现有学生表（学号，姓名，性别，年龄，入学年份，籍贯，系号，班长学号），学号是主码，系号和班长学号是外部码，手机号码必须唯一，学生的年龄不得小于10岁和大于50岁，性别必须是'男'或者'女'。

系表（系号，系名，系主任），其中系号是主码，系名不能有重复的。

选课表（学号，课程号，成绩），（学号，课程号）是主码，学号和课程号是外部码，成绩不能小于0分和大于100分

课程表（课程号，课程名，先修课，学分），课程号是主码，课程名必须唯一，学分必须大于0小于5。

试完成以下工作：

1. 创建上述表，并定义相应的完整性约束。

**CREATE TABLE** `系表` (  
 **`系号` CHAR**(3) **PRIMARY KEY** ,  
 **`系名` CHAR**(100) **UNIQUE** ,  
 **`系主任` CHAR**(40)  
);  
  
**CREATE TABLE** `学生表` (  
 **`学号` CHAR**(10) **PRIMARY KEY** ,  
 **`姓名` CHAR**(40),  
 **`性别` CHAR**(5) **CHECK** (**`性别` IN**(**'男'**,**'女'**)),  
 **`年龄` INT CHECK** (**`年龄`**>=10 **AND `年龄`**<=50),  
 **`入学年份` CHAR**(2),  
 **`籍贯` CHAR**(40),  
 **`系号` CHAR**(3),  
 **`班长学号` CHAR**(10),  
 **FOREIGN KEY** (**`系号`**) **REFERENCES** `系表`(**`系号`**),  
 **FOREIGN KEY** (**`班长学号`**) **REFERENCES** `学生表`(**`学号`**)  
);  
  
**CREATE TABLE** `课程表`(  
 **`课程号` CHAR**(10) **PRIMARY KEY** ,  
 **`课程名` CHAR**(40) **UNIQUE** ,  
 **`先修课` CHAR**(10),  
 **`学分` INT CHECK** (**`学分`**>0 **AND `学分`**<5),  
 **FOREIGN KEY** (**`先修课`**) **REFERENCES** `课程表`(**`课程号`**)  
);  
  
**CREATE TABLE** `选课表` (  
 **`学号` CHAR**(10),  
 **`课程号` CHAR**(10),  
 **`成绩` DOUBLE CHECK** (**`成绩`**>=0 **AND `成绩`**<=100),  
 **PRIMARY KEY** (**`学号`**,**`课程号`**),  
 **FOREIGN KEY** (**`学号`**) **REFERENCES** `学生表`(**`学号`**),  
 **FOREIGN KEY** (**`课程号`**) **REFERENCES** `课程表`(**`课程号`**)  
);  
  
**CREATE TABLE** `学分计算表` (  
 **`最低成绩` INT**,  
 **`最高成绩` INT**,  
 **`计算比率` DOUBLE**);

1. 将提供的上述各表的数据导入SQL Server。如果导入过程存在错误，请说明可能是什么原因。

没有错误

1. 在学生表中插入学号为26，姓名为'李四'，性别为'女'，年龄为20，入学年份为2008，籍贯为'广东'，手机号码为10010001000，班长学号为10的一条记录。

**INSERT INTO** 学生表(学号, 姓名, 性别, 年龄, 入学年份, 籍贯, 班长学号, 手机号)  
 **VALUES** (26, **'李四'**, **'女'**, 20, 2008, **'广东'**, 10, 10010001000)

1. 删除上述记录。

**DELETE FROM** 学生表  
**WHERE 学号**=26

1. 将学生表中的姓名字段的长度改为6个汉字。

**ALTER TABLE** 学生表 **MODIFY 姓名 CHAR**(18)

1. 为学生表增加一个字段电子邮件，20个字符。

**ALTER TABLE** 学生表 **ADD COLUMN 电子邮件 CHAR**(20);

1. 对课程表的学分字段上的完整性约束进行修改，使其在0到6之间取值。

**ALTER TABLE** 课程表 **ADD CONSTRAINT** CK\_score **CHECK** (0 <= **学分 AND 学分** <= 6);

1. 为学生表在学号列上创建cluster索引。

**ALTER TABLE** 学生表 **ADD INDEX** cluster(**'学号'**);

1. 创建一个视图，计算每门课的最高分。

**CREATE VIEW** 每门课的最高分 **AS SELECT** 课程表.**课程号**, *MAX*(**成绩**) **FROM** 课程表, 选课表 **WHERE** 课程表.**课程号**=选课表.**课程号 GROUP BY** 课程表.**课程号**;

