上堂课程主要内容复习

- 关系数据库
  - 关系数据结构 – 关系
    - 基本概念：关系、域、笛卡儿积、候选码、主码、主属性、外部码、关系模式R(U)、型、值、.....
  - 关系操作
    - 关系代数：集合方式；基本操作：查、增、删、改
      - 关系运算：选择、投影、连接、除
    - SQL
  - 关系的完整性
    - 关系的值随时间变化时应该满足的约束条件。
  - 关系数据理论

- 已知学生课程数据库中包含学生关系S，课程关系C和选修关系SC，关系模式分别为
  - S(Sno, Sname, Sgender, Sage, Sdept)
  - C(Cno, Cname, Cpno, Ccredit)
  - SC(Sno, Cno, Grade)

- 查询计算机系的全体学生；
- $\sigma_{Sdept='CS'}(S)$
- 查询都有哪些系；
- $\pi_{Sdept}(S)$
- 查询至少选修1号课程和3号课程的学生学号；
- (1) 建立临时关系K（Cno），只包含1、3两门课程编号
- (2) $\pi_{Sno,Cno}(SC) \div K$

- 已知学生课程数据库中包含学生关系S，课程关系C和选修关系SC，关系模式分别为
  - S(Sno, Sname, Sgender, Sage, Sdept)
  - C(Cno, Cname, Cpno, Ccredit)
  - SC(Sno, Cno, Grade)

- 查询选修了2号课程的学生学号；
- $\pi_{Sno}\left(\sigma_{Cno='2'}(SC)\right)$
- 查询至少选修了一门其先修课程为5号课程的学生姓名；
- $\pi_{Sname}\left(\sigma_{Cpno='5'}(C)\infty SC\infty\pi_{Sno,Sname}(S)\right)$
- $\pi_{Sname}\left(\pi_{Sno}\left(\sigma_{Cpno='5'}(C)\infty SC\right)\infty\pi_{Sno,Sname}(S)\right)$
- 查询选修了全部课程的学生学号和姓名。
- $\pi_{Sno,Cno}(SC)\div\pi_{Cno}(C)\infty\pi_{Sno,Sname}(S)$

关系数据库标准语言SQL语言

2020/4/2

主要内容

- **SQL概述**
- 数据查询
- 数据操纵
- 数据定义
- 视图

1 SQL概述

- **SQL的发展及现状**
- **SQL的特点**
- **SQL的基本概念**
- **SQL的功能**
- **SQL的应用形式**

1.1 SQL的发展及现状

SQL：Structured Query Language

–1974年，由Boyce和Chamberlin提出，命名Sequel

– Structured English Query Language” (SEQUEL)

–1975~1979，IBM San Jose Research Lab的关系

数据库管理系统原型System R实施了这种语言

–SQL-86是第一个SQL标准文本

–SQL-89、SQL-92(SQL2)、SQL-99(SQL3)

–SQL2003、SQL2008、SQL2011、……

–大部分DBMS产品都支持SQL

Pronouncing SQL: S-Q-L or Sequel?

* Hello Don,
* I'm sorry to waste your time with such a silly question, but I've often heard SQL pronounced S-Q-L or as Sequel. I've also seen the official pronunciation listed both ways. According to wikipedia, you and Raymond Boyce created the language and it was shortened to SQL after some legal dispute. So my question is, is there an official pronunciation to SQL? Thank you for your time.
* Pat
* Hi Pat,
* Since the language was originally named SEQUEL, many people continued to pronounce the name that way after it was shortened to SQL. Both pronunciations are widely used and recognized. As to which is more "official", I guess the authority would be the ISO Standard, which is spelled (and presumably pronounced) S-Q-L.
* Thanks for your interest,
* Don Chamberlin

1.2 SQL的特点

1. 高度非过程化的语言：用户只需提出“干什么”，至于“怎么干”由DBMS解决；用户只需要在查询语句中提出需要什么，DBMS即可完成存取操作，并把结果返回给用户。

