关系数据库语言SQL’92

关系数据库语言SQL'92

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图
6. DB2 SQL

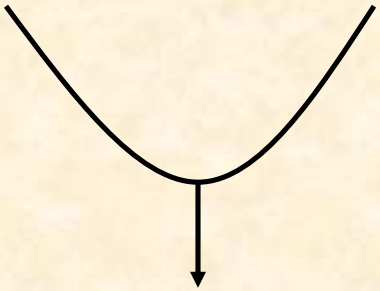
1. SQL概貌

- SQL history
  - SEQUEL（IBM, 1970）
  - SQL-86（SQL1）（ANSI, 1986）
  - SQL-89（ANSI/ISO, 1989）
    - SQL-92（SQL2）（ANSI/ISO, 1992）
  - SQL-93（ANSI/ISO, 1993）
    - SQL-99（SQL3）（ANSI/ISO, 1999）
  - SQL-2000
  - SQL-2003

1. SQL概貌

❑ 基本概念的变化

关系模型	关系	关系子模式	属性	元组
SQL	基表 （base table）	视图 （view）	列 （column）	行 （row）



视图也被称为‘虚表’
（virtual table）

两者统称为‘表’
（table）

1. SQL概貌

- SQL语言的两种使用方式
  - 自含式
    - 独立的交互式命令行语言
  - 嵌入式
    - 嵌入到某种高级程序设计语言（被称为‘主语言’）中使用
    - 嵌入方式
      - 嵌入式SQL（ESQL）
      - 函数调用

1. SQL概貌

- SQL的功能
  - 数据定义功能
    - 基表/视图的定义与删除
    - 索引/集簇的定义与删除
  - 数据操纵功能
    - 数据的查询/插入/删除/修改
    - 简单的数值计算与统计功能
  - 数据控制功能
    - 数据的完整性、安全性、并发控制、故障恢复等
  - 数据交换功能
    - 会话、连接、游标、诊断、动态SQL
  - 扩展功能

1. SQL概貌

❑ SQL的特点

- ➢ 综合统一
  - – 集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动
- ➢ 高度非过程化
  - – 用SQL进行数据操作，只要提出“做什么”，而无须指明“怎么做”
- ➢ 面向集合的操作方式
- ➢ 以同一种语法结构提供多种使用方式
  - – SQL既是独立的语言，又是嵌入式语言
- ➢ 语言简洁，易学易用

关系数据库系统数据子语言SQL

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图

2. SQL数据定义功能

- 在本节我们只介绍简单的基表创建与删除功能
  - 其它的数据定义功能我们放在以后的相关章节中再去介绍
  - 本节内容
    - SQL数据类型
    - 基表的创建
    - 基表的修改
    - 基表的删除

2. SQL数据定义功能

❑ SQL基本数据类型

	符　　号	数据类型
1	INT	整数
2	SMALLINT	短整数
3	DEC(m, n)	十进制数
4	FLOAT	浮点数
5	CHAR(n)	定长字符串
6	VARCHAR(n)	变长字符串
7	BIT(n)	定长位串
8	BIT VARYING(n)	变长位串
9	DATE	日期
10	TIME	时间
11	TIMESTAMP	时间戳

2. SQL数据定义功能

- 小整型，SMALLINT
  - 小整型是两个字节的整数，精度为5位。小整型的范围是从-32,768到32,767
- 大整型，INTEGER或INT
  - 大整型是四个字节的整数，精度为10位。大整型的范围是从-2,147,483,648到2,147,483,647
- 巨整型，BIGINT
  - 巨整型是八个字节的整型，精度为19位。巨整型的范围是从-9,223,372,036,854,755,808到9,223,372,036,854,775,807

2. SQL数据定义功能

- 小数，DECIMAL(p, s)、DEC(p, s)、NUMERIC(p, s)或NUM(p, s)
  - 小数类型的值是一种紧凑的十进制数，它有一个隐含的小数点。紧凑十进制数将以千变万化的二十一进制编码（binary-coded decimal，BCD）记法来存储
  - 小数点的位置取决于数字的精度（p）和小数位（s）
    - 小数位是指数字的小数部分的位数，它不可以是负数，也不能大于精度
    - 最大精度是31位
    - 十进制数的范围是从-10**31+1到10**31-1

