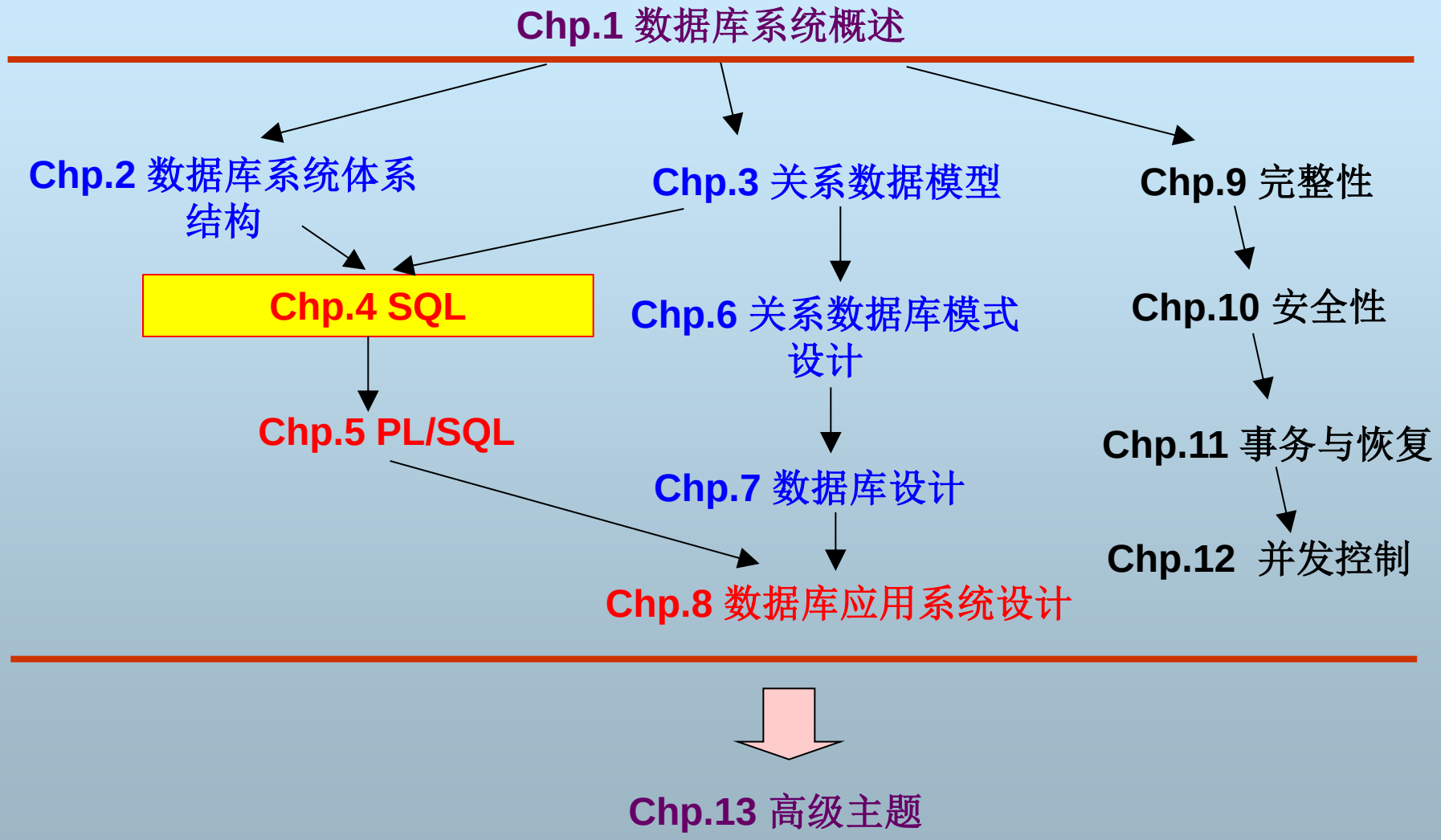
第4章 关系数据库语言SQL

金培权（jpq@ustc.edu.cn）

课程知识结构

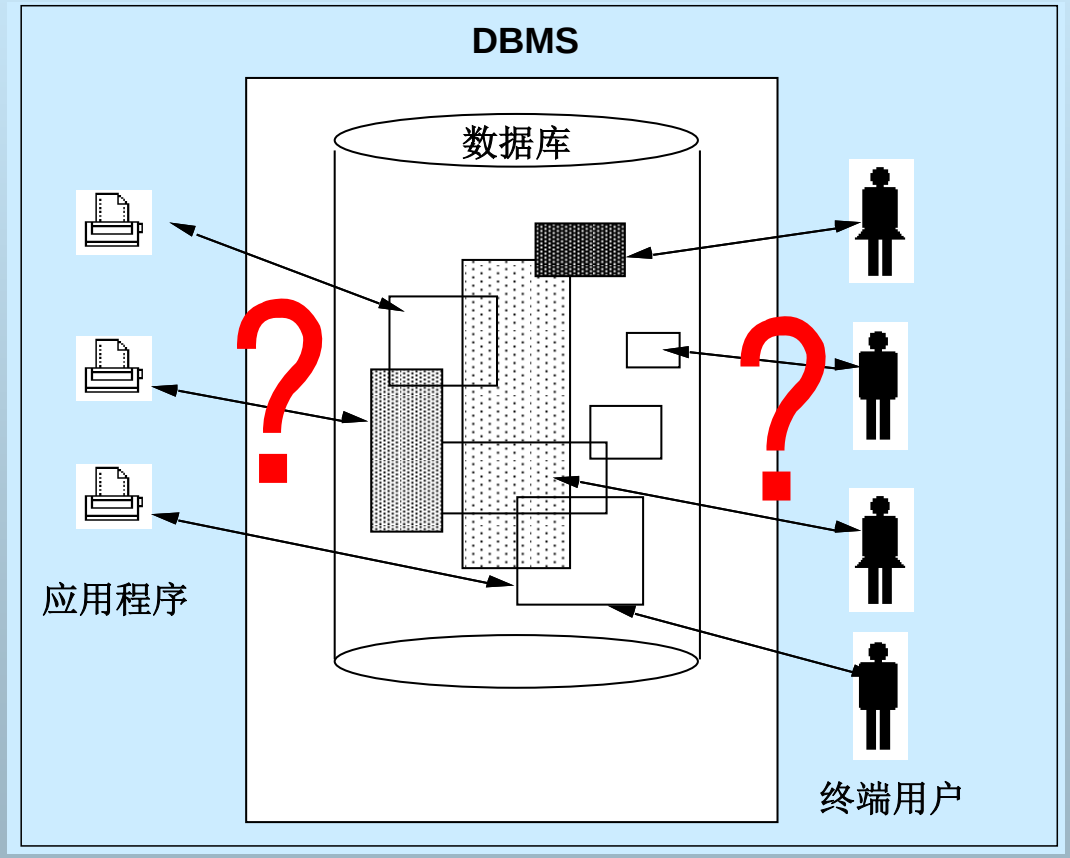

Chp.1 数据库系统概述
Chp.2 数据库系统体系结构
Chp.3 关系数据模型
Chp.9 完整性
Chp.4 SQL
Chp.6 关系数据库模式设计
Chp.10 安全性
Chp.5 PL/SQL
Chp.11 事务与恢复
Chp.7 数据库设计
Chp.12 并发控制
Chp.8 数据库应用系统设计
Chp.13 高级主题

本章主要内容

- 数据库语言
- SQL概述
- SQL DDL
- SQL DML
- 视图

一、数据库语言

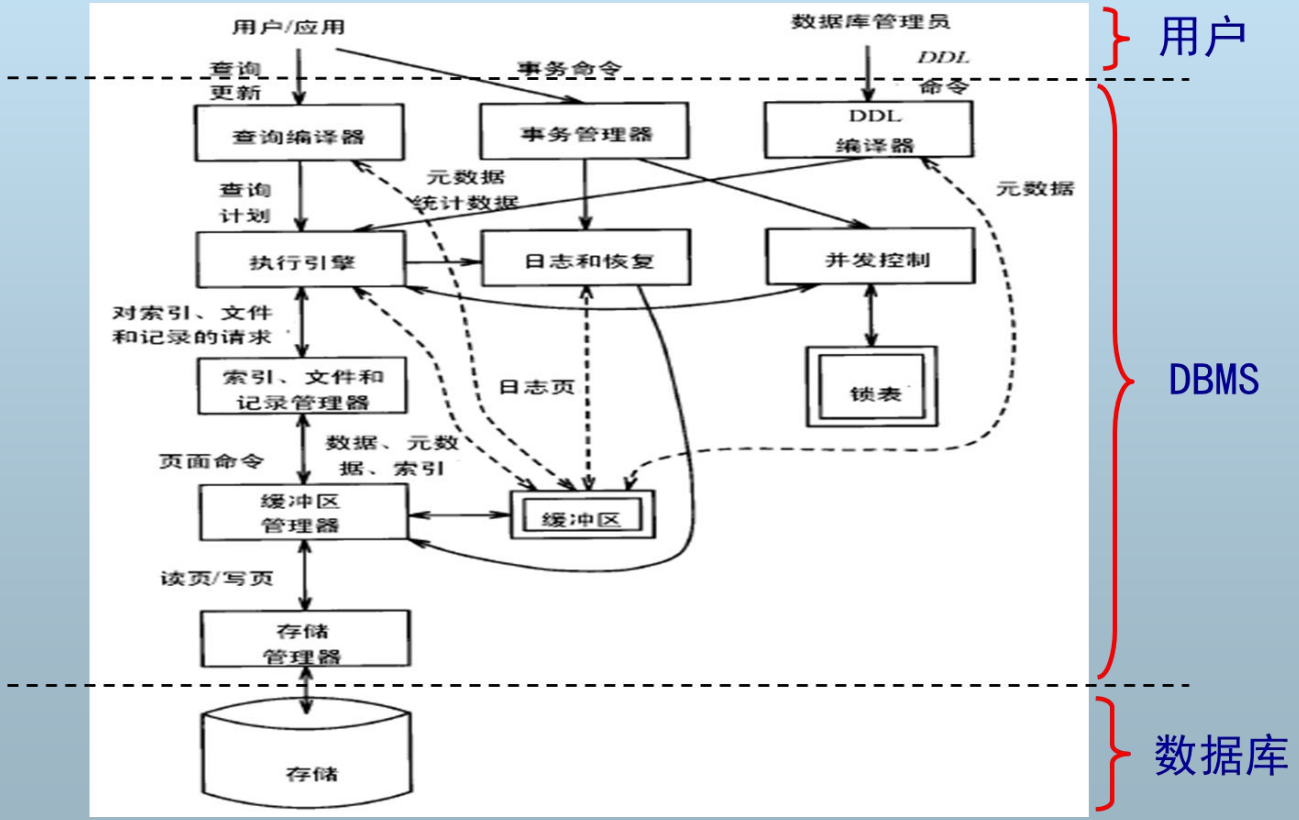
- 用户如何存取数据库中的数据？需要存取哪些数据？



- 需存取三类数据
  - 数据库的存取？
  - 数据库模式的存取？
  - 数据库访问控制信息的存取？

一、数据库语言

- 用户与数据库的唯一接口——数据库语言
- DBMS支持用户通过数据库语言进行数据存取



一、数据库语言

- 数据库语言包括三类子语言
  - 数据定义语言（Data Definition Language, DDL）
    ——存取数据库模式
  - 数据操纵语言（Data Manipulation Language， DML）
    ——存取数据库数据
  - 数据库控制语言（Data Control Language，DCL）
    ——存取访问控制信息

二、SQL概述

- SQL的发展历程
- SQL数据库中的术语
- SQL数据库的三级体系结构
- SQL的组成

1、SQL的发展历程

- 1972：IBM开始研究System R系统，配置了数据库语言SQUARE
  - SQUARE (Specifying Queries As Relational Expressions)
  - 使用了大量的数学符号
- 1974：Boyce和Chamberlin将SQUARE修改为SEQUEL
  - SEQUEL (Structured English QUEry Language)
  - 去掉了数学符号，以英语单词和结构式语法代替查询
  - 后简称为SQL (Structured Query Language)

