NOCHECK CONSTRAINT fk_cno;

-- 启用外键约束
ALTER TABLE sc CHECK CONSTRAINT fk_cno;

-- 删除约束
ALTER TABLE sc DROP CONSTRAINT fk_cno;
```

约束管理要点:

- 约束名称应具有描述性，便于识别和管理
- 禁用约束后记得及时启用，避免数据完整性问题
- 删除约束前要谨慎考虑，确保不会影响数据完整性
- 可以通过系统视图查询表中的所有约束信息

5 第五题：游标的创建与使用

5.1 题目

将 student 表中的同学按照姓名升序后的前 7 位同学的相关信息打印在消息窗格中，格式为：姓名 +‘的年龄为’+ 年龄。游标名为 stu_cur，为了方便，将姓名、年龄存储在局部变量为 @stu_name, @stu_age 上。局部变量声明在打开游标之前。

5.2 解析

这道题需要创建并打开游标，所以我先介绍一下游标的创建。

5.2.1 创建游标

创建游标的简单模板

```
declare 游标名 cursor
for 数据库查询语句
```

说明： 这不是完整的游标创建模板，完整版考试不要求。

简单示例

```
declare Mycur cursor
for select Sno,Sname
from student
where Ssex='男'
```

5.2.2 打开游标

打开游标的模板

```
open [local | global] 游标名 | 游标变量名
```

简单示例

```
open Mycur
```

5.2.3 读取游标

游标的读取使用 FETCH 语句，其过程分两步：

1. 将游标当前指向的记录保存到一个局部变量中
2. 游标自动移动到下一条记录

当记录读入局部变量后，就可以根据需要进行处理。

读取游标的模板

```
fetch [[next | prior | first | last |
absolute{n | @nvar | relative{n | @nvar}}]
from ] 游标名 [into @局部变量名 [,...n]]
```

参数含义如1。

5.2.4 题目解答

根据题目要求，需要创建游标并获取前 7 位学生信息。完整代码如下：

```
-- 声明局部变量（在打开游标之前）
declare @stu_name varchar(20);
declare @stu_age int;
declare @count int = 0;
```

参数	含义
NEXT	移动到下一条记录（默认选项）
PRIOR	移动到上一条记录
FIRST	移动到第一条记录
LAST	移动到最后一条记录
ABSOLUTE n	移动到第 n 条记录（正数从头开始，负数从尾开始）
RELATIVE n	相对当前位置移动 n 条记录（正数向前，负数向后）
INTO @ 变量名	将获取的数据存储到指定的局部变量中

表 1: FETCH 语句参数含义

```
-- 声明游标
declare stu_cur cursor
for select sname, sage
    from student
    order by sname asc;

-- 打开游标
open stu_cur;

-- 读取第一条记录
fetch next from stu_cur into @stu_name, @stu_age;

-- 循环处理前7条记录
while @@fetch_status = 0 and @count < 7
begin
    -- 打印信息到消息窗格
    print @stu_name + '的年龄为' + cast(@stu_age as varchar(10));

    -- 计数器加1
    set @count = @count + 1;

    -- 读取下一条记录
    fetch next from stu_cur into @stu_name, @stu_age;
end

-- 关闭游标
close stu_cur;
```

```
-- 释放游标
deallocate stu_cur;
```

代码说明：

- **@@FETCH_STATUS**：系统变量，表示上一次 FETCH 操作的状态（0 表示成功）
- **CAST 函数**：将整数类型的年龄转换为字符串，便于拼接
- **计数器 @count**：确保只处理前 7 条记录
- **CLOSE**：关闭游标，释放资源
- **DEALLOCATE**：释放游标内存

6 第六题：存储过程

6.1 题目

1. 创建存储过程（p_stu），实现给定学号（局部变量名为 @stu_sno，并取默认值为 95001），列出年龄大于该同学的学生信息，姓名和年龄（列名不变）。存储过程中的查询语句通过子查询实现。
2. 执行上述存储过程，取 95004 实验。
3. 创建存储过程（p_stu2），实现给定学号（局部变量名为 @stu_sno，并取默认值为 95001），列出与该学生属于同一系的其他学生姓名和年龄（原样显示）。存储过程中的查询语句通过自身链接（别名采用 s1,s2，且通过 s2 表返回）实现。
4. 通过 95002 验证。
5. 创建存储过程 p_count_cs, 根据输入学生学号（@stu_sno），返回该学生选了多少门课。返回值为整数型，且取名为 @c_ss。
6. 检查学号为 95001 的情况验证准确性。输出变量取名为 @pp_cnos，打印内容为仅 @pp_cnos。考虑如果是指定课程号，返回有多少学生选了该课程，应该怎么写和验证。

6.2 解析

在解析题目之前，我先讲一下存储过程的创建和执行。

6.2.1 存储过程的创建

带参数的存储过程创建简单模板