2. SQL数据定义功能

- 字符大对象字符串，CLOB(n[K|M|G])
  - 字符大对象的长度可变的字符串，最长可以达到2,147,483,647字节
  - 只要指定了n，那么n的值就是最大长度
  - 如果指定了nK，那么最大长度就是n*1024（n的最大值为2,097,152）
  - 如果指定了nM，那么最大长度就是n*1,048,576（n的最大值为2048）
  - 如果指定了nG，那么最大长度就是n*1,073,741,824（n的最大值为2）
  - CLOB用与存储基于大单字节字符集字符的数据或基于混合（与多字节字符集）字符的数据

图形字符串

- 图形字符串是代表双字节字符数据的字节序列。图形字符串包括类型为GRAPHIC(n)的定长图形字符串和类型为VARGRAPHIC(n)、LONG VARGRAPHIC和DBCLOB(n)的变长图形字符串。字符串的长度就是序列中双字节字符的数目
  - 定长图形字符串
    - 定长图形字符串的长度介于1到127个双字节字符之间。如果没有指定长度，就认为是1个字节
  - 变长图形字符串
    - VARGRAPHIC(n)类型的字符串是变长图形字符串，最大长度可达16,336个双字节字符

图形字符串

- **LONG VARGRAPHIC**
  - **LONG VARGRAPHIC类型的字符串是变长图形字符串，最大长度可达16,350个双字节字符**
- 双字节字符大对象字符串
  - 双字节字符大对象是变长双字节字符图形字符串，最大可达1,073,741,823字符
  - **DBCLOB用于存储基于大DBCS字符的数据**

二进制数据类型

- 二进制串是字节序列
- 二进制串包括可变长度的BLOB(n)类型，它用于容纳非传统型的数据，诸如图片、语音或混合媒体等，还可以容纳用户定义类型及函数的结构化数据
- 二进制大对象，BLOB(n[K|M|G])：最长可达2,147,483,647字节

日期时间数据类型

- DATE是一个由三部分组成的值（年、月和日）。年份部分的范围是从0001到9999。月份部分的范围是从1到12。日部分的范围是从1到n，n的值取决于月份。DATE列长10个字节
- TIME是一个由三部分组成的值（小时、分和秒）。小时部分的范围是从0到24。分和秒部分的范围都是从0到59。如果小时为24，分和秒的值都是0。TIME列长8个字节
- TIMESTAMP是一个由七部分组成的值（年、月、日、小时、分钟、秒和微秒）。微秒部分的范围是从000000到999999。TIMESTAMP列的长度是26字节

日期时间类型的值的字符串表示

- 日期、时间和时间戳也可以用字符串来表示，CHAR标量函数可用来创建日期时间类型的值的字符串表示
- 日期值的字符串表示是一个以数字开始，长度不少于8个字符的字符串。日期值的月份和日部分中前面的零可以略去
- 时间值的字符串表示是以数字开头，长度不少于4个字符的字符串。时间值的小时部分前面的零可以省略，秒部分可以完全省略
- 时间戳值的字符串表示是以数字开头，长度不少于16个字符的字符串。完整的时间戳字符串表示形式为**yyyy-mm-dd-hh.mm.ss.nnnnnn**。时间戳的月、日或小时等几部分前面的零可省略，微秒可截断或完全省略

2. SQL数据定义功能

❑ 基表的创建命令

```
CREATE  TABLE  tablename (
    colname  datatype  [ NOT NULL ]
    { , colname  datatype  [ NOT NULL ] }
) ;
```

➢ 命令的格式定义符号

1) [...]　/* 0或1个 */
2) { ... }　/* 0到若干个 */
3) CREATE TABLE
4) NOT NULL

2. SQL数据定义功能

【例】创建‘学生’基表S

```
CREATE TABLE S (
    sno CHAR(5) NOT NULL,
    sn CHAR(20),
    sd CHAR(2),
    sa SMALLINT
);
```

2. SQL数据定义功能

【例】创建‘课程’基表C

```
CREATE TABLE C (
    cno CHAR(4) NOT NULL,
    cn CHAR(30),
    pcno CHAR(4)
);
```

2. SQL数据定义功能

【例】创建‘选课’基表SC

```
CREATE TABLE SC (
    sno CHAR(5) NOT NULL,
    cno CHAR(4) NOT NULL,
    g CHAR(1)
);
```

2. SQL数据定义功能

❑ 基表的修改命令

➢ 是对基表定义信息的修改，包括：

1) 对基表结构的修改

- 表中属性的增加/删除

```
ALTER TABLE <基表名> ADD <列名> <数据类型>;
ALTER TABLE <基表名> DROP <列名> ;
```