1、SQL的发展历程

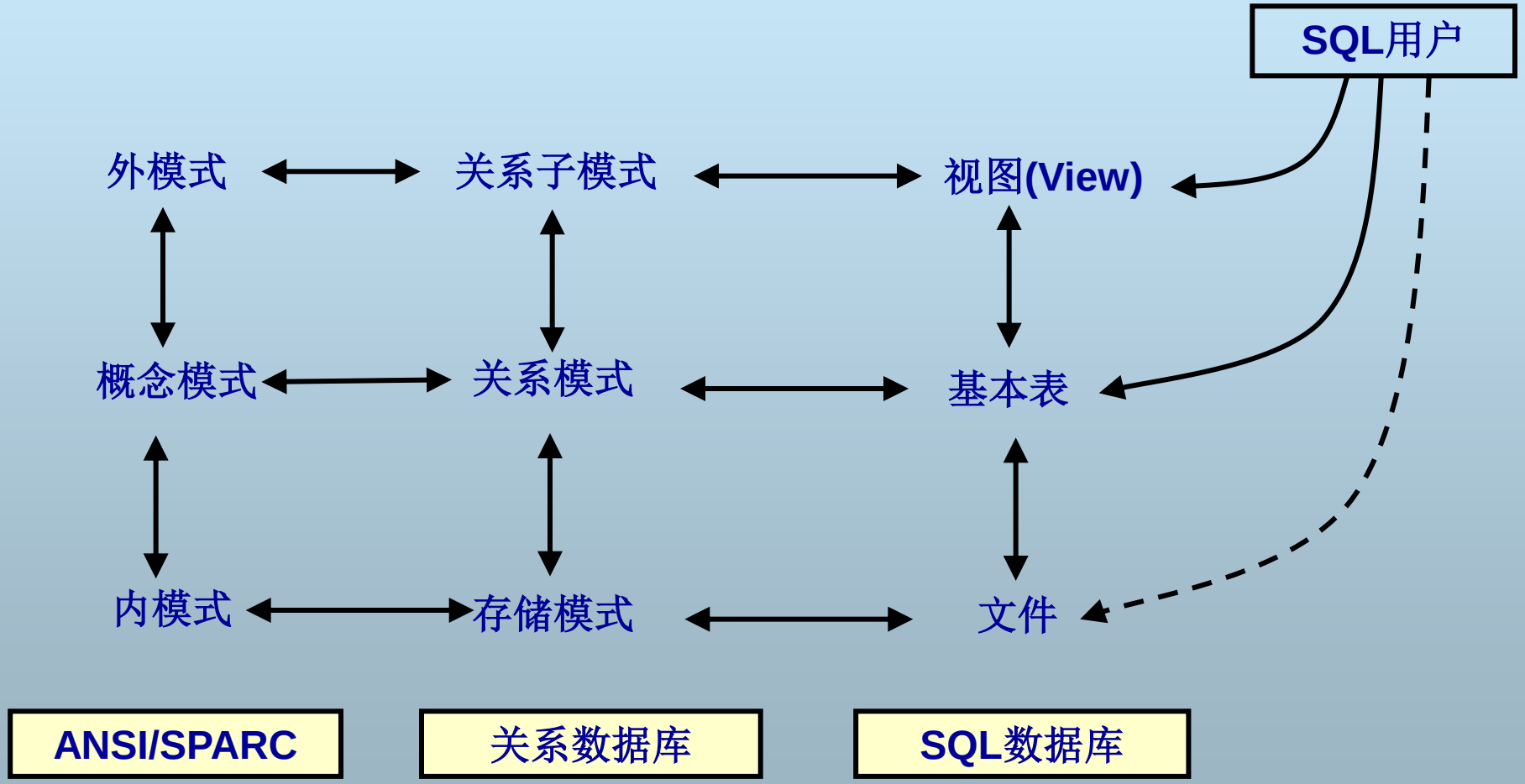
- 1970s末起：主流的数据库厂商纷纷在其产品中支持SQL
  - Oracle、DB2、Sybase等
- 1986.10：ANSI颁布了美国标准的SQL
- 1987.4：ISO采纳美国标准为国际标准，后称“SQL86”
- 1989.4：SQL89，增强了完整性特征
- 1992：SQL92（“SQL2”）
- 1999：SQL3

2、SQL数据库中的术语

- 基本表（Table） ——关系
  - 简称“表”。 表结构——关系模式
- 记录（Record） ——元组
- 字段（列）（Field/Column） ——属性
- 字段值 ——属性值
- 字段类型（列类型） ——域
- 健（Key） ——码
- 主健（Primary Key） ——主码
- 外健（Foreign Key） ——外码

3、SQL数据库的三级体系结构

- **SQL数据库：** 支持SQL语言的关系数据库



4、SQL的组成


SQL
DDL
（模式）
Create Table/Alter Table/Drop Table：定义基本表
Create View/Drop View：视图操作
Create Index/Drop Index：索引操作
DML
（记录）
Insert
Delete
Select
Update
记录操作
DCL
（权限）
Grant
Revoke
权限管理
嵌入式SQL

三、SQL的数据定义——DDL

- 基本表的结构
- 创建基本表：Create Table
- 修改基本表：Alter Table
- 删除基本表：Drop Table

1、基本表的结构

- 一个基本表的结构包括：
  - 表名 ——对应关系模式名
  - 列 ——对应关系模式的属性
  - 完整性约束 ——对应关系模式的三类完整性

（1）列

- 列名
  - 字母开头，可含字母、数字、＃、$、_
  - <=30字符
- 列类型
  - Char(n)　　【定长字符串类型】
  - Varchar2(n)　　【可变长字符串类型】
  - Number　　【数值型】
  - Date　　【日期时间型】
  - ……

（1）列

	ANSI/ISO	Oracle
字符型	Char(n)	Char(n)
	Character(n)	
	Varchar(n)	Varchar2(n)
	Char Varying(n)	
数值型	Numeric	Number
	Decimal	
	Integer	
	Int	
	Float	
	Double	
	Real	
日期型	Date	Date
	Time	