2. 面向集合的语言：每一个SQL的操作对象是一个或多个关系，操作的结果也是一个关系。

3. 一种语法结构，两种使用方式：即可独立使用，又可嵌入到宿主语言中使用，具有自主型和宿主型两种特点。

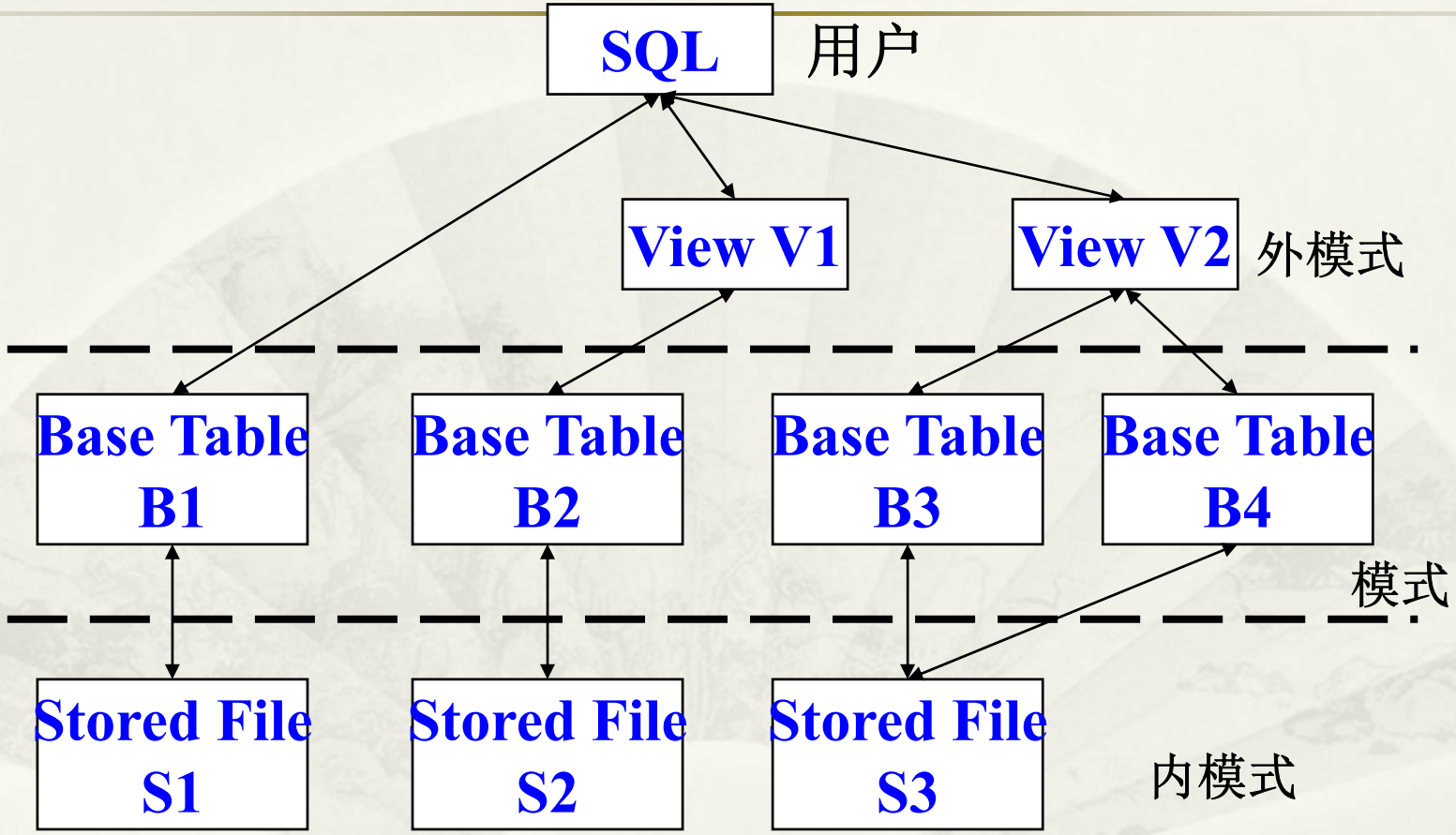
4. 具有查询、定义、操纵和控制四种语言一体化的特点。它只向用户提供一种语言，但该语言具有上述多种功能，可独立完成数据库生命周期中的全部活动。

1.2 SQL的特点 （续）

5. 语言简洁、易学易用：核心功能只有9个动词，语法简单，接近英语。

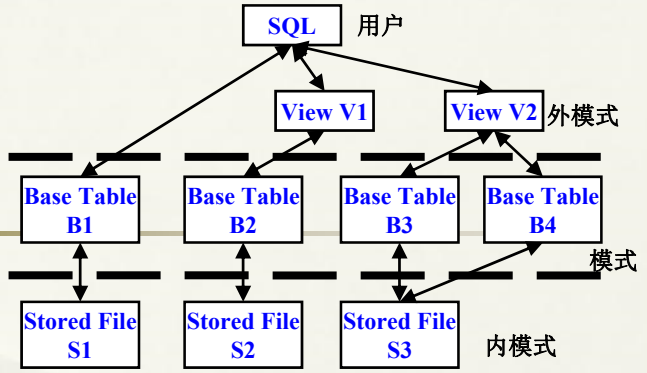
SQL功能	动词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

1.3 SQL基本概念



SQL语言支持的关系数据库的三级模式结构

1.3 基本概念 - 视图、基本表



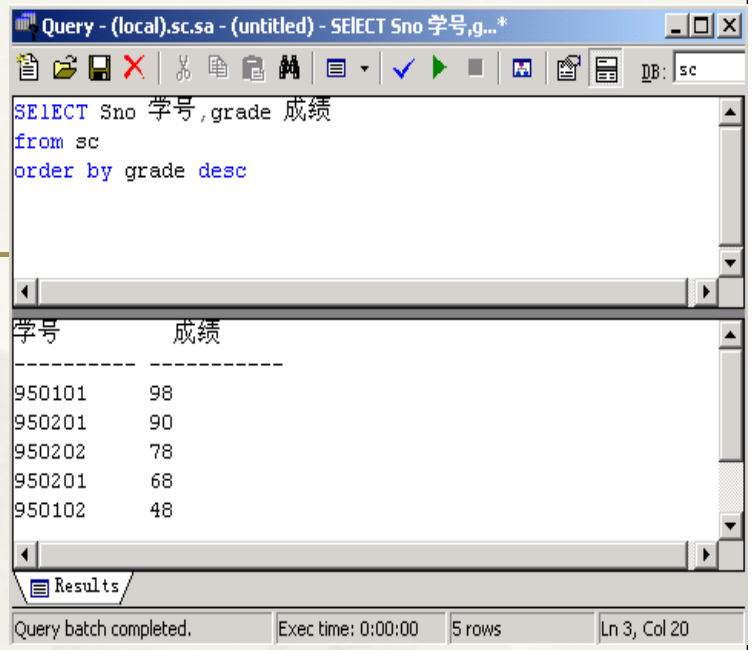
- 用户可以用SQL语言对视图(View)和基本表(Base Table)进行查询等操作，在用户观点里，视图和表一样，都是关系。
- 视图是从一个或多个基本表中导出的表，本身不存储在数据库中，只有其定义，可以将其理解为一个虚表。
- 基本表是本身独立存在的表，一个（或多个）基本表对应一个存储文件，一个表可以带若干索引，存储文件及索引组成了关系数据库的内模式。