2) 数据完整性约束的修改

2. SQL数据定义功能

❑ 基表的删除命令

- ➢ 删除指定的表及其数据

```
DROP  TABLE  <基表名>;
```

关系数据库系统数据子语言SQL

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图

3. SQL数据操纵功能

- 数据查询
- 插入、删除、修改
- 作为自含式语言还具有如下功能：
  - ✓赋值功能
  - ✓计算功能
    - 简单的四则运算
    - 统计计算
      - « 求总数
      - « 求和、求平均值
      - « 求最大值、求最小值
    - 分组统计
  - ✓输入/出功能

3. SQL数据操纵功能

❑数据查询功能

SQL语言数据查询功能的数学基础来源于关系代数

➢关系代数

$$\Pi_{A1, A2, ..., Am}(\sigma_F(R_1 \times R_2 \times \cdots\cdots \times R_n))$$

➢SQL语言

SELECT $A_1, A_2, ..., A_m$

FROM $R_1, R_2, \cdots\cdots, R_n$

WHERE F

3. SQL数据操纵功能

- 假设: Head(R) = { A_1, …, A_n, B_1, …, B_k }

  Head(S) = { B_1, …, B_k, C_1, …, C_m }

  - 关系代数

    $\Pi_{A1, A2, ..., Am}(\sigma_F(R \bowtie S))$

  - SQL语言

    SELECT A_1, A_2, …, A_m

    FROM R, S

    WHERE F and $R.B_1=S.B_1$ and $R.B_2=S.B_2$

    and …… and $R.B_k=S.B_k$

3. SQL数据操纵功能

❑映像语句

目 标 子 句：　SELECT * | colname { , colname ... }

范 围 子 句：　FROM tablename { , tablename ... }

条 件 子 句：　[WHERE search_condition]

分 组 子 句：　[GROUP BY colname { , colname ... }

分组查询子句：[HAVING group_condition]]

排序输出子句：[ORDER BY colname [ASC | DESC]

{ , colname [ASC | DESC] ... }];

3. SQL数据操纵功能

- 映像语句各子句的执行顺序

```
SELECT  * | colname { , colname ... }
FROM  tablename { , tablename ... }
[ WHERE  search_condition ]
[ GROUP BY  colname { , colname ... } ]
    [ HAVING  group_condition ] ]
[ ORDER BY  colname [ ASC | DESC ]
        { , colname [ ASC | DESC ] ... } ];
```

3. SQL数据操纵功能

3.1 SQL的基本查询功能

3.2 分层结构查询与集合谓词使用

3.3 SELECT语句间的运算

3.4 SQL计算、统计、分类的功能

3.5 SELECT语句使用的一般规则

3.1 SQL的基本查询功能

- 映像语句的组成

1) 目标子句： SELECT * | colname { , colname ... }

– 定义结果关系所需要的属性

➢ 目标子句的构造方式

– 给出结果属性的属性名

- 可以通过‘表名.属性名’的方式来表明是哪一张表中的属性
- 结果属性的重命名
  - » <column_expression> AS <colname>

– 可用‘*’来代替表中的所有属性

– 可使用保留字‘distinct’来消除结果关系中的重复元组

3.1 SQL的基本查询功能

- 映像语句的组成（cont.）

  2) 范围子句： FROM tablename { , tablename ... }

  – 指定操作对象（被访问的关系）

  - 可以在FROM子句中对一个关系重新命名（即定义一个别名（alias））

    <table_name> <alias_name>

  – 主要用于关系自身的联接运算

‘SELECT子句’和‘FROM子句’是一条映像语句中必不可少的两个组成部分

3.1 SQL的基本查询功能

- 映像语句的组成（cont.）

  3) 条件子句： WHERE search_condition

  – 是映像语句中的可选成分，用于定义查询条件（即结果关系中的元组必须满足的条件）

  – 包括‘单个关系中的元组选择条件’以及‘关系与关系之间的联接条件’都需要在WHERE子句中通过一定的逻辑表达式显式地表示出来。

    - 在FROM子句中给出的关系只是表明此次查询需要访问这些关系，它们之间是通过笛卡儿乘积运算进行合并的
    - 如果需要执行它们之间的‘θ-联接’或‘自然联接’运算，则需要在WHERE子句中显式地给出它们的联接条件

3.1 SQL的基本查询功能

- SQL查询功能的例子

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

(1) 单表简单查询

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.25 查询S的所有情况

【answer 1】

```
SELECT  sno, sn, sd, sa  FROM S;
```

【answer 2】

```
SELECT  *  FROM  S;
```

(1) 单表简单查询

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.26 查询全体学生名单

```
SELECT  sn
FROM  S;
```