- Oracle数据类型与ANSI有一定的差别
- 若使用ANSI类型，Oracle自动转换为Oracle类型

（2）完整性约束

- 主键约束（Primary Key）
  - 实体完整性
- 唯一键约束（Unique）
  - 定义候选码
- 外键约束（Foreign Key）
  - 参照完整性
- 检查约束（Check）
  - 用户自定义完整性

这些约束既可以定义在列上，也可以定义在基本表之上

列约束：在每列后定义，只对当前列有效

表约束：在全部列定义后定义，可定义多个列上的约束

2、创建基本表

- 基本表构成：表名，列和约束
- Create Table <基本表名>（
  列名1 列类型1 [列约束1],
  列名2 列类型2 [列约束2],
  ……
  [表约束]
  ）
  - 定义列
  - 定义约束

在SQL数据库中，不一定必须定义主键，这与关系模型有差别

```
Create Table Student(
  S#  Varchar2(10) Constraint PK Primary Key,
  Sname Varchar2(20),
 Age Number(3),
 Sex Char(1)
)
```

（1）定义列

- 完整格式
  - <列名> <列类型> [DEFAULT <默认值>] [[NOT] NULL] [<列约束>]

```
Create Table Student(
   S#  Varchar2(10) Constraint PK Primary Key,
   Sname Varchar2(20) NOT NULL,
   Age Number(3),
   Sex Char(1) DEFAULT 'F'
)
```

NOT NULL表示不允许空值，实际上是Check约束的简化

A）默认值

- 当往表中插入一条新记录时，如果某列上有默认值，并且新记录中未指定该列的值，则自动以默认值填充

```
Insert Into Student(s#,sname,age)
Values('001','John',20)
```

插入一条新记录

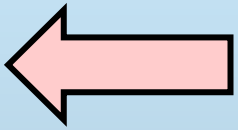
S#	Sname	Age	Sex
001	John	20	F

自动以默认值填充

B）列约束

- 必须更在每个列定义后定义
- 只对当前列有效
- 可以使用四种类型的约束
- 格式
  - [Constraint <约束名>] <约束类型>
- 例
  - S# char(n) Constraint PK_Student Primary Key
  - S# char(n) Primary Key

Where are we?

- 创建基本表
  - 定义列
  - 定义约束 

（2）定义约束

- 列约束：在每个列后定义，可以有多个约束子句
  - 但不能定义多个列上的约束
- 表约束：在全部列定义完成后定义，可以有多个约束子句
  - 多个列上的约束必须使用表约束
  - 单列上的约束可以用列约束，也可用表约束
- 四种约束都可以作为列约束或表约束

A）列约束和表约束举例

```
Create Table Student(
   S#  Varchar2(10) Constraint PK_S Primary Key,
   Sname Varchar2(20),
  Age Number(3) Constraint CK_S Check (age>14 and age<100),
  Sex Char(1)，
  Constraint UQ_S Unique(Sname),
  Constraint CK_SS Check (Sex IN (‘M’,’F’))
)
```

B)Primary Key约束

- 定义主键：不许有空值，也不可重复

```
Create Table Student(
   S#  Varchar2(10) Constraint PK_S Primary Key,
   Sname Varchar2(20),
  Age Number(3) ,
  Sex Char(1))
```


```
Create Table SC( ——选课表
   S#  Varchar2(10)
   C# Varchar2(20),
   Score Number(3) ,
   Constraint PK_SC Primary Key(S#,C#))
```

C)Unique约束

- 唯一性约束：值不可重复，但可以为空

```
Create Table Department(
   NO  Varchar2(10),
   NAME Varchar2(20),
  SCHOOL Char(20),
  Constraint UQ_D Unique(NAME，SCHOOL)
)
)
```

多个列上的约束只能用表约束来实现

D)Unqiue约束对空值的处理

若约束列中有一列不为空，就实施约束；若约束列都为空，则不实施约束

NO	NAME	SCHOOL		
1	管理系	商学院		
2	管理系	管理学院	OK	值唯一
3	管理系	管理学院	Error!	值重复
4	管理系		OK	值唯一
5		管理学院	OK	值唯一
6		管理学院	Error!	实施约束
7			OK	约束列都空，
8			OK	不实施约束

E)Foreign Key约束

- 外键约束：表中某列值引用其它表的主键列或Unique列，参照完整性含义

```
Create Table Student(
  S#  Varchar2(10) Constraint PK_S Primary Key,
  Sname Varchar2(20),
  Age Number(3))
```

```
Create Table SC( ——选课表
  S#  Varchar2(10) Constraint FK_SC References Student(S#),
  C# Varchar2(20),
  Score Number(3) ,
  Constraint FK_SC Foreign Key(S#) References Student(S#))
```

F)Foreign Key约束示例

被参照表（主表）：Student表

S#	SNAME	AGE
001	John	20
002	Rose	21

参照表（子表）：SC表

S#	C#	Score
001	c001	90

```
Insert Into SC values('003','c001',85);   -- Error!!

Delete From Student where S#='001';   -- Error!!
```

在子表中（如SC）插入记录时，若主表中对应的列值不存在，则插入出错；

删除主表中的记录时，若有子表中的相应记录存在，也出错。——若设置了级联删除则不会出错

G)Foreign Key约束的选项

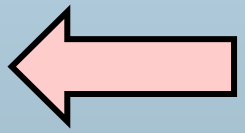
- **级联删除：**删除主表中的记录时，同时删除子表中相关联的记录:
  On Delete Cascade
- **级联设空：**删除主表中的记录时，同时将子表中相应记录的外键列值设为空:
  On Delete Set NULL

```
Create Table SC( ——选课表
  S#  Varchar2(10) ,
  C# Varchar2(20),
  Score Number(3) ,
  Constraint FK_SC Foreign Key(S#) References Student(S#) On Delete Cascade
  )
```

H)Check约束

- 检查约束：自定义某些列上的约束
  - Constraint CK_S1 Check (Age>15)
  - Constraint CK_S2 Check (Sex In ('M','F'))
  - Constraint CK_SC Check (Score>=0 and Score<=100)
  - Constraint CK_S3 Check (Sname Is Not NULL)

Where are we?