1.4 SQL的功能

SQL功能	动词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

- 数据查询语言（DQL）
- 数据定义语言（DDL）
  - 定义、删除模式（Schema）
  - 定义、删除视图（View）
  - 定义、删除索引（Index）
  - 定义、删除、修改基本表（Base Table）
- 数据操纵语言（DML）
  - 数据增、删、改
- 数据控制语言（DCL）
  - 用户访问权限的授予、收回

1.5 SQL的应用形式



- 交互式SQL
  - 联机交互工具
  - 用户可直接键入SQL命令对数据库进行操作
  - 由DBMS来进行解释
- 嵌入式SQL
  - 将SQL语句嵌入到高级语言（宿主语言）
  - 使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力。
  - 一般需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句。

关系数据库：数据类型、函数

1、字符型：**CHAR(n), VARCHAR(n)**

2、数字型：**INT, SMALLINT, REAL,…**

3、日期型：**DATE, TIME**

1、数字函数：**ABS(X), SQRT(X), LOG(X), …**

2、字符函数：**LOWER(X), UPPER(X), …**

3、聚集函数：**COUNT(*), MAX(X), MIN(X), AVG(X), SUM(X), …**

2 数据查询

数据查询是数据库应用的核心功能

- 基本结构

Select $A_1, A_2, \ldots, A_n$
From $R_1, R_2, \ldots, R_m$
Where P

$$\pi_{A_1,A_2,\cdots,A_n}\left(\sigma_P\left(R_1 \times R_2 \times \cdots \times R_m\right)\right)$$

Select **Where** **From**

2.1 数据查询 - 语句格式

- SELECT [ALL|DISTINCT] [表名.]<*|列名|表达式 [AS 新列名]>
[INTO :主变量1[,:主变量2]…]
FROM <表名或视图名> …
[WHERE 条件表达式|子查询]
[GROUP BY 列名1, …[HAVING 分组表达式]]
[{UNION|INTERSECT |EXCEPT}SELECT…FROM…]
[ORDER BY 列名|列序号[ASC|DESC], …];

2.1 数据查询 - 语句执行过程

SELECT…	⑤	投影
FROM…	①	TABLE→内存
WHERE…	②	选取元组
GROUP…	③	分组
HAVING…	④	选择分组
[{UNION|… }	⑥	查询结果的集合运算
SELECT…]	①～⑤	
ORDER BY……	⑦	排序输出

学生课程数据库

* **该数据库模式中包含三个基本表**

1. **学生表**

   S（S#, SN, SA, SD），Primary Key: S#

2. **课程表**

   C（C#, CN, PC#），Primary Key: C#

3. **学生选课表**

   SC（S#, C#, GR），Primary Key: (S#, C#)

2.2 数据查询 - 单表查询

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

1. 简单查询

- 求学生所在系及姓名
- 求学生的全部信息
- 查询全体学生的姓名、出生年份和所在系，要求用小写字母表示所有系名。

```
SELECT   SN,  'Year of Birth:',
         2020-SA,LOWER(SD)
FROM     S;
```

2.2 数据查询 - 单表查询

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求选修了课程的学生号（消除重复行：DISTINCT）

```
SELECT    DISTINCT   S#
FROM      SC;
```

2.2 数据查询 - 单表查询

2. Where定义的复杂查询

Where 子句——运算符

- 比较：<、<=、>、>=、=、<>、NOT + *前面符号*
- 确定范围：

  Between A and B、Not Between A and B
- 确定集合：IN、NOT IN
- 字符匹配：LIKE，NOT LIKE
- 空值：IS NULL、IS NOT NULL
- 多重条件：AND、OR、NOT

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求广电工系年龄小于19的学生姓名及年龄

```
SELECT    SN, SA
FROM      S
WHERE     SD= 'GDG'    AND   SA<19;
```

例：　求年龄在18～22　(含18，22)之间的学生姓名及年龄（或不在18～22之间）

```
SELECT   SN, SA
FROM     S
WHERE    SA BETWEEN 18   AND   22;
(WHERE   SA>=18   AND   SA<=22; )
(WHERE SA NOT BETWEEN 18 AND 22; )
(WHERE   SA<18   OR    SA>22; )
```

S	S#, SN, SA, SD
C	C#, CN, PC#
SC	S#, C#, GR

查询条件来自集合：

用IN or NOT IN

例：求广电工系、通信系、数媒系的系名、学生名 (或不是这些系的学生)

```
SELECT    SD,   SN
   FROM      S
WHERE     SD   IN ( 'GDG' ,  'TX' ,  'SM' );
   (WHERE SD= 'GDG'  OR SD= 'SM'  OR SD= 'TX' ;)

(WHERE SD NOT IN ( 'GDG' ,  'TX' ,  'SM' );
(WHERE SD!= 'GDG'  AND SD!= 'TX'  AND
 SD!= 'SM' ;)
```