(1) 单表简单查询

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.27 查询学号为990137的学生学号与姓名

```
SELECT  sno, sn
FROM  S
WHERE  sno = '990137';
```

- 常用的比较谓词是一些算术比较运算符

(1) 单表简单查询

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.28 查询所有年龄大于20岁的学生的姓名与学号

```
SELECT  sn, sno
FROM  S
WHERE  sa > 20;
```

(2) 常用谓词

- 除常用的算术比较运算符外，SQL语言还提供了若干比较谓词，以增强查询语句的表达能力
  - **DISTINCT**
  - **BETWEEN … AND …**
  - **NOT BETWEEN … AND …**
  - **LIKE**
  - **NOT LIKE**
  - **IS NULL**
  - **IS NOT NULL**

DISTINCT仅用于SELECT子句中，而其它谓词一般用于WHERE子句中，用于构造查询条件

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.29 查询所有选修了课程的学生的学号

```
SELECT  DISTINCT  sno
FROM  SC;
```

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.30 查询年龄在18至21岁（包括18与21岁）的学生姓名与年龄

```
SELECT  sn, sa
FROM  S
WHERE  sa BETWEEN 18 AND 21;
```

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.31 查询年龄不在18至21岁的学生姓名与年龄

```
SELECT  sn, sa
FROM  S
WHERE  sa NOT BETWEEN 18 AND 21;
```

(2) 常用谓词

❑LIKE谓词的使用方法

column [NOT] LIKE val1 [ESCAPE val2]

- **模版（pattern）：val1**
  - 下划线（_）：可以匹配任意一个字符
  - 百分号（%）：可以匹配任意一个字符串（包括长度为0的空字符串）
  - 其它字符：只能匹配其自身
- **转义指示字符：val2**
  - 紧跟在转义指示字符**val2**之后的‘_’或‘%’（包括转义字符自身）不再是通配符，而是其自身

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.32 查询姓名以A开头的学生的姓名与所在系别

```
SELECT  sn, sd
FROM    S
WHERE  sn  LIKE  'A%';
```

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.33 查询姓名以字母A开始，且第三个字符必为P的学生的姓名与系别

```
SELECT  sn, sd
FROM    S
WHERE  sn  LIKE  'A_P%';
```

(2) 常用谓词

❑转义字符使用的例子

例1：查询在课程名中含有下划线（_）的课程的课程号

```
SELECT  cno
FROM  C
WHERE  cn  LIKE  '%A_%'  ESCAPE  'A';
```

例2：查询在课程名中含有百分号（%）的课程的课程号

```
SELECT  cno
FROM  C
WHERE  cn  LIKE  '%A%%'  ESCAPE  'A';
```

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.34 查询无课程分数的选课记录中的学号与课程号

```
SELECT  sno, cno
FROM    SC
WHERE   g  IS  NULL;
```

(2) 常用谓词

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例：查询预修课程号不为空的课程的课程号

```
SELECT  cno
FROM    C
WHERE   pno  IS  NOT  NULL;
```

(3) 布尔表达式

- 在WHERE子句中，可以使用NOT、AND与OR这三个逻辑运算符构造出复杂的查询条件，称之为布尔表达式

例3.35 查询计算机系（CS）年龄小于或等于20岁的学生的姓名

```
SELECT  sn
FROM  S
WHERE  sd = 'CS'  AND  sa <= 20;
```

(3) 布尔表达式

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.36 查询非计算机系或年龄不为18岁的学生的姓名

```
SELECT  sn
FROM    S
WHERE   NOT sd = 'CS' OR  NOT sa = 18;
```

(4) 简单连接

- 在WHERE子句中，通过两个属性之间的相等比较实现表与表之间的连接

例3.37 查询修读课程号为C1的所有学生的姓名

```
SELECT  S.sn
FROM    S, SC
WHERE   SC.sno = S.sno AND SC.cno = 'C1';
```

(4) 简单连接

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.38 查询修读课程名为DATABASE的所有学生的姓名

```
SELECT  S.sn
FROM    S, SC, C
WHERE   S.sno = SC.sno AND SC.cno =
        C.cno AND C.cn = 'DATABASE';
```