- SQL的数据定义——DDL
  - 基本表的组成
  - 创建基本表
    - 定义列
    - 定义约束
  - 修改基本表 
  - 删除基本表

2、修改基本表

- Alter Table <表名>
  [Add <列定义>] |
  [Modify <列定义>] |
  [Rename Column <old> To <new>]|
  [Drop Column <列名>] |
  [Add <表约束>] |
  [Drop Constraint <约束名>] |
  [Rename To <new_table_name>]

（1）增加列

- Alter Table <表名>
  Add <列定义>
- <列定义>与Create Table中相同

```
Alter Table Student(
        Add Class Varchar2（10）
)
```

```
Alter Table Student(
        Add Dept Varchar2（10） Constraint UQ_S3 UNIQUE
)
```

（2）删除列

- Alter Table <表名>
  Drop Column <列名>

```
Alter Table Student(
        Drop Column age
)
```

（3）修改列

- Alter Table <表名>
  Modify <列定义>
- <列定义>与Create Table中相同
  - 但列名不能修改

```
Alter Table Student(
        Modify  age Integer NOT NULL
)
```

（4）重命名列

- Alter Table <表名>
  Rename Column <old> To <new>

```
Alter Table Student(
        Rename Column sex To gender
)
```

（5）增加约束

- **Alter Table <表名>**
  Add <表约束>
- **只能增加表约束**
- **表约束格式与创建表时相同**

```
Alter Table Student(
        Add Constraint PK_Student Primary Key(S#)
)
```

（6）删除约束

- **Alter Table <表名>**
  Drop Constraint <约束名>

```
Create Table SC( ——选课表
  S#  Varchar2(10) ,
  C# Varchar2(20),
  Score Number(3) ,
  Constraint FK_SC Foreign Key(S#) References Student(S#) On Delete Cascade
  )
```

```
Alter Table SC(
        Drop Constraint FK_SC
)
```

（7）重命名表

- **Alter Table <表名>**
  Rename To <新的表名>

```
Create Table SC( ——选课表
  S#  Varchar2(10) ,
  C# Varchar2(20),
  Score Number(3) ,
  Constraint FK_SC Foreign Key(S#) References Student(S#) On Delete Cascade
  )
```

```
Alter Table SC(
        Rename To course_selection
)
```

3、删除基本表

- Drop Table <表名> [Cascade Constraints]
- Cascade Constraints表示删除表时同时删除该表的所有约束

```
Drop Table Student
```

```
Drop Table Student Cascade Constraints
```

四、DML——插入/修改/删除记录

- DML
  - Insert：插入记录
  - Delete：删除记录
  - Update：修改记录
  - Select：查询记录

1、插入新记录到基本表中

- Insert Into <表名> (列名1，列名2，……，列名n)
  Values（值1，值2，……，值n）

```
Create Table Student(
   S#  Varchar2(10) Constraint PK Primary Key,
   Sname Varchar2(20),
  Age Number(3),
  Sex Char(1) DEFAULT 'F'
)
```

例1：

```
Insert Into Student（S#, Sname, Age, Sex）
Values（'s001'，'John'，21，'M'）
```

（1）Insert其它例子

例2：
Insert Into Student

Values（'s002'，'Mike'，21，'M'）

如果插入的值与表的列名精确匹配（顺序，类型），则可以省略列名表

例3：
Insert Into Student（s#, sname）

Values（'s003'，'Mary'）

如果列名没有出现在列表中，则插入记录时该列自动以默认值填充，若没有默认值则设为空

S#	Sname	Age	Sex
s003	Mary		F

（2）日期数据的插入

例4：

```
Alter Table Student Add birth Date;

Insert Into Student Values('s004', 'Rose', 22, 'F',
to_date('11/08/1981', 'dd/mm/yyyy'));

Insert Into Student Values('s005', 'Jack', 22, 'M', to_date('12-08-
1981', 'dd-mm-yyyy'));
```

使用To_Date函数插入日期型

2、修改表中的数据

- Update <表名>
  Set <列名1>=<值1>，<列名2>=<值2>，……
  Where <条件>
- 将符合<条件>的记录的一个或多个列设置新值

（1）Update例子

- 将学生John的性别改为'F'，年龄改为23

例1：
```
Update Student
Set sex＝ ' F ' ， age＝23
Where sname＝ ' John '
```

- 将所有学生的年龄都减1岁

例2：
```
Update Student
Set age＝age－1
```

3、删除表中的记录

- **Delete From <表名>**
  Where <条件>
- **将符合<条件>的记录从表中删除**

例1：从数据库中删除学号为s001的学生

```
Delete From Student
Where s# = 's001'
```

例2：从数据库中删除所有的学生

```
Delete From Student
```

五、DML：查询数据

- SELECT查询结构
- SELECT基本查询
- 联接查询
- 嵌套查询
- 查询结果的连接：并、交、差

1、Select查询结构

- Select <列名表> ——指定希望查看的列
  From <表名列表> ——指定要查询的表
  Where <条件> ——指定查询条件
  Group By <分组列名表> ——指定要分组的列
  Having <条件> ——指定分组的条件
  Order By <排序列名表> ——指定如何排序

2、Select基本查询

- 查询全部记录：查询全部的学生信息
  - Select * From Student
  - *表示所有列
  - 等同于
    Select s#, sname, age, sex From Student
- 查询特定的列：查询所有学生的学号和姓名
  - Select s#, sname From Student

2、Select基本查询

- 使用别名：查询所有学生的学号和姓名
  - Select s# AS 学号, sname AS 姓名 From Student
  - 如果别名包含空格，须使用双引号
  - Select s# AS “Student Number” From Student

2、Select基本查询

- 使用表达式：查询所有学生的学号、姓名和出生年份，返回两列信息，其中一列是“学号：姓名”，另一列是出生年份
  - Select s# || ':' || sname AS 学生，2003－age AS 出生年份 From Student
  - 字符串表达式
  - 算术表达式
  - 函数表达式
    - Select sno, to_char(birth, 'mm-dd-yyyy') AS birthday From Student
    - Select Count(sno) As 学生人数 From Student