字符匹配：

Where 子句——Like

格式：

[NOT] LIKE ‘匹配串’ [ESCAPE ‘换码字符’]

- **%**：表示任意长度（≥0）的任意字符
- **_**：表示单个的任意字符
- **ESCAPE ‘换码字符’ ：**

  匹配串中 ‘换码字符’ (转义符)之后的字符(%, _)，被定义为普通字符(不作通配符用)

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

* 例：列出课程名称中带有‘_’的课号及课名

```
SELECT C#, CN
FROM   C
WHERE CN LIKE  ‘%\_%’  ESCAPE  ‘\’ ;
```

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求课程名中有‘数据库’的课程记录

```
SELECT    *
FROM   C
WHERE   CN   LIKE    ‘%数据库%’；
```

例：求倒数第三、四个汉字为‘系统’的课程名

```
SELECT   CN
FROM   C
WHERE   CN   LIKE    ‘%系统_ _ _ _’；
```

注：数据库字符集为ASCII时，一个汉字需要两个字符表示。

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求以‘DATA_BASE’开头且倒数第五个字符为’S’的课程名

```
SELECT  CN
FROM  C
WHERE CN LIKE
        ‘DATA\_BASE%S_ _ _ _’
ESCAPE’\’;
```

涉及空值的查询:

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：查缺少成绩的学生的学号和相应的课程号

```
SELECT    S#, C#
FROM      SC
WHERE     GR IS NULL;
```

2.2 数据查询 - 单表查询

3. 分组与组函数

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

GROUP BY 子句

* 将查询结果集按某一列或多列的值分组，值相等的为一组，一个分组以一个元组的形式出现。

例：统计各系学生的人数

```
SELECT SD, COUNT(*) AS STU_COUNT
FROM   S
GROUP BY SD;
```

组函数/聚集函数:

1) 组函数的使用格式:

   * COUNT([DISTINCT|ALL] *|列名)
   * SUM([DISTINCT|ALL] 列名)
   * AVG([DISTINCT|ALL] 列名)
   * MAX([DISTINCT|ALL] 列名)
   * MIN([DISTINCT|ALL] 列名)

2) 组函数可用于SELECT子句中的目标列表中，或在HAVING子句的分组表达式中用作条件。

3) 对分出的每一组用HAVING进行筛选,筛选条件要用到组函数。

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：查询各课程号与相应选课人数

```
SELECT    C#,  COUNT(S#)
FROM      SC
GROUP     BY   C#;
```

例：求选修课程在5门以上且都及格的学号及总平均分

```
SELECT   S#,    AVG(GR)
FROM   SC
GROUP   BY   S#
HAVING   COUNT(*)>5
                AND MIN(GR)≥60;
```

Having 与 Where的区别

- Where 决定哪些元组被选择参加运算，作用于关系中的元组。
- Having 决定哪些分组符合要求，作用于分组。

2.2 数据查询 - 单表查询

4、排序

1) 用ORDER BY子句对查询结果按照一个或多个列的值进行升/降排列输出。

2) 升序为ASC；降序为DESC 。

3) 空值将作为最大值排序。

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例:对选修‘C5’课程的学生按成绩降序排列，同分数者按学号升序排列。

```
SELECT   S#, GR
FROM     SC
WHERE    C#= ‘C5’
ORDER    BY GR DESC, S# ASC;
```

2.3 数据查询 - 连接查询

- 单表查询
- 连接查询

  多表连接查询、自身连接查询、

  外连接查询、复合条件连接查询.
- 嵌套查询

  返回单个值的子查询

  返回一组值的子查询

  多重子查询

2.3 连接查询

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

多表连接查询、自身连接查询、
复合条件连接查询

2.3.1 多表连接

1) 连接条件一：

[表名1.] 列名1 比较运算符 [表名2.]列名2

2) 连接条件二：

[表名1.]列名1 BETWEEN [表名2.]列名2
AND [表名2.]列名3

执行过程：

在表1中找到第一个元组，然后从头开始扫描表2，查找到满足条件的元组即进行串接并存入结果表中；再继续扫描表2，依次类推，直到表2末尾。再从表1中取第二个元组，重复上述的操作，直到表1中的元组全部处理完毕。

S

S#	SN	SD	SA
S1	A	GDG	20
S2	B	GDG	21
S3	C	SM	19
S4	D	TX	19
..	..	..	..

SC

S#	C#	GR
S1	C1	A
S1	C2	A
S1	C3	A
S1	C5	B
S2	C1	B
S2	C2	C
..	..	..