(5) 自连接

- 在查询中，有时需要对相同的表进行连接。为了区分两张相同的表，必须在FROM子句中至少对其中之一进行换名（即定义别名），以区分开这两张表

(5) 自连接

例3.39 查询至少修读学号为S5的学生所修读的一门课程的学生的学号

- 首先从选课（SC）关系中查到S5所修过课程的课程号，然后再用这些课程号到选课（SC）关系中查出有哪些学生修过其中的课程
  - 因此，这是一个表与表自身进行联接的查询，同一个表名（SC）需要在FROM子句中出现两次
  - 为了区分开它们，必须至少对其中的一个（SC）进行换名

(5) 自连接

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例3.39 查询至少修读学号为S5的学生所修读的一门课程的学生的学号

```
SELECT  SC1.sno
FROM    SC  SC1，SC  SC2
WHERE   SC1.cno = SC2.cno AND
        SC2.sno = ‘S5’；
```

(6) 结果排序

- 有时，希望查询结果能按某种顺序显示，此时须在语句后加一个排序子句ORDER BY，该子句具有下面的形式：

  ORDER BY <列名> [ASC | DESC] { , … }

- 其中：
  - **<列名>给出了所需排序的列的列名**
  - **ASC | DESC则给出了排序的升序ASC或降序DESC，缺省值是升序**

(6) 结果排序

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例0 查询计算机系所有学生名单并按学号顺序升序显示

```
SELECT  sno, sn
FROM    S
WHERE  sd = 'CS'
ORDER  BY  sno  ASC;
```

(6) 结果排序

S (sno, sn, sd, sa)

C (cno, cn, pno)

SC (sno, cno, g)

例1 查询全体学生情况，结果按学生年龄降序排例

```
SELECT *
FROM   S
ORDER BY sa DESC;
```

3. SQL数据操纵功能

3.1 SQL的基本查询功能

3.2 分层结构查询与集合谓词使用

3.3 SELECT语句间的运算

3.4 SQL计算、统计、分类的功能

3.5 SELECT语句使用的一般规则

3.2 分层结构查询与集合谓词使用

- 分层结构指的是在一条映像语句的某个子句中嵌入另一条映像语句，被嵌入的映像语句通常称为‘子查询’

查询 Q_1

```
SELECT ……
FROM ……
WHERE ……
```

子查询 Q_{11}

```
SELECT ……
FROM ……
WHERE ……
```

3.2 分层结构查询与集合谓词使用

- 这样的子查询通常被嵌入在WHERE子句中，可以构造出逻辑关系相对复杂但可以描述清晰的查询条件
- 像这样嵌入有子查询的查询语句也被称为‘嵌套查询’。也可以在其它子句中嵌入‘子查询’
- 由于子查询的查询结果是一个集合，因此需要在WHERE子句中引入集合谓词

3.2 分层结构查询与集合谓词使用

- WHERE子句中的集合谓词主要有：
  - IN谓词：标量与集合量之间的属于比较

    expr [NOT] IN (subquery)

  - 限定比较谓词：标量与集合中元素之间的量化比较

    expr θ ANY|ALL (subquery)

  - EXISTS谓词：是否为空集的判断谓词

    [NOT] EXISTS (subquery)

3.2 分层结构查询与集合谓词使用

❑嵌套查询的处理顺序

- ➢一般情况下，嵌套查询中的子查询只需要被执行一次，然后利用所获得的中间查询结果来计算外层的查询语句
- ➢这样的子查询也被称为‘独立子查询’，其处理顺序由‘内’到‘外’

3.2 分层结构查询与集合谓词使用

- 在有些情况下，在‘子查询’中调用了外层查询中的表及其元组变量。随着外层元组变量的每一次的取值变化，都需要重新执行‘子查询’以获得相关的中间查询结果
  - 这样的子查询也被称为‘相关子查询’，其处理顺序是由‘外’到‘内’，直至处理完外层查询表中的所有元组

(1) IN谓词的使用

例3 查询在计算机系、数学系及物理系的学生的姓名

```
SELECT  sn
FROM    S
WHERE  sd  IN  ('MA', 'CS', 'PY') ;
```

(1) IN谓词的使用

例4：查询修读课程号为C1的所有学生的姓名

```
SELECT  S.sn
FROM    S
WHERE   S.sno  IN  (
    SELECT  SC.sno
    FROM    SC
    WHERE   SC.cno = 'C1' );
```

注意：与例3.37在表示方法上的区别

(1) IN谓词的使用

例4：两种嵌套查询的表示方法

方法一

```
SELECT  S.sn
FROM    S
WHERE   S.sno  IN  (
    SELECT  SC.sno
    FROM    SC
    WHERE   SC.cno = 'C1' );
```