```
create procedure 存储过程名
[{@参数名称 参数数据类型} [ = 参数的默认值]
[output] ] -- 参数后面带output的为输出参数
[,...n]
as
sql_statement
```

简单示例

```
-- 带输入参数的存储过程
create procedure p_student
@Sno char(5)
as
select Sname, Sdept
from student where Sno=@Sno

-- 带输出参数的存储过程
create procedure p_sum
@var1 int, @var2 int,
@var3 int output -- 输出参数
as
set @var3 = @var1 * @var2
```

不带参数的存储过程创建简单模板

```
create procedure 存储过程名
as
sql_statement
```

说明： 完整版模板考试不要求，变量前必须加 @。

简单示例

```
create procedure p_course
as
select * from course
```

6.2.2 存储过程的执行

不带参数的存储过程执行模板

```
exec 存储过程名
```

简单示例

```
exec p_course
```

带输入参数的存储过程

```
exec 存储过程名
[@参数名 = 参数值][default]
[,...n]
```

简单示例

```
exec p_grade2 @dept='计算机系'
```

带输出参数的存储过程

```
exec 存储过程名
[[@参数名=]{参数值 | @变量[output] | [默认值]}][,...n]
```

简单示例

```
declare @res int
exec p_sum @var1=3,@var2=8,@res=@res output

-- 或者使用参数顺序调用
exec p_sum 3,8,@res output
```

6.2.3 问题一

题目 创建存储过程（*p_stu*），实现给定学号（局部变量名为 *@stu_sno*，并取默认值为 *95001*），列出年龄大于该同学的学生信息，姓名和年龄（列名不变）。存储过程中的查询语句通过子查询实现。

根据存储过程的知识可得到答案。

```
create procedure p_stu
@stu_sno char(5) = '95001'
as
select Sname, Sage
from student
where Sage > (
    select Sage from student
    where Sno=@stu_sno
```

```
)
```

6.2.4 问题二

题目 执行上述存储过程，取 *95004* 实验。

传入 95004 作为参数执问题一的存储过程，得到答案。

```
exec p_stu '95004' -- 注意char类型需要加引号
```

6.2.5 问题三

题目 创建存储过程（*p_stu2*），实现给定学号（局部变量名为 *@stu_sno*，并取默认值为 *95001*），列出与该学生属于同一系的其他学生姓名和年龄（原样显示）。存储过程中的查询语句通过自身链接（别名采用 *s1,s2*，且通过 *s2* 表返回）实现。

难点在于如何查询。得到答案。

```
create procedure p_stu2
@stu_sno char(5) = '95001'
as
select s2.Sname, s2.Sage
from student s2 join student s1
on s2.Sdept=s1.Sdept
where s1.Sno=@stu_sno and s2.Sno ≠ @stu_sno
```

注意：本题要查询的是“其他学生”，不包括输入的学生，注意排除。

6.2.6 问题四

题目 通过 *95002* 验证。

执行时传入参数即可得到答案。

```
exec p_stu2 '95002'
```

6.2.7 问题五

题目 创建存储过程 *p_count_cs*，根据输入学生学号（*@stu_sno*），返回该学生选了多少门课。返回值为整数型，且取名为 *@c_ss*。

这道题的考点在于存储过程的输出参数。得到答案。

```
create procedure p_count_cs
@stu_sno char(5), @c_ss int output
as
select @c_ss=count(*)
```

```
from SC
where Sno=@stu_sno
```

6.2.8 问题六

题目 检查学号为 *95001* 的情况验证准确性。输出变量取名为 *@pp_cnos*，打印内容为仅 *@pp_cnos*。考虑如果是指定课程号，返回有多少学生选了该课程，应该怎么写和验证。

这题的考点是带参数的存储过程的执行。得到答案。

```
-- 定义变量
declare @pp_cnos int

-- 执行存储过程
exec p_count_cs '95001', @pp_cnos output

-- 打印结果
print @pp_cnos
```

注意：输出参数的后面要加 output，不要忘记。

6.3 知识点拓展

6.3.1 查看存储过程

可以通过以下几种方式查看存储过程：

1. **查看所有存储过程**：

   ```
   SELECT * FROM sys.procedures;
   ```

2. **查看存储过程定义**：

   ```
   EXEC sp_helptext '存储过程名';
   ```

3. **查看存储过程参数**：

   ```
   EXEC sp_help '存储过程名';
   ```

示例：

```
EXEC sp_helptext 'p_stu';
```

6.3.2 删除用户存储过程

删除存储过程的语法：

```
DROP PROCEDURE 存储过程名;
```

示例：

```
-- 删除单个存储过程
DROP PROCEDURE p_stu;