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求选课情况，要求输出学号、姓名、课程名与成绩。

```
SELECT  S.S#，SN，CN，GR
FROM S，C，SC
WHERE S.S#=SC.S# AND C.C#=SC.C#;
```

2.3.2 自身连接

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

用表别名把一个表定义为两个不同的表进行连接。

例：求每门课的间接先修课（即先修课的先修课）

```
SELECT   FIRST.C#, SECOND.PC#
FROM     C FIRST,  C SECOND
WHERE    FIRST.PC# = SECOND.C#;
```

2.3.3 复合条件连接

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

WHERE子句中除了连接条件，还有其它限制条件。

例：求选修‘C6’课程且成绩超过90分的学生姓名与成绩

```
SELECT SN, GR
FROM   S, SC
WHERE  S.S#=SC.S# AND SC.C#= ‘C6’
                  AND SC.GR>90;
```

连接条件

限制条件

2.4 数据查询 - 嵌套查询

- 在SELECT … FROM … WHERE语句结构的WHERE子句中可嵌入一个SELECT语句块。
  - 上层查询称为外层查询或父查询
  - 下层查询称为内层查询或子查询
- SQL语言允许使用多重嵌套查询
  - 在子查询中不允许使用ORDER BY子句
- 嵌套查询的实现一般是从里到外，即先进行子查询，再把其结果用于父查询作为条件。

2.4.1 返回单个值的子查询

例：求与‘刘力’同一个系学生名，年龄

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

方法一：

```
SELECT   SN,    SA
 FROM     S
 WHERE    SD =
          ( SELECT     SD
            FROM       S
           WHERE     SN = ‘刘力’ ）；
```

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求与‘刘力’同一个系的学生姓名和年龄

方法二：

```
SELECT    FIRST.SN,   FIRST.SA
FROM      S FIRST , S SECOND
WHERE     FIRST.SD = SECOND.SD
          AND SECOND.SN = ‘刘力’;
```

2.4.2 返回一组值的子查询:

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：求选修‘C6’课程且成绩超过90分的学生信息

方法一：

```
SELECT   *
FROM     S
WHERE    S#   IN
         ( SELECT     S#
           FROM       SC
           WHERE      C#='C6'
                AND   GR>90 );
```

例：求选修‘C6‘课程且成绩超过90分的学生

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

方法二（连接查询）：

```
SELECT    S.*
FROM      S，SC
WHERE     S.S# = SC.S#
          AND   GR＞90
          AND   C# = ‘C6’;
```

2.4.3 多重子查询

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：查询选修了课程名为“信息系统”的学生学号和姓名

```
SELECT   S#,   SN
FROM     S
WHERE    S# IN
      (SELECT    S#
       FROM      SC
       WHERE     C# IN
          (SELECT   C#
           FROM     C
           WHERE    CN = ‘信息系统’));
```

2.5 数据查询 – SQL集合操作

集合操作注意事项：

- 属性个数必须一致
- 对应的类型必须一致
- 属性名无关
- 最终结果集采用第一个结果的属性名
- 缺省为自动去除重复元组
  - 除非显式说明**ALL**
- **ORDER BY**放在整个语句的最后

SQL的集合操作——并

例：查询广电工系的学生或者年龄不大于19岁的学生，并按年龄倒排序。

```
SELECT   *
FROM     S
WHERE    SD='GDG'
UNION
SELECT   *
FROM     S
WHERE    SA<=19
ORDER BY SA DESC;
```

SQL的集合操作——交

例：查询广电工系的学生并且年龄不大于19岁的学生，并按年龄倒排序。

```
(SELECT   *
 FROM     S
 WHERE    SD='GDG')
INTERSECT
(SELECT   *
 FROM     S
 WHERE    SA<=19)
ORDER BY SA DESC;
```

SQL的集合操作——差

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：查询选修课程1但没有选修课程2的学生

```
SELECT      *
FROM        S
WHERE       S#    IN
        (( SELECT   S#
           FROM     SC
           WHERE    C#= '1' )
         EXCEPT
          ( SELECT S#
           FROM     SC
           WHERE    C#= '2'));
```

2.6 带有EXISTS谓词的子查询

- EXISTS代表存在量词 ∃。带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值true或假值false。
- 查询选修了1号课程的学生姓名

```
SELECT SN
    FROM S
    WHERE EXISTS
        (SELECT * FROM SC
            WHERE S#=S.S# AND C#='1');
```

查询没有选修1号课程的学生姓名

```
SELECT  SN
    FROM S
      WHERE NOT EXISTS
        （SELECT *
          FROM SC
          WHERE S#=S.S# AND C#=‘1’）；
```

* 也可以通过其他谓词查询，如IN。

查询选修了所有课程的学生姓名

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

```
SELECT SN
FROM S
WHERE NOT EXISTS
    ( SELECT *
        FROM C
        WHERE NOT EXISTS
        ( SELECT *
                FROM SC
                WHERE S#=S.S# AND C#=C.C#));
```

查询至少选修了3号学生选修的全部课程的学生学号

- 查询条件可转换为：查询学号为x的学生，对所有课程y，只要3号学生选修了，则x号学生也必然选修。
- P表示谓词：3号学生选修了课程y；
- Q表示谓词：x号学生选修了课程y；
- 即 $(\forall y)P \rightarrow Q$

$$(\forall y)P \rightarrow Q \equiv \neg\exists y(P \wedge \neg Q)$$

查询至少选修了3号学生选修的全部课程的学生学号

$$(\forall y)P \rightarrow Q \equiv \neg\exists y(P \wedge \neg Q)$$

```
SELECT S#
FROM SC SCX
WHERE NOT EXISTS
        ( SELECT * FROM SC SCY
                WHERE SCY.S#='3' AND
                        NOT EXISTS
                        ( SELECT * FROM SC SCZ
                          WHERE SCZ.S#=SCX.S# AND
                                SCZ.C#=SCY.C#));
```

3 数据操纵

一、插入操作　　INSERT

二、删除操作　　DELETE

三、修改操作　　UPDATE

3.1 插入操作

1、插入单个元组：

格式：INSERT INTO <表名>[(列名1,...)]