1. 查找每个学生的学号、姓名、总成绩和平均分。

**SELECT** 学生表.**学号**, **姓名**, *SUM*(**成绩**) **FROM** 学生表, 选课表 **WHERE** 学生表.**学号**=选课表.**学号 GROUP BY** 学生表.**学号**;

1. 将6系所有学生的年龄，改为7系学生的平均年龄。

**UPDATE** 学生表  
**INNER JOIN** (  
 **SELECT** *AVG*(**年龄**) 7系平均年龄 **FROM** 学生表 **WHERE 系号** = **'07'**) **AS** B  
**SET 年龄** = B.7系平均年龄 **WHERE 系号** = **'06'**;

1. 将’曹洪’同学操作系统课程的成绩改为62分。

**UPDATE** 选课表  
**SET 成绩**=67 **WHERE** 学生表.**姓名**=**'曹洪'**

1. 查找所有学生的姓名、入学年份和籍贯。

**SELECT 姓名**, **入学年份**, **籍贯 FROM** 学生表;

1. 列出籍贯为'山东'的同学的所有属性。

**SELECT** *\** **FROM** 学生表 **WHERE 籍贯**=**'山东'**

1. 查找年龄最小的学生的学号和姓名。

**SELECT 学号**, **姓名 FROM** 学生表 **WHERE 年龄**=(**SELECT** *MIN*(**年龄**) **FROM** 学生表);

1. 查找选修了'数据库'的学生的学号。

**SELECT DISTINCT** 学生表.**学号 FROM** 学生表 **INNER JOIN** 课程表 **INNER JOIN** 选课表 **WHERE 课程名**=**'数据库'**;

1. 查找选修了'编译技术'的女学生的学号和姓名。

**SELECT DISTINCT** 学生表.**学号**, 学生表.**姓名 FROM** 学生表 **INNER JOIN** 选课表 **INNER JOIN** 课程表 **WHERE 课程名**=**'编译技术' AND 性别**=**'女'**;

1. 查找'典韦'同学的班长所选修的课程的课程号。

**SELECT DISTINCT** 课程表.**课程号 FROM** 学生表 **INNER JOIN** 选课表 **INNER JOIN** 课程表 **WHERE** 学生表.**学号**=(**SELECT 班长学号 FROM** 学生表 **WHERE 姓名**=**'典韦'**);

1. 查找名字中倒数第二字为'侯'的学生的学号、姓名和所在系的系名。

**SELECT DISTINCT 学号**, **姓名**, **系名 FROM** 学生表 **INNER JOIN** 系表 **WHERE 姓名 LIKE '%侯\_'**;

1. 查找名字以P打头，倒数第三字为L的课程的名字。

**SELECT DISTINCT 课程名 FROM** 课程表 **WHERE 课程名 LIKE 'P%L\_\_'**;

1. 查找'甘宁'同学所有选修课程的总分。

**SELECT DISTINCT** *SUM*(**成绩**) **FROM** 学生表 **INNER JOIN** 选课表 **WHERE 姓名**=**'甘宁'**;

1. 查找既选修了'数据库'，也选修了'操作系统'的同学。

**SELECT DISTINCT 学号**, **姓名 FROM** 成绩表  
**WHERE** *EXISTS*(**SELECT** *\** **FROM** 成绩表 X **WHERE** X.**学号** = **学号 AND 课程名**= **'数据库'**)  
**AND** *EXISTS*(**SELECT** *\** **FROM** 成绩表 Y **WHERE** Y.**学号** = **学号 AND 课程名**= **'操作系统'**);

1. 查找没有选修'数据库'课程的学生的学号和姓名。

**SELECT DISTINCT 学号**, **姓名 FROM** 成绩表 A  
**WHERE 学号 NOT IN** (**SELECT 学号 FROM** 成绩表 **WHERE '数据库' IN** (**SELECT 课程名 FROM** 成绩表 X **WHERE** X.**学号** = A.**学号**));