-- 删除多个存储过程
DROP PROCEDURE p_stu, p_stu2, p_count_cs;
```

注意：删除存储过程是不可逆操作，请谨慎执行。

6.3.3 修改存储过程

修改存储过程的语法：

```
ALTER PROCEDURE 存储过程名
[参数列表]
AS
BEGIN
    -- 修改后的SQL语句
END
```

示例：

```
ALTER PROCEDURE p_stu
@stu_sno CHAR(5) = '95001'
AS
BEGIN
   SELECT Sname, Sage, Sdept -- 增加系别信息
   FROM student
   WHERE Sage > (
      SELECT Sage FROM student
      WHERE Sno = @stu_sno
   )
END
```

存储过程管理要点：

- 修改存储过程时使用 ALTER PROCEDURE，而不是重新 CREATE
- 存储过程名在数据库中必须唯一

- 定期检查和优化存储过程的性能
- 为存储过程添加适当的注释，便于维护
- 测试存储过程的各种参数组合，确保逻辑正确

7 第七题：安全管理

7.1 题目

1. 请建立 SQL Server 登录名 sql_user1，并映射至同名用户名。其中登录密码为'nulibeikao'。
2. 对象权限，授予 sql_user1 用户在 student 表中的插入，更新，查询权利。
3. 语句权限，允许 sql_user1 用户在数据库上创建视图、存储过程的权限。
4. 语句权限，拒绝 sql_user1 用户在数据库上创建表的权限
5. 对象权限，拒绝 sql_user1 用户在 sc 表上删除数据的权利

7.2 解析

这一章的内容基本都可以通过 SSMS 软件进行图形化操作，详见书本第九章 P202。这里主要介绍如何通过指令操作。

7.2.1 问题一

题目 *请建立 SQL Server 登录名 sql_user1，并映射至同名用户名。其中登录密码为'nulibeikao'。*

分析创建和映射数据库用户的简单模板。

```
-- 创建数据库用户
create user 用户名
[with Password=密码]
[DEFAULT_DATABASE=默认数据库]

-- 创建数据库登录名
create login 登录名
[with Password=密码]

-- 映射登录名到数据库用户
create user 用户名
for login 登录名
```

用户名、登录名、映射关系简介：

- **登录名（Login）**：是 SQL Server 服务器级别的安全主体，用于连接到 SQL Server 实例。登录名存储在 master 数据库中，是进入 SQL Server 的”钥匙”。
- **用户名（User）**：是数据库级别的安全主体，存在于特定的数据库中。用户名决定了在该数据库内能执行哪些操作。
- **映射关系**：登录名和用户名之间需要建立映射关系，这样登录名才能访问特定的数据库。一个登录名可以映射到多个数据库中的不同用户名。一个登录名在一个数据库中只能有有唯一一个数据库用户与之对应。
- **权限层次**：
  - 服务器级权限 → 登录名
  - 数据库级权限 → 用户名

简单来说：**登录名负责”进门”，用户名负责”干活”**。先用登录名连接到 SQL Server，再通过映射的用户名在具体数据库中执行操作。

根据模板可以得到答案。

```
-- 创建登录名
create login sql_user1
with Password='nulibeikao'

-- 映射登录名到用户
create user sql_user1
for login sql_user1
```

注意：如果题目中要求建立映射关系，但前面没有建立登录名，要先建立登录名再建立映射关系。

7.2.2 问题二

题目 对象权限，授予 *sql_user1* 用户在 *student* 表中的插入，更新，查询权利。

这道题涉及到数据库用户权限管理，我先介绍一下数据库的权限类别，如表2。

权限类别	描述
SELECT	查询权限，允许用户查看表或视图中的数据
INSERT	插入权限，允许用户向表中添加新的数据行
UPDATE	更新权限，允许用户修改表中现有的数据
DELETE	删除权限，允许用户删除表中的数据行
REFERENCES	引用权限，允许用户创建引用该表的外键约束
ALTER	修改权限，允许用户修改表结构（添加/删除列等）
INDEX	索引权限，允许用户在表上创建或删除索引
EXECUTE	执行权限，允许用户执行存储过程或函数
CREATE TABLE	创建表权限，允许用户在数据库中创建新表
CREATE VIEW	创建视图权限，允许用户在数据库中创建视图
CREATE PROCEDURE	创建存储过程权限，允许用户创建存储过程

表 2: SQL Server 权限类别说明

下面给出授予用户权限的模板并给出几个例子。

```
-- 模板
grant 权限类别[,...n] [on 表名[,...n]] to 用户名

-- 授予用户创建数据库的权限
grant create database to Qtcyy

-- 授予用户对student表进行插入、更新数据的权限
grant insert, update on student to Qtcyy
```

根据模板可以得到答案。