2、Select基本查询

- 检索特定的记录：查询20岁以上的学生的学号和姓名
  - Select s# AS 学号, sname AS 姓名 From Student Where age > 20
  - 无Where子句时返回全部的记录
  - WHERE子句中的关系运算符
    - 算术比较符：>, <, >=, <=, =, <>
    - IN
    - IS NULL和IS NOT NULL
    - LIKE
    - EXISTS

2、Select基本查询

- IN：查询‘s001’,’s003’,’s006’和’ s008’四学生的信息
  - Select * From Student
    Where s# IN (‘s001’,’s003’,’s006’,’s008’)
- IS [NOT] NULL：查询缺少年龄数据的学生
  - Select * From Student Where age IS NULL
- LIKE：查询姓名的第一个字母为‘R’的学生
  - Select * From Student Where sname LIKE ‘R%’
  - %：任意长度的字符串
  - _：单个字符
  - 查询姓名的第一个字母为’R’并且倒数第二个字母为’S’的学生
  - Select * From Student Where sname LIKE ’R%S_’
- 多个比较式可用NOT、AND和OR连接
  - Select * From Student
    Where age IS NULL and sname LIKE ‘R%’

2、Select基本查询

- 去除重复记录：查询学生的姓名
  - Select Distinct sname From Student
  - Distinct只对记录有效，不针对某个特定列
    - Select Distinct sname, age From Student
- 排序查询结果：
  - 查询所有学生信息并将结果按年龄升序排列
  - Select * From Student Order By age
  - 将结果按年龄升序排列,按姓名降序排列
  - Select * From Student
    Order By age ASC, sname DESC
  - ASC表示升序，DESC表示降序

2、Select基本查询

- 使用聚集函数
  - Count(列名)：对一列中的值计数
  - Count(*)：计算记录个数
  - SUM(列名)：求一列值的总和（数值）
  - AVG (列名)：求一列值的平均值
  - MIN (列名)：求一列值的最小值
  - MAX (列名)：求一列值的最大值

2、Select基本查询

- 聚集函数例子
  - 求学生的总人数
    - Select count(*) From student
  - 求选修了课程的学生人数
    - Select count(distinct s#) From SC
  - 求学生的平均年龄
    - Select avg(age) as average_age From student
- 单独使用聚集函数时（Select子句中的列名都是聚集函数形式），表示对所有记录进行聚集

2、Select基本查询

- 聚集函数和分组操作：
  - 聚集函数：MIN, MAX, SUM, AVG, COUNT
  - 聚集函数一般与分组操作一起使用$\gamma_L(R)$
  - 查询男生和女生的平均年龄
    - Select sex, AVG(age) as Average_age From Student
      Group By sex

  - 除聚集函数外的属性必须全部出现在Group By子句中

2、Select基本查询

- 返回满足特定条件的分组结果
  - 查询不同年龄的学生人数，并返回人数在5人以上的结果
    - Select age, COUNT(*) as students From Student
      Group By age
      Having COUNT(*) > 5
  - Having子句中必须聚集函数的比较式，而且聚集函数的比较式也只能通过Having子句给出
  - Having中的聚集函数可与Select中的不同
  - 查询人数在60以上的各个班级的学生平均年龄
    - Select class, AVG(age) From Student
      Group By class
      Having COUNT(*) > 60

3、连接查询

- 一个查询从两个表中联合数据
- 返回两个表中与联接条件相互匹配的记录，不返回不相匹配的记录

Student表

S#	Sname	Age
01	Sa	20
02	Sb	21
03	sc	21

SC表(s#是外键, c#是外键)

S#	C#	Score
01	C1	80
01	C2	85
02	C1	89

Course表

c#	Cname	credit
C1	Ca	3
C2	Cb	4
C3	Cc	3.5

（1）连接查询例子

- 查询学生的学号，姓名和所选课程号
  - Select student.s#, student.sname,sc.c#
    From student,sc
    Where student.s# = sc.s# ——连接条件
- 若存在相同的列名，须用表名做前缀
- 查询学生的学号，姓名，所选课程号和课程名
  - Select student.s#,
    student.sname,sc.c#,course.cname
    From student,sc,course
    Where student.s# = sc.s# and sc.c# = course.c#
    ——连接条件

（2）使用表别名

- 查询姓名为'sa'的学生所选的课程号和课程名
  - Select b.c#, c.cname
    From student a, sc b, course c
    Where a.s#=b.s# and b.c#=c.c# and
    a.sname='sa'
  - 表别名可以在查询中代替原来的表名使用
- 联接查询与基本查询结合：查询男学生的学号，姓名和所选的课程数，结果按学号升序排列
  - Select a.s#, b.sname, count(b.c#) as c_count
    From student a, sc b
    Where a.s# = b.s# and a.sex='M'
    Group By a.s#, b.sname
    Order By student.s#

4、嵌套查询

- 在一个查询语句中嵌套了另一个查询语句
- 三种嵌套查询
  - 无关子查询
  - 相关子查询
  - 联机视图

（1）无关子查询

- 父查询与子查询相互独立，子查询语句不依赖父查询中返回的任何记录，可以独立执行
- 查询没有选修课程的所有学生的学号和姓名
  - Select s#,sname
    From student
    Where s# NOT IN (select distinct s# From sc)
  - 子查询返回选修了课程的学生学号集合，它与外层的查询无依赖关系，可以单独执行
  - 无关子查询一般与IN一起使用，用于返回一个值列表

（2）相关子查询

- 相关子查询的结果依赖于父查询的返回值
- 查询选修了课程的学生学号和姓名
  - Select s#, sname
    From student
    Where EXISTS (Select * From sc Where sc.s# = student.s#)
  - 相关子查询不可单独执行，依赖于外层查询
  - EXISTS（子查询）：当子查询返回结果非空时为真，否则为假
  - 执行分析：对于student的每一行，根据该行的s#去sc中查找有无匹配记录

（3）联接视图

- 子查询出现在From子句中作为表使用
- 查询只选修了1门或2门课程的学生学号、姓名和课程数
  - Select s#, count_c#
    From (Select s#, count(c#) as count_c#
    From sc
    Group by s#) sc2, student s
    Where sc2.s# = s.s# and (count_c#=1 OR
    count_c#=2)
- 联机视图可以和其它表一样使用