1. 查找'数据库'课程及格了，但'编译技术'没有及格的学生的学号和姓名。

**CREATE VIEW** 成绩表 **AS SELECT** 学生表.**学号**, **姓名**, **课程名**, **成绩 FROM** ((学生表 **INNER JOIN** 选课表 **ON** 学生表.**学号** = 选课表.**学号**) **INNER JOIN** 课程表 **ON** 选课表.**课程号** = 课程表.**课程号**) ;  
**SELECT DISTINCT 学号**, **姓名 FROM** 成绩表 A  
**WHERE** *exists*(**SELECT** *\** **FROM** 成绩表 X **WHERE** A.**学号**=X.**学号 AND 课程名**=**'数据库' AND 成绩**>=60)  
 **AND** *exists*(**SELECT** *\** **FROM** 成绩表 Y **WHERE** A.**学号**=Y.**学号 AND 课程名**=**'编译技术' AND 成绩**<60)

1. 查找数据库成绩低于数据库课平均成绩的同学的学号和姓名。

**SELECT 学号**, **姓名 FROM** 成绩表 **WHERE 课程名**=**'数据库' AND 成绩** < (**SELECT** *AVG*(**成绩**) **FROM** 成绩表 **WHERE 课程名**=**'数据库'**);

1. 查找与'貂蝉'同学选修课程完全相同的学生的学号和姓名（不能多选也不能少选）。

**SELECT 学号**,**姓名 FROM** 学生表 学生表OUT  
 **WHERE NOT** *EXISTS*( *#不存在课程此学生选了但"貂蝉"没选,反之亦然* **SELECT 课程号 FROM** 选课表  
 **WHERE** (**课程号 IN** ( *#此学生选了但"貂蝉"没选* **SELECT 课程号 FROM** 选课表  
 **WHERE** 选课表.**学号** = 学生表OUT.**学号** ) **AND 课程号 NOT IN** (  
 **SELECT 课程号 FROM** 选课表,学生表 学生表IN  
 **WHERE** 选课表.**学号** = 学生表IN.**学号 AND** 学生表IN.**姓名** = **'貂蝉'** ) **OR** (**课程号 NOT IN** ( *#或此学生没选但"貂蝉"选了* **SELECT 课程号 FROM** 选课表  
 **WHERE** 选课表.**学号** = 学生表OUT.**学号** ) **AND 课程号 IN** (  
 **SELECT 课程号 FROM** 选课表,学生表 学生表IN  
 **WHERE** 选课表.**学号** = 学生表IN.**学号 AND** 学生表IN.**姓名** = **'貂蝉'** )  
 )));

1. 查找不仅选修了'貂蝉'同学选修的课程，而且还选修了其他课程的同学。

**SELECT 学号**,**姓名 FROM** 学生表 学生表OUT  
 **WHERE** *EXISTS*(*#此学生选了但"貂蝉"没选* **SELECT 课程号 FROM** 选课表  
 **WHERE 课程号 IN** (  
 **SELECT 课程号 FROM** 选课表  
 **WHERE** 选课表.**学号** = 学生表OUT.**学号** ) **AND 课程号 NOT IN** (  
 **SELECT 课程号 FROM** 选课表,学生表 学生表IN  
 **WHERE** 选课表.**学号** = 学生表IN.**学号 AND** 学生表IN.**姓名** = "貂蝉"  
 )) **AND  
 NOT** *EXISTS*( *#或此学生没选但"貂蝉"选了* **SELECT 课程号 FROM** 选课表  
 **WHERE 课程号 NOT IN** (  
 **SELECT 课程号 FROM** 选课表  
 **WHERE** 选课表.**学号** = 学生表OUT.**学号** ) **AND 课程号 IN** (  
 **SELECT 课程号 FROM** 选课表,学生表 学生表IN  
 **WHERE** 选课表.**学号** = 学生表IN.**学号 AND** 学生表IN.**姓名** = "貂蝉"  
 )  
 );

1. 查找'高等数学'平均成绩最高的系的系名。

**CREATE VIEW** 高等数学学生成绩表 **AS  
 SELECT** 学生表.**学号**, 学生表.**姓名**, 系表.**系名**, 选课表.**成绩**, 课程表.**课程名 FROM** ((((学生表  
 **INNER JOIN** 选课表 **ON** 学生表.**学号** = 选课表.**学号**)  
 **INNER JOIN** 系表 **ON** 学生表.**系号** = 系表.**系号**)  
 **INNER JOIN** 课程表 **ON** 选课表.**课程号** = 课程表.**课程号**));  
(**SELECT 系名**, *MAX*(平均成绩) **FROM** (**SELECT 系名**, *AVG*(**成绩**) 平均成绩 **FROM** 高等数学学生成绩表 **WHERE 课程名** = **'数学' GROUP BY 系名**) **as** 学) ;