VALUES (列值1,...);

- 插入一已知元组的全部列值
- 插入一已知元组的部分列值

- **插入一已知元组的全部列值**

S	S#，SN，SA，SD
C	C#，CN，PC#
SC	S#，C#，GR

例：新增一个学生信息

```
INSERT   INTO    S
VALUES('990021','陈冬' ,18,'GDG');
```

- **插入一已知元组的部分列值**

例：新增一条选课记录

```
INSERT   INTO    SC（S#，C#）
VALUES  ('9807121','C175');
```

3.1 插入操作

2、插入子查询的结果：

格式：INSERT INTO <表名> [(列名1,...)]
(子查询);

例：设关系S_G(S#,AVG_G),请将平均成绩大于80的广电工系学生的学号及平均成绩存入S_G中。

```
INSERT INTO    S_G(S#, AVG_G)
 (SELECT   S#,   AVG(GR)
   FROM      SC
   WHERE     S#   IN
      (SELECT   S#
       FROM      S
       WHERE     SD= ‘GDG’ )
   GROUP   BY   S#
   HAVING   AVG(GR)>80);
```

设关系S_G(S#,AVG_G),请将平均成绩大于80的广电工系学生的学号及平均成绩存入S_G中。

3.2 删除操作

格式：　DELETE　FROM　〈表名〉
　　[WHERE　条件]；

- 只能对整个元组操作，不能只删除某些属性上的值
- 只能对一个关系起作用，若要从多个关系中删除元组，则必须对每个关系分别执行删除命令
- 从关系 *r* 中删除满足**P**的元组，只是删除数据，而不是定义

3.2 删除操作

1、删除单个和多个元组：

例：从选课系统删除学号为‘95019’的学生

```
DELETE   FROM    SC
WHERE    S# =   ‘95019’;

DELETE   FROM S
WHERE    S# =   ‘95019’;
```

3.2 删除操作

删除多个元组：

例：删除选课但无成绩的学生的选课信息

```
DELETE  FROM SC
WHERE   GR IS NULL;

DELETE  FROM SC;          //清空SC表
```

3.2 删除操作

3、带子查询的删除语句:

例：删除选修‘C4’且成绩小于该课程的平均成绩的记录

```
DELETE  FROM  SC
WHERE  C#='C4'  AND  GR <
        (SELECT AVG(GR)
         FROM  SC
         WHERE  C#='C4');
```

3.3 修改操作

格式1：UPDATE　〈表名〉　[别名]
　　SET　　〈列名〉=〈表达式〉，…
　　[WHERE　条件]；

格式2：UPDATE　〈表名〉　[别名]
　　SET　　〈列名,…〉=〈子查询〉
　　[WHERE　条件]；

3.3 修改操作

1、修改单个和多个元组的值：

例：将所有学生的年龄增加1岁

```
UPDATE S
SET    SA=SA+1;
```

2、带子查询的修改语句:

例：广电工系全体学生成绩上浮15%

```
UPDATE  SC
SET     GR=GR*1.15
WHERE   S#  IN
        (SELECT  S#
         FROM  S WHERE  SD='GDG');
```

4 数据定义

一、定义基本表　　CREATE　TABLE

二、删除基本表　　DROP　TABLE

三、修改基本表　　ALTER　TABLE

四、定义索引　　CREATE　INDEX

五、删除索引　　DROP　INDEX

4.1 定义基本表

格式1：　CREATE TABLE <表名>
(列名 类型(长度)　[NOT　NULL]
[DEFAULT {常量|系统变量|NULL}]
[列约束],... ...)
[PRIMARY KEY (列名,...)]
[FOREIGN KEY (列名,...)
　　REFERENCES　表名(列名,...)]
[CHECK　条件];

- 建立一个新表，表中无记录

4.1 定义基本表

格式2：　CREATE TABLE 〈表名〉

[(列名　[NOT　NULL],...)]

[PRIMARY KEY (列名,...)]

[FOREIGN KEY (列名,...)

REFERENCES 表名1(列名,...)]

AS　子查询;

- 建立一个带有子查询结果记录的新表。

例：建立学生S、课程C、选课SC三个表

S表：

```
CREATE  TABLE  S
      ( S#  CHAR(6)  NOT  NULL,
        SN  CHAR(8)  NOT  NULL,
        SA  SMALLINT,  DEFAULT   18,
        SD  CHAR(10));
   PRIMARY   KEY   (S#);
```

例：建立学生S、课程C、选课SC三个表

C表：

```
CREATE  TABLE  C
        (C# CHAR(6) NOT NULL,
         CN CHAR(30) NOT NULL,
         PC#  CHAR(6));
PRIMARY  KEY  (C#);
```

例：建立学生S、课程C、选课SC三个表

SC表：

```
CREATE  TABLE  SC
        (S#   CHAR(6)   NOT   NULL,
         C#   CHAR(6)   NOT   NULL,
         GR   SMALLINT DEFAULT NULL)
PRIMARY   KEY (S#,C#),
FOREIGN KEY (S#) REFERENCES S(S#),
FOREIGN KEY (C#) REFERENCES C(C#),
CHECK (GR IS NULL)
                OR (GR BETWEEN 0 AND 100);
```