5、查询结果的连接

- Union和Union All
- Minus
- Intersect

（1）Union和Union All

- 查询课程平均成绩在90分以上或者年龄小于20的学生学号
  - (Select s# From student where age<20)
    UNION
    (Select s#
    From (Select s#, AVG(score)
    From SC
    group by s#
    having avg(score)>90) SC2
)
- UNION操作自动去除重复记录 ——Set Union
- Union All操作不去除重复记录 ——Bag Union

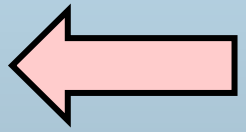
（2）Minus操作：差

- 查询未选修课程的学生学号
  - (Select s# From Student)
    Minus
    (Select distinct s# From SC)

(3)Intersect操作

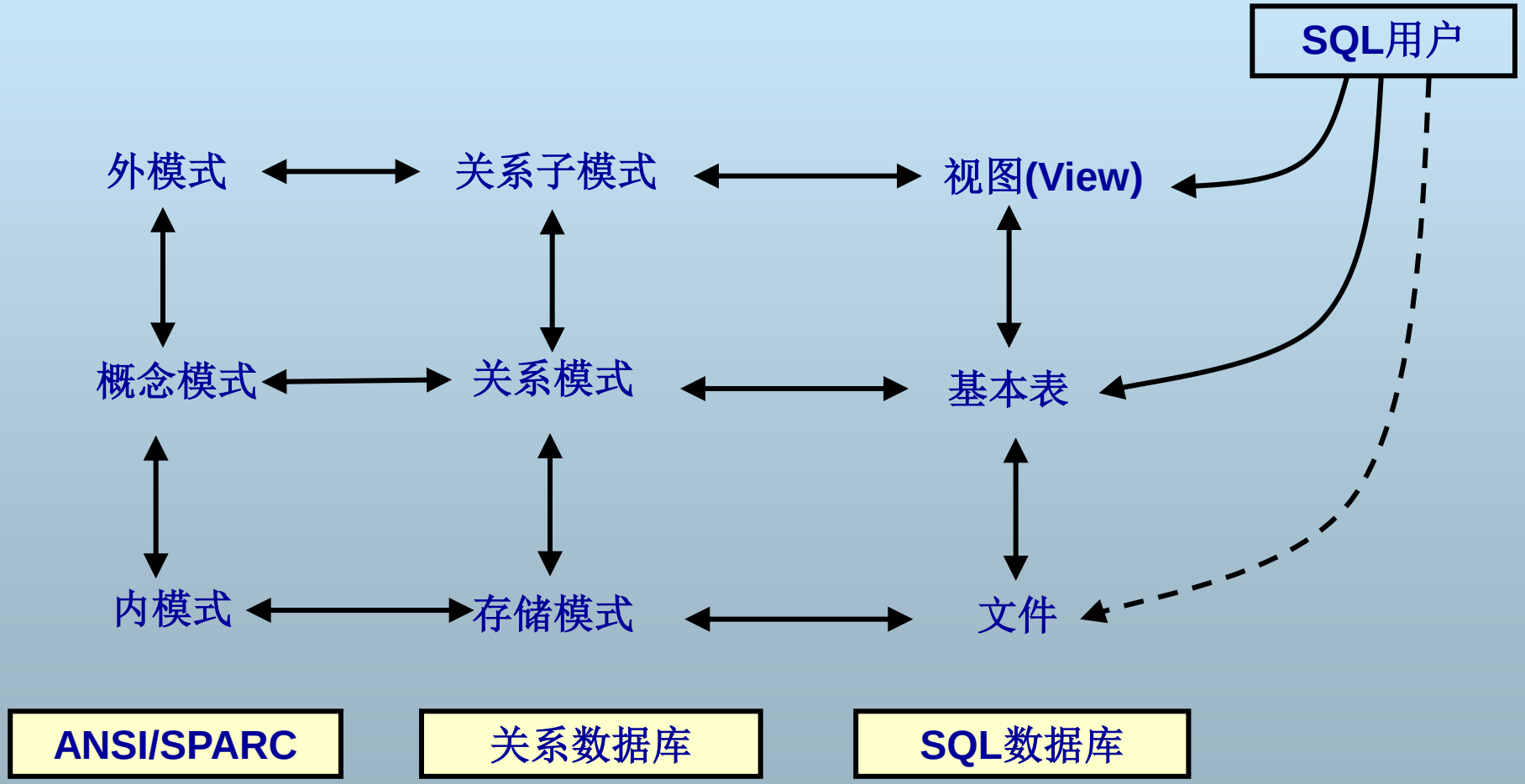
- 返回两个查询结果的交集
- 查询课程平均成绩在90分以上并且年龄小于20的学生学号
  - (Select s# From student where age<20)
    Intersect
    (Select s#
    From (Select s#, AVG(score)
    From SC
    group by s#
    having avg(score)>90) SC2
)

Where are we?