方法二

```
SELECT  S.sn
FROM    S
WHERE  'C1'  IN  (
    SELECT  SC.cno
    FROM    SC
    WHERE   SC.sno = S.sno );
```

(1) IN谓词的使用

例5：查询所有成绩都及格的学生的姓名

```
SELECT  sn
FROM    S
WHERE   sno  NOT  IN  (
    SELECT  sno
    FROM    SC
    WHERE   g < 60 );
```

(1) IN谓词的使用

例6：查询修读课程名为JAVA的所有学生的姓名

```
SELECT  S.sn
FROM    S
WHERE   S.sno  IN  (
        SELECT  SC.sno
        FROM    SC
        WHERE  SC.cno  IN  (
               SELECT  C.cno
               FROM    C
               WHERE   C.cn = 'JAVA' ) );
```

```
SELECT  S.sn
FROM    S, SC, C
WHERE   S.sno = SC.sno AND
        SC.cno = C.cno AND
        C.cn = 'JAVA';
```

注意：与例3.38在表示方法上的区别

(2) 限定比较谓词的使用

例7：查询有学生成绩大于C1课程号中所有学生成绩的学生学号

```
SELECT  sno
FROM    SC
WHERE   g  > ALL  (
    SELECT  g
    FROM    SC
    WHERE   cno = 'C1' );
```

(2) 限定比较谓词的使用

例8：查询有学生成绩大于等于C1课程号中的任何一位学生成绩的学生学号

```
SELECT  sno
FROM    SC
WHERE   g  >= ANY  (
    SELECT  g
    FROM    SC
    WHERE   cno = 'C1' );
```

(3) 谓词CONTAINS的使用

例9：查询至少修读学号为S4的学生所修读的所有课程的学生的学号

```
SELECT  sno
FROM    SC
WHERE
      ( SELECT  SCx.cno
        FROM    SC  SCx
        WHERE   SC.sno = SCx.sno )
         CONTAINS
      ( SELECT SCy.cno
        FROM    SC  SCy
        WHERE  SCy.sno = 'S4');
```

(3) 谓词CONTAINS的使用

- 注意：
  - 在SQL标准中并没有提供‘CONTAINS’操作符，因此这是一个不能执行的SQL语句
  - 那些在关系代数中需要用除法（division）来实现的查询，在SQL中的表示方法我们将在后面有关除法的例子中说明

(4) 谓词EXISTS的使用

例3.50：查询修读课程号为C1的所有学生的姓名

```
SELECT  S.sn
FROM    S
WHERE   EXISTS  (
    SELECT  *
    FROM    SC
    WHERE   S.sno = SC.sno  AND
            SC.cno = 'C1' );
```

注意：与例3.37和例4在表示方法上的区别

(4) 谓词EXISTS的使用

例3.50：查询修读课程号不为C1的所有学生的姓名

```
SELECT  S.sn
FROM    S
WHERE   NOT EXISTS  (
    SELECT  *
    FROM    SC
    WHERE   S.sno = SC.sno  AND
            SC.cno = 'C1' );
```

(4) 谓词EXISTS的使用

关系代数	SQL 谓词
自然连接	IN
	EXISTS
减法运算	NOT IN
	NOT EXISTS

3. SQL数据操纵功能

3.1 SQL的基本查询功能

3.2 分层结构查询与集合谓词使用

3.3 SELECT语句间的运算

3.4 SQL计算、统计、分类的功能

3.5 SELECT语句使用的一般规则

3.3 SELECT语句间的运算

- 子查询之间的并、交、差运算

  <子查询1> UNION [ALL] <子查询2>

  <子查询1> INTERSECT [ALL] <子查询2>

  <子查询1> EXCEPT [ALL] <子查询2>

例3.52：查询计算机系的学生以及年龄小于20岁的学生

```
( SELECT * FROM S WHERE sd = 'CS' )
   UNION
( SELECT * FROM S WHERE sa < 20 );
```

3. SQL数据操纵功能

3.1 SQL的基本查询功能

3.2 分层结构查询与集合谓词使用

3.3 SELECT语句间的运算

3.4 SQL计算、统计、分类的功能

3.5 SELECT语句使用的一般规则

SQL计算、统计、分类的功能

- 简单的四则运算
- 统计计算
  - 求总数
  - 求和、求平均值
  - 求最大值、求最小值
- 分组统计

(1) 统计功能

函数名称	参数类型	结果类型	说　明
COUNT	any (can be *)	numeric	count of rows
SUM	numeric	numeric	sum of argument
AVG	numeric	numeric	average of argument
MAX	char or numeric	same as argument	maximum value
MIN	char or numeric	same as argument	minimum value