1. 查找至少有一个籍贯为'四川'同学所选修的课程的课程名。

**CREATE VIEW** 籍贯课程 **AS SELECT 籍贯**, **课程名 FROM** ((学生表 **INNER JOIN** 选课表 **ON** 学生表.**学号** = 选课表.**学号**) **INNER JOIN** 课程表 **ON** 选课表.**课程号** = 课程表.**课程号**);  
**SELECT DISTINCT 课程名 FROM** 籍贯课程 **WHERE** 籍贯课程.**籍贯** = **'四川'**

1. 编写一个函数，能够根据学生获得的成绩计算其获得的学分。并利用该函数检索选修了'数据库'课程的学生的学号和获得的学分。

**CREATE FUNCTION** *getCredit*(Sno **CHAR**(10),Cno **CHAR**(10)) **RETURNS DOUBLE  
 BEGIN  
 DECLARE** score **INT**;  
 **DECLARE** Ccredit **INT**;  
 **DECLARE** ratio **DOUBLE**;  
 **SELECT 成绩 INTO** score **FROM** 选课表  
 **WHERE 学号**=Sno **AND 课程号** = Cno;  
 **SELECT 学分 INTO** Ccredit **FROM** 课程表  
 **WHERE 课程号** = Cno;  
 **SELECT** 计算比率 **INTO** ratio **FROM** 学分计算表  
 **WHERE** 最低成绩 <= score **AND** 最高成绩 >= score;  
 **SET** *@result* = Ccredit\*ratio;  
 **RETURN** @result;  
 **END**;

1. 编写一个存储过程，能够根据输入的学号，计算出该学生所在系的学生人数，平均的选课门数和平均成绩，以及该学生在该系的排名。

**CREATE PROCEDURE** *info*(id **CHAR**(10))  
 **BEGIN  
 DECLARE** Dno **CHAR**(10);  
 **DECLARE** StudentCount **INT**;  
 **DECLARE** CourseCount **DOUBLE**;  
 **DECLARE** AvgScore **DOUBLE**;  
 **DECLARE** Srank **INT**;  
 **SELECT 系号 INTO** Dno **FROM** 学生表 **WHERE 学号**=id;  
 **SELECT** *COUNT*(*\**) **INTO** StudentCount **FROM** 学生表  
 **WHERE 系号** = Dno;  
 **SELECT** *COUNT*(*\**) **INTO** CourseCount **FROM** 选课表 **INNER JOIN** 学生表 **ON** 选课表.**学号** = 学生表.**学号  
 WHERE 系号** = Dno;  
 **SELECT** *AVG*(**成绩**) **INTO** AvgScore **FROM** 选课表 **INNER JOIN** 学生表 **ON** 选课表.**学号** = 学生表.**学号  
 WHERE 系号** = Dno;  
 **SELECT** 排名 **INTO** Srank **FROM** (  
 **SELECT 学号**, *@rank*:=@rank+1 排名 **FROM**(  
 **SELECT** 学生表.**学号**, *AVG*(**成绩**) 平均成绩 **FROM** 选课表 **INNER JOIN** 学生表 **ON** 选课表.**学号** = 学生表.**学号  
 WHERE 系号** = Dno  
 **GROUP BY** 学生表.**学号  
 ORDER BY** 平均成绩 **DESC** ) **AS** temp1,(**SELECT** @rank := 0) **AS** temp2  
 ) **AS** 排名表  
 **WHERE 学号** = id;  
 **SELECT** StudentCount 学生人数,CourseCount/StudentCount 平均选课门数,  
 AvgScore 平均成绩, Srank 排名;  
 **END**;