- 数据库语言
- SQL概述
- SQL DDL
- SQL DML
- 视图 

六、视图（View）

- 视图（View）给出了SQL数据库的外模式定义

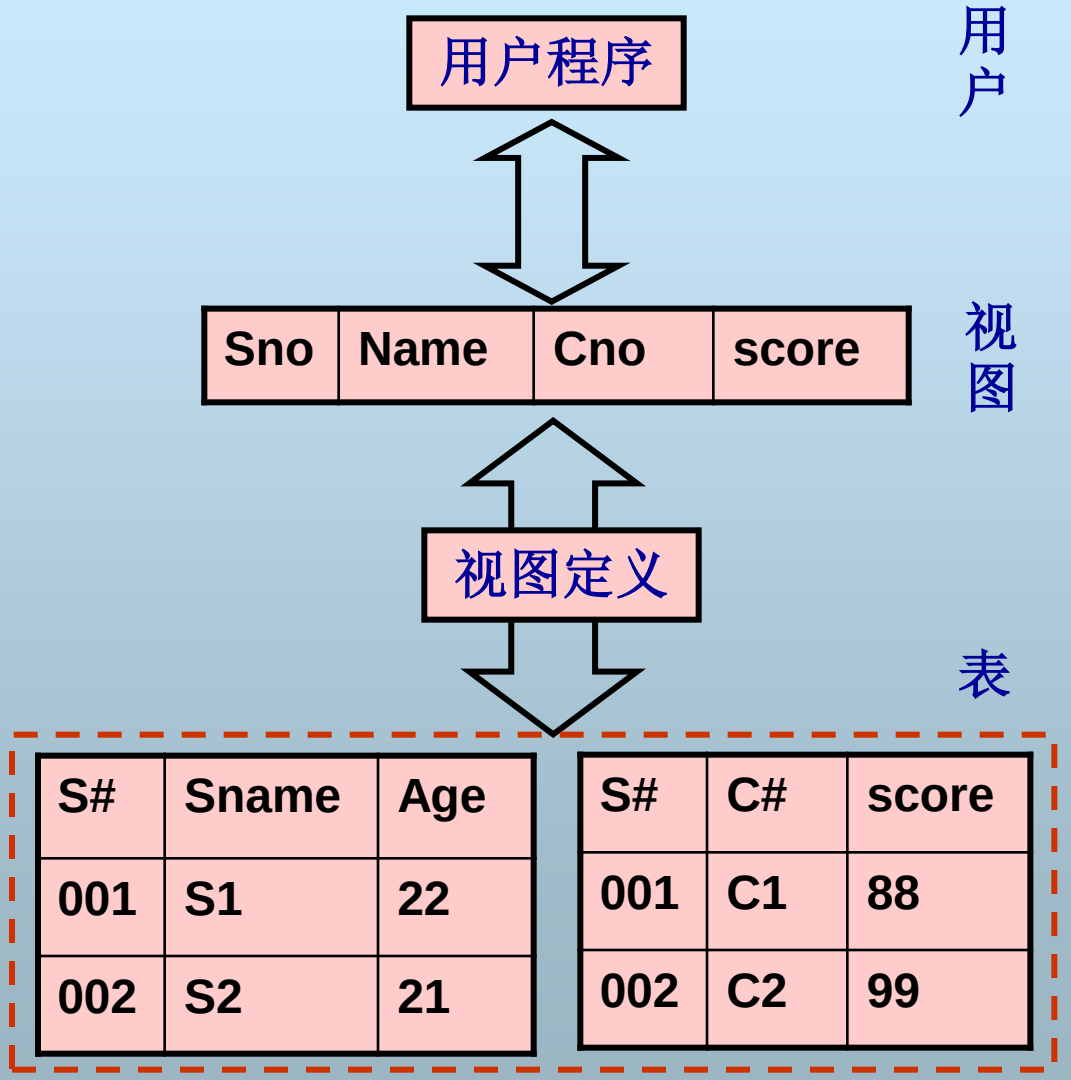


1、视图的概念

- 视图是从一个或几个基本表中导出的虚拟表，其数据没有实际存储，但可以和表一样操作
  - 视图具有和表一样的逻辑结构定义
  - 但视图没有相应的存储文件，而每个表都有相应的存储文件

2、视图的用途

- 逻辑数据独立性：用户程序与数据库结构
- 简化了用户眼中的数据，使用户可以集中于所关心的数据上
- 同一数据库对不同用户提供不同的数据呈现方式
- 安全保护



3、视图的定义

- Create View <视图名>（列名1，列名2，...）
  AS <查询>
  [With Read Only]
- <查询>是一个Select语句，指明视图定义在哪些基本表上，定义了什么内容的数据
- <列名表>定义了视图的逻辑结构，与<查询>中返回的数据相对应
- 若加上With Read Only选项表示所定义的视图是只读视图

3、视图的定义

- 例1：定义计算机系的学生视图
  - Create View cs_view (sno, name, age)
    As Select s#,sname,age
    From student
    Where Dept='计算机系'
    With Read Only

cs_view(sno,name,age)

  - Create View cs_view
    As Select s#,sname,age
    From student
    Where Dept='计算机系'
    With Read Only

cs_view(s#,sname,age)

  - 若省略视图的列名表，则自动获得Select查询返回的列名

3、视图的定义

- 例2：把每门课程的课程号和平均成绩定义为视图
  - Create View c_view
    As Select c#, AVG(score) as avg_score
    From sc
    Group By c#
  - Create View c_view (cno, avg_score)
    As Select c#, AVG(score)
    From sc
    Group By c#
- 在查询中使用了函数时
  - 若省略列名表，则必须为函数指定别名
  - 若使用了列名表，则可以不指定函数的别名

4、视图的查询

- 与基本表的查询相同
- 例：查询平均成绩在80分以上的课程号与课程名
  - 不使用视图
    - Select a.c#, a. cname
      From Course a, (select c#,avg(score) as avg_score From sc Group By c#) SC2
      Where a.c#=sc2.c# and SC2.avg_score>80
  - 使用前面定义的视图 c_view
    - Select a.c#, a.cname From course a, c_view b
      Where a.c#=b.c#

5、视图的更新

- 与表的更新类似
- 例：将计算机系学号为'001'的学生的姓名改为'Rose'
  - Update cs_view
    Set name='Rose'
    Where s#='001'
  - 执行时先转换为student上的更新语句再执行
- 不是所有视图都是可更新的
  - 基于联接查询的视图不可更新
  - 使用了函数、表达式、Distinct的视图不可更新
  - 使用了分组聚集操作的视图不可更新
- 只有建立在单个表上，而且只是去掉了基本表的某些行和列，但保留了主键的视图才是可更新的

6、视图的删除

- Drop View <视图名>

本章小结

- SQL数据库
- DDL:
  - Create Table/Alter Table/Drop Table
- DML
  - Insert, Delete, Update
  - Select: 基本查询、连接查询、嵌套查询
- View
  - 作用与优点、Create View/Drop View

本章小结

- Select <列名表> ——指定希望查看的列
  From <表名列表> ——指定要查询的表
  Where <条件> ——指定查询条件
  Group By <分组列名表> ——指定要分组的列
  Having <条件> ——指定分组的条件
  Order By <排序列名表> ——指定如何排序