(1) 统计功能

例3.53：给出全体学生的人数

```
SELECT   COUNT ( * )
FROM      S
```

例3.54：给出学号为S1学生修读的课程门数

```
SELECT  COUNT ( * )
FROM     SC
WHERE   sno = 'S1'
```

例3.55：给出学号为S7学生所修读课程的平均成绩

```
SELECT  AVG ( G )
FROM     SC
WHERE   sno = 'S7'
```

统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例1】查询所有订单的销售总金额

```
SELECT   SUM ( dollars )
FROM     Orders
```

【例2】查询‘p03’号商品被订购的总数量

```
SELECT   SUM ( qty )
FROM     Orders
WHERE    pid = ‘p03’
```

统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例3】查询客户的总人数

```
SELECT   COUNT ( * )
FROM     Customers
```

【例4】查询居住有公司客户的城市数

```
SELECT   COUNT ( distinct  city )
FROM     Customers
```

统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例5】查询所享受的折扣（discnt）并非最高的客户的编号（cid）

不能在WHERE子句中直接使用统计函数

```
SELECT   cid
FROM     Customers
WHERE    discnt < max ( discnt )
```

✗

✓

```
SELECT   cid
FROM     Customers  c1
WHERE    discnt < ALL ( SELECT  max( c2.discnt )
                        FROM    Customers  c2 )
```

统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例6】查询有两个或两个以上的客户订购过的商品的编号（pid）

```
SELECT p.pid
FROM   products  p
WHERE  2 <= ALL (SELECT count(distinct cid)
                 FROM   orders  o
                 WHERE  o.pid = p.pid )
```

(1) 统计功能

- 空值（NULL）在统计函数中的处理
  - 在使用统计函数对一个集合中的元素进行统计计算时，将忽略其中的‘空’值元素
    - 在一个学生S元组集合上，可以执行如下的统计查询
      - COUNT(*)
      - COUNT(sno)
      - COUNT(sd)
  - 在一个空集上进行统计计算时，其返回结果如下：
    - COUNT( ) 返回 0
    - 其它的统计函数均返回空值 NULL

空值处理的例子

- 在Customers表中插入一条记录

  ('c009', 'New_Customer', 'New York')

  - 没有给出discnt属性的取值（系统自动为该属性赋上空值）

【例6】

```
SELECT  count(*),
        count(discnt),  count(distinct discnt),
        avg(discnt),  sum(discnt)
FROM    customers;
```

(2) 计算功能

例3.56：给出修读了课程C_7的学生在该课程上的学分绩（假设课程C_7的学分数为3，成绩g采用5分制）

```
SELECT    sno, cno, g*3
FROM      SC
WHERE     cno = 'C7'
```

例3.57：给出计算机系下一年度学生的年龄

```
SELECT  sn, sa + 1
FROM    S
WHERE   sd = 'CS';
```

(3) 分类功能

- 分类（组）统计查询
  - 分组查询子句

    GROUP BY colname { , colname ... }

    – 根据属性colname的取值的不同，将满足WHERE条件的元组划分为不同的集合

  - 使用GROUP BY子句的目的

    – 可以在SELECT子句中针对不同的元组集合分别进行统计计算，实现分类统计查询

(3) 分类功能

例3.58：给出每个学生的平均成绩

```
SELECT      sno, AVG(G)
FROM        SC
GROUP BY    sno
```

例3.59：给出每个学生修读课程的门数

```
SELECT      sno, COUNT(cno)
FROM        SC
GROUP BY    sno
```

(3) 分类功能

- 可以在同一条SQL语句中同时执行多个统计计算。例如：
  - 统计查询每一个学生的课程选修情况：选修课程的门数，平均成绩，总成绩，最高成绩以及最低成绩

```
SELECT    sno, COUNT(cno), AVG(G),
                SUM(G), MAX(G), MIN(G)
FROM      SC
GROUP BY  sno
```

(3) 分类功能

- 一个很重要的事情是需要保证出现在select语句中但没有被聚集的属性只能是出现在group by子句中的那些属性
- 换句话说，任何没有出现在group by子句中的属性如果出现在select子句中的话，它只能出现在聚集函数内部，否则这样的查询就是错误的
- 例如，下面的查询是错误的

```
SELECT dept_name, ID, avg( salary )
FROM instructor
GROUP BY dept_name
```