例：设关系S_G(S#,AVG_G),把平均成绩大于80的广电工系学生的学号及平均成绩存入S_G中。

```
CREATE  TABLE  S_G
 (S#  CHAR(6)  NOT  NULL,
  AVG_G  SMALLINT DEFAULT NULL)
   AS
   (SELECT  S#,  AVG(GR)
      FROM     SC
      WHERE    S#  IN
        (SELECT  S#
         FROM     S
         WHERE    SD='GDG' )
      GROUP  BY  S#
      HAVING  AVG(GR)>80);
```

4.2 删除基本表

格式： DROP TABLE <表名>

[CASCADE|RESTRICT];

- CASCADE 连同引用该表的视图、完整性约束一起自动撤消。
- RESTRICT无引用时，才可撤消。

- 删除一个表，及与该表相关的索引、视图、码和外部码。

4.3 修改基本表

- 基本表的修改操作：
  - 增加列
  - 改变列的数据类型
  - 删除列的约束
  - 删除列
  - 改变列名
  - ……

4.3 修改基本表

格式：ALTER TABLE <表名>

[ADD <新列名><类型(长度)>[NOT NULL]

[列约束],...]

[ALTER COLUMN <列名><类型(长度)>]

[DROP <列约束>];

- 增加新列，修改列，删除列的完整性约束；
  改变列名或数据类型。

例：在S表中增加一个入学时间，为日期型

```
ALETR TABLE S ADD SCOME DATE;
```

例：把SA列的类型改为短整型

```
ALETR TABLE S ALTER COLUMN SA SMALLINT;
```

例：删除对SN列的唯一约束

```
ALETR TABLE S DROP UNIQUE(SN);
```

4.4 定义索引

格式：

CREATE[UNIQUE][CLUSTER] INDEX <索引名>
ON <表名> (列名[ASC|DESC],...);

- UNIQUE 表示索引值唯一
- CLUSTER 表示索引是聚簇索引
- 索引一旦建立，交由系统使用和维护

- 对指定的表的列建立索引

- CREATE UNIQUE INDEX SIDX ON S(S# ASC);
- CREATE CLUSTER INDEX SNCDX
  ON S(SN ASC);
- CREATE UNIQUE INDEX CIDX ON C(C# ASC);
- CREATE UNIQUE INDEX SCIDX
  ON SC(S# ASC,C# DESC);

4.5 删除索引

格式：

DROP INDEX <索引名>;

例： DROP INDEX CIDX;

5 视图

视图：从一个或几个表(或视图)导出的一个特殊的表。

- 视图是一个虚表
- 数据库中只存放视图的定义
- 视图对应的数据仍存放在原来的表中
- 随着表中数据的变化，视图的数据随之改变
- 对视图的查询与基本表一样
- 对视图的更新将受到一定的限制

视图

一、视图概念

二、定义、删除视图

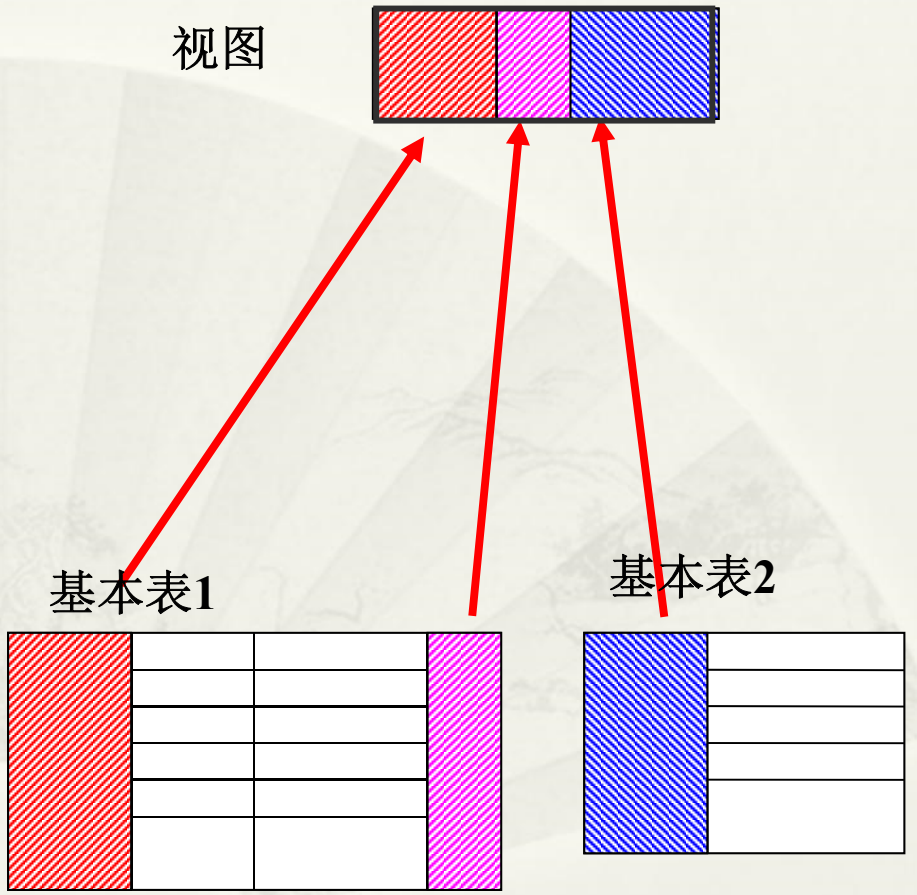
三、查询视图

四、更新视图

五、视图的作用

5.1 视图概念

- 视图是一个虚表
- 数据库中只存放视图的定义
- 视图对应的数据仍存放在原来的表中
- 随着表中数据的变化，视图的数据随之改变。
- 对视图的查询与基本表一样
- 对视图的更新将受到一定的限制



视图概念示意图

5.1 建立视图

下述必须指定全部列名：

- 某个目标列是组函数或表达式
- 多表连接时，目标列中出现同名列
- 不用原基本表的列名作为视图名

格式：CREATE VIEW <视图名>[(列名,...)]