```
grant insert, update, select on student to sql_user1
```

老师提醒： 注意题目的各个权限的顺序，考试系统内标准比较严，请按照顺序给。

7.2.3 问题三

题目 语句权限，允许 *sql_user1* 用户在数据库上创建视图、存储过程的权限。

这题考查的是创建视图、存储过程的权限，根据模板可以得到答案。

```
grant create view, create procedure to sql_user1
```

7.2.4 问题四

题目 语句权限，拒绝 *sql_user1* 用户在数据库上创建表的权限

这道题涉及到拒绝用户权限，需要使用 DENY 语句。下面给出拒绝用户权限的模板。

```
-- 拒绝权限模板
deny 权限类别[,...n] [on 表名[,...n]] to 用户名

-- 拒绝用户创建表的权限
deny create table to Qtcyy

-- 拒绝用户对student表进行删除数据的权限
deny delete on student to Qtcyy
```

根据模板可以得到答案。

```
deny create table to sql_user1
```

权限控制语句对比：

- **GRANT**：授予权限，允许用户执行某些操作
- **DENY**：拒绝权限，明确禁止用户执行某些操作
- **REVOKE**：撤销权限，取消之前授予或拒绝的权限

注意： DENY 的优先级高于 GRANT，即使用户通过其他角色获得了权限，DENY 仍然有效。

7.2.5 问题五

题目 *对象权限，拒绝 sql_user1 用户在 sc 表上删除数据的权利*

根据模板可以得到答案。

```
deny delete on sc to sql_user1
```

7.3 知识点拓展

7.3.1 身份验证模式

SQL Server 支持两种身份验证模式：

- **Windows 身份验证模式**：
  - 使用 Windows 用户账户或 Windows 组账户进行身份验证
  - 更安全，因为利用了 Windows 的安全机制
  - 支持密码策略、账户锁定等 Windows 安全功能
  - 适用于企业内部网络环境
- **混合模式（SQL Server 和 Windows 身份验证）**：

- 同时支持 Windows 身份验证和 SQL Server 身份验证
- SQL Server 身份验证使用存储在 SQL Server 中的登录名和密码
- 适用于需要支持非 Windows 客户端的环境
- 默认启用 sa 账户（系统管理员账户）

7.3.2 服务器角色

SQL Server 提供了预定义的服务器级固定角色，用于管理服务器级权限：

服务器角色	权限描述
sysadmin	系统管理员，拥有服务器的完全控制权限
serveradmin	服务器管理员，可以更改服务器范围的配置选项和关闭服务器
securityadmin	安全管理员，管理登录名和权限，可以重置密码
processadmin	进程管理员，可以终止在 SQL Server 实例中运行的进程
setupadmin	安装管理员，可以添加和删除链接服务器
bulkadmin	批量管理员，可以执行 BULK INSERT 语句
diskadmin	磁盘管理员，管理磁盘文件
dbcreator	数据库创建者，可以创建、修改、删除和还原任何数据库
public	公共角色，每个 SQL Server 登录名都属于此角色

表 3: SQL Server 服务器角色

添加用户到服务器角色的语法：

```
-- 将登录名添加到服务器角色
ALTER SERVER ROLE 角色名 ADD MEMBER 登录名

-- 示例：将sql_user1添加到dbcreator角色
ALTER SERVER ROLE dbcreator ADD MEMBER sql_user1
```

7.3.3 数据库角色

数据库角色用于管理数据库级别的权限，包括固定数据库角色和用户定义角色：

数据库角色	权限描述
db_owner	数据库所有者，拥有数据库的完全控制权限
db_accessadmin	访问管理员，可以添加或删除数据库用户
db_securityadmin	安全管理员，可以管理角色成员身份和权限
db_ddladmin	DDL 管理员,可以运行任何 DDL 命令(CREATE、ALTER、DROP)
db_backupoperator	备份操作员，可以备份数据库
db_datareader	数据读取者，可以从所有用户表中读取数据
db_datawriter	数据写入者，可以在所有用户表中添加、更新或删除数据
db_denydatareader	拒绝数据读取者，不能读取数据库中的任何数据
db_denydatawriter	拒绝数据写入者，不能修改数据库中的任何数据
public	公共角色，每个数据库用户都属于此角色

表 4: SQL Server 数据库角色

数据库角色管理语法：

```
-- 将用户添加到数据库角色
ALTER ROLE 角色名 ADD MEMBER 用户名

-- 从数据库角色中删除用户
ALTER ROLE 角色名 DROP MEMBER 用户名

-- 创建用户定义的数据库角色
CREATE ROLE 角色名

-- 示例：将sql_user1添加到db_datareader角色
ALTER ROLE db_datareader ADD MEMBER sql_user1
```

权限继承关系：

- 用户可以同时属于多个角色
- 用户获得所属所有角色的权限（权限的并集）
- DENY 权限始终优先于 GRANT 权限
- 角色权限与直接授予用户的权限相互补充