回答问题：

1. SQL语言的特点。
2. 统一综合
3. 高度非过程化
4. 面向集合的操作方式
5. 以同一种语法结构提供多种使用方式
6. 语言简洁，易学易用
7. 对于数据统计方便直观
8. 基本表和视图有什么区别？

1、视图是已经编译好的sql语句。而表不是  
2、视图没有实际的物理记录。而表有。  
3、表是内容，视图是窗口  
4、表只用物理空间而视图不占用物理空间，  
视图只是逻辑概念的存在，表可以及时四对它进行修改，  
但视图只能有创建的语句来修改  
5、表是内模式，试图是外模式  
6、视图是查看数据表的一种方法，  
可以查询数据表中某些字段构成的数据，  
只是一些SQL语句的集合。从安全的角度说，  
视图可以不给用户接触数据表，从而不知道表结构。  
7、表属于全局模式中的表，是实表；视图属于局部模式的表，  
是虚表。  
8、视图的建立和删除只影响视图本身，不影响对应的基本表。

1. 创建一个数据库，需要创建几个文件，它们分别是做什么用的？它们对应于三级模式中的哪一级？创建的表存储在什么地方？它们对应于三级模式中的哪一级？
2. 数据文件和日志文件。数据文件用于保存数据，日志文件用于记录对数据库的操作，用于恢复数据。
3. 对应内模式。
4. 储存在硬盘里。
5. 对应模式。
6. 可以为表定义哪些完整性约束？它们各自的作用是什么？

1) 实体完整性：规定表的每一行在表中是惟一的实体。  
2) 域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。  
3) 参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。  
4) 用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

1. 自然连接和等值连接有什么差别？

自然连接不需要连接从句，同过两个表的相同列名自动完成等值关系；  
等值连接需要连接从句，通过where从句指定等值连接关系。

1. 子查询分为哪几种？它们之间有什么区别？

**单行子查询：子查询语句只返回单行单列的结果，即返回一个常量值。**

**多行子查询：子查询语句返回多行单列的结果，即返回一系列值。**

**多列子查询：子查询语句返回多列的结果。**

1. 索引有什么作用和缺点？

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。   
第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。   
第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。   
第四，在使用分组和排序 子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。   
第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。   
  
增加索引也有许多不利的方面:  
第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。   
第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。   
第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

1. 视图有什么优点？什么样的视图是可以更新的？

简单性。视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使用户不必为以后的操作每次都指定全部的条件。  
安全性。通过视图用户只能查询和修改他们所能见到的数据。数据库中的其他数据则既看不见也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上，但不能授权到数据库特定行和特定的列上。通过视图，用户可以被限制在数据的不同子集上。  
逻辑数据独立性。视图可以使应用程序和数据库表在一定程度上独立。如果没有视图，应用一定是建立在表上的。有了视图之后，程序可以建立在视图之上，从而程序与数据库表被视图分割开来。

不能更新的条件：

1. 若视图是由两个以上基本表导出的，则此视图不允许更新。

2. 若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT和UPDATE操作，但允许执行DELETE操作。

3. 若视图的字段来自集函数，则此视图不允许更新。

4. 若视图定义中含有GROUP BY子句，则此视图不允许更新。

5. 若视图定义中含有DISTINCT短语，则此视图不允许更新。

6. 若视图定义中有嵌套查询，并且内层查询的FROM子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。例如将成绩在平均成绩之上的元组定义成一个视图GOOD\_SC： CREATE VIEW GOOD\_SC AS SELECT Sno, Cno, Grade FROM SC WHERE Grade > (SELECT AVG(Grade) FROM SC); 　　导出视图GOOD\_SC的基本表是SC，内层查询中涉及的表也是SC，所以视图GOOD\_SC是不允许更新的。

7. 一个不允许更新的视图上定义的视图也不允许更新。

1. 请针对第三章SQL语言讲义中的除法例子，给出其他两种除法的实现方法。

SELECT DISDINCT Sno FROM SC SCX

WHERE NOT EXISTS(

SELECT \* FROM SC SCY

WHERE SCY.Cno IN(

SELECT Cno FROM SC SCZ

WHERE SCZ.Sno = "201215122"

) AND

SCY.Cno NOT IN(

SELECT Cno FROM SC SCZ

WHERE SCZ.Sno = SCX.Sno

)

);

SELECT DISDINCT Sno FROM SC SCX

WHERE NOT EXISTS(

SELECT \* FROM SC SCY

WHERE SCY.Cno = “201215122" AND

SCY.Cno NOT IN(

SELECT Cno FROM SC SCZ

WHERE SCZ.Sno = SCX.Sno

)

);