AS <子查询>

[WITH {CHECK OPTION|READ ONLY}];

视图内容的修改需满足子查询条件

- 在定义视图时要么指定全部视图列，要么全部省略不写；如果省略了视图的属性列名，则视图的列名与子查询列名相同。
- 虚拟列：经过各种计算派生出的数据所设置的派生属性列。

5.1 建立视图

视图分类

行列子集视图：从单个基本表导出，保留基本表的码，但去掉其它的某些列和部分行的视图。

表达式视图：带虚拟列的视图。

分组视图：子查询目标表带有组函数或子查询带有**GROUP BY**子句的视图。

例：建立广电工系学生视图

```
CREATE VIEW GDG_S
AS (SELECT S#, SN, SA
      FROM S
      WHERE SD='GDG')
WITH CHECK OPTION;
```

（行列子集视图）

例：建立广电工系选修‘计算机体系结构’课程的学生视图

```
CREATE VIEW   GDG_SCA(S#,SN,GR)
AS SELECT S.S#,SN,GR
      FROM S,SC,C
      WHERE S.S#=SC.S#
         AND C.C#=SC.C#
         AND S.SD='GDG'
         AND CN='计算机体系结构';
```

例：建立学生出生年份的视图

```
CREATE  VIEW BT_S(S#,SN,BIRTH)
AS
 SELECT S#,SN,2020-SA
   FROM  S;
```

（表达式视图）

（虚拟列）

例：建立学生平均成绩视图

```
CREATE VIEW S_AVG_G(S#, AVG_G)
    AS   SELECT S#, AVG(GR)
         FROM   SC
         GROUP BY S#;
```

（分组视图）

5.2 视图查询

视图消解（View Resolution）

在对视图查询时，DBMS将进行有效性检查（表及视图）。若存在，则从数据字典中取出视图定义，并把定义中的子查询与用户查询结合起来转换为等价的对基本表的查询，然后再执行。

例：求广电工系年龄小于20的学生

```
SELECT S#, SN
FROM GDG_S
WHERE SA<20;
```

视图消解

```
SELECT  S#, SN
FROM S
WHERE SD='GDG' AND SA<20;
```

例：求广电工系选修 ‘C2’课程的学生

```
SELECT S#,SN
FROM GDG_S,SC
WHERE GDG_S.S#=SC.S#
   AND SC.C#=‘C2’;
```

例： 在学生平均成绩视图S_AVG_G中查询平均成绩在90分以上的学生号及成绩

```
SELECT *
FROM S_AVG_G
WHERE AVG_G>=90;
```

（系统转换后）

```
SELECT S#, AVG(GR)
FROM SC
GROUP BY S#
HAVING AVG(GR)>=90;
```

5.3 更新视图

1、更新视图即通过视图插入(**INSERT**)、删除(**DELETE**)和修改(**UPDATE**)数据，实质上转换为对基本表的更新。

2、为了防止用户对超出视图范围的基本表的数据进行操作，在定义视图时，应加上**WITH CHECK OPTION**子句，则在视图上更新数据时，**DBMS**将检查视图定义中的条件，不满足将拒绝执行。

例：GDG_S视图的‘刘茜’的年龄改为20

转换前：

```
UPDATE GDG_S
SET SA=20
WHERE SN=‘刘茜’；
```

转换后：

```
UPDATE S
SET SA=20
WHERE SN=‘刘茜’
    AND SD=‘GDG’;
```

例：在GDG_S中插入 ‘990075,吴迪,’19’的学生记录

转换前：

```
INSERT INT GDG_S
VALUES(‘990075’,‘吴迪’ ,19);
```

转换后：

```
INSERT INTO S
VALUES(‘990075’,‘吴迪’ ,19,‘GDG’);
```

例：删除GDG_S中年龄大于23的学生

转换前：

```
DELETE FROM GDG_S
WHERE SA>23;
```

转换后：

```
DELETE FROM S
WHERE SA>23
   AND SD='GDG';
```

例：修改 ‘990075’学生平均成绩为90

转换前：

不可转换

左边程序操作失败

```
UPDATE S_AVG_G
SET AVG_G=90
WHERE SNO='990075';
```

3、一般情况下，行列子集视图是可更新的，所以各RDBS均只允许对行列子集视图进行更新。

4、不可更新的视图（各系统不太一致）

- 由多个表导出的视图，不可更新
- 视图的列来自表达式或常数，不可插、改、可删
- 视图列是来自组函数，不可更新
- 视图定义中含有GROUP BY子句，不可更新
- 视图定义中内层嵌套的表与查询目标同一个表，不可更新
- 在不允许更新的视图上定义的视图，不可更新

5.4 视图的优点

- 提供数据的逻辑独立性
- 提供数据的安全保护功能
- 简化用户的操作

（对系统构成的视图，用户不必关心各表间的联系）

- 同一数据多种用法

小结

- SQL概述
- 数据查询
- 数据操纵
- 数据定义
- 视图

作业

* P.130　4、5、6