  - 在一个特定分组（通过dept_name定义）中的每位教师都有不同的ID，每个分组只能输出一个元组，无法确定输出哪个

(3) 分类功能

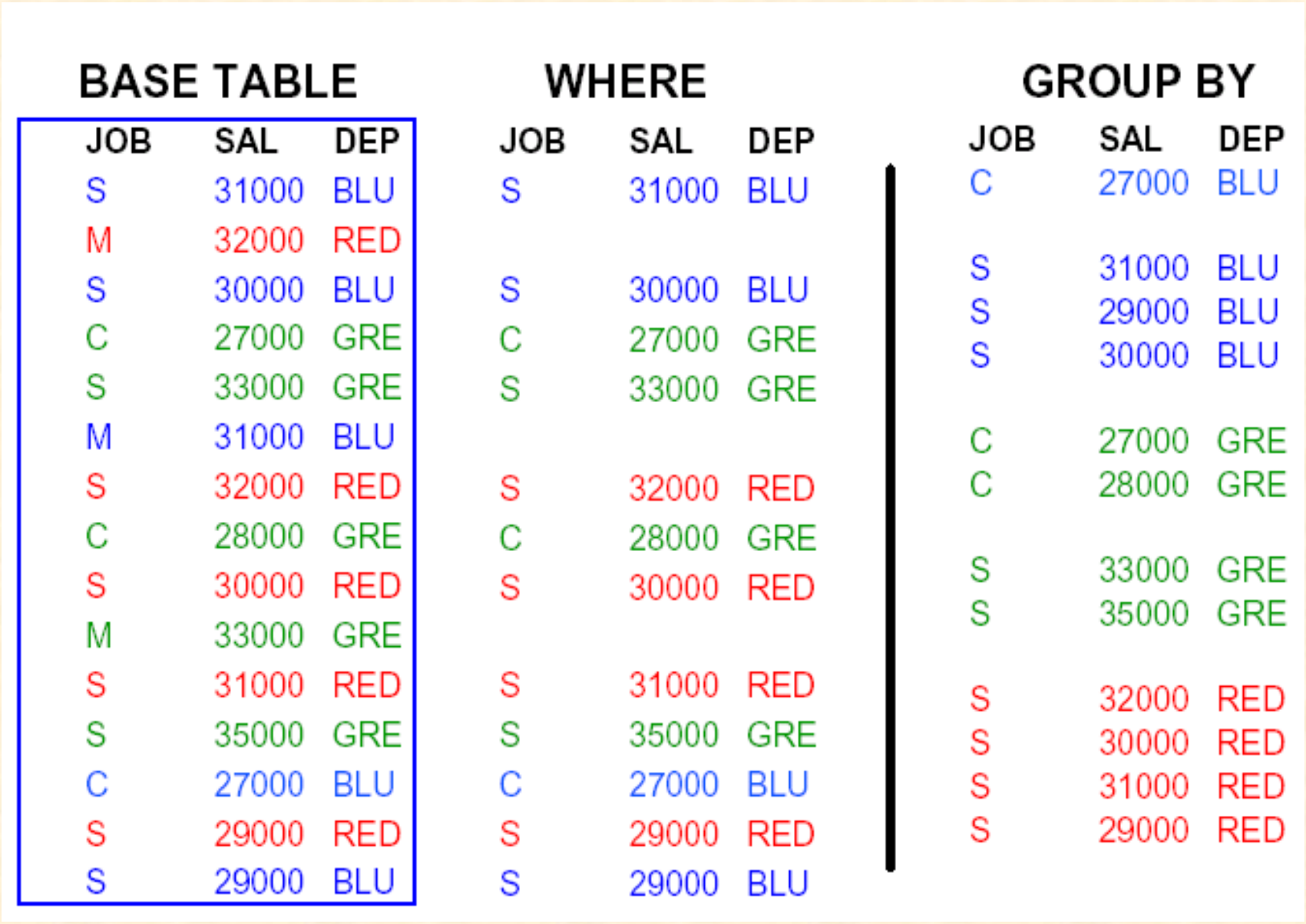
- 分组查询子句： **HAVING group_condition**
  - 根据**GROUP BY**子句的分组结果，定义分组查询条件
    - 在**HAVING**子句中给出的查询条件是定义在分组后的元组集合上的，通常是根据该集合上的某种统计计算的结果来定义查询条件
    - 只有满足条件**group_condition**的元组集合才会被保留下来，用于生成最终的查询结果

(3) 分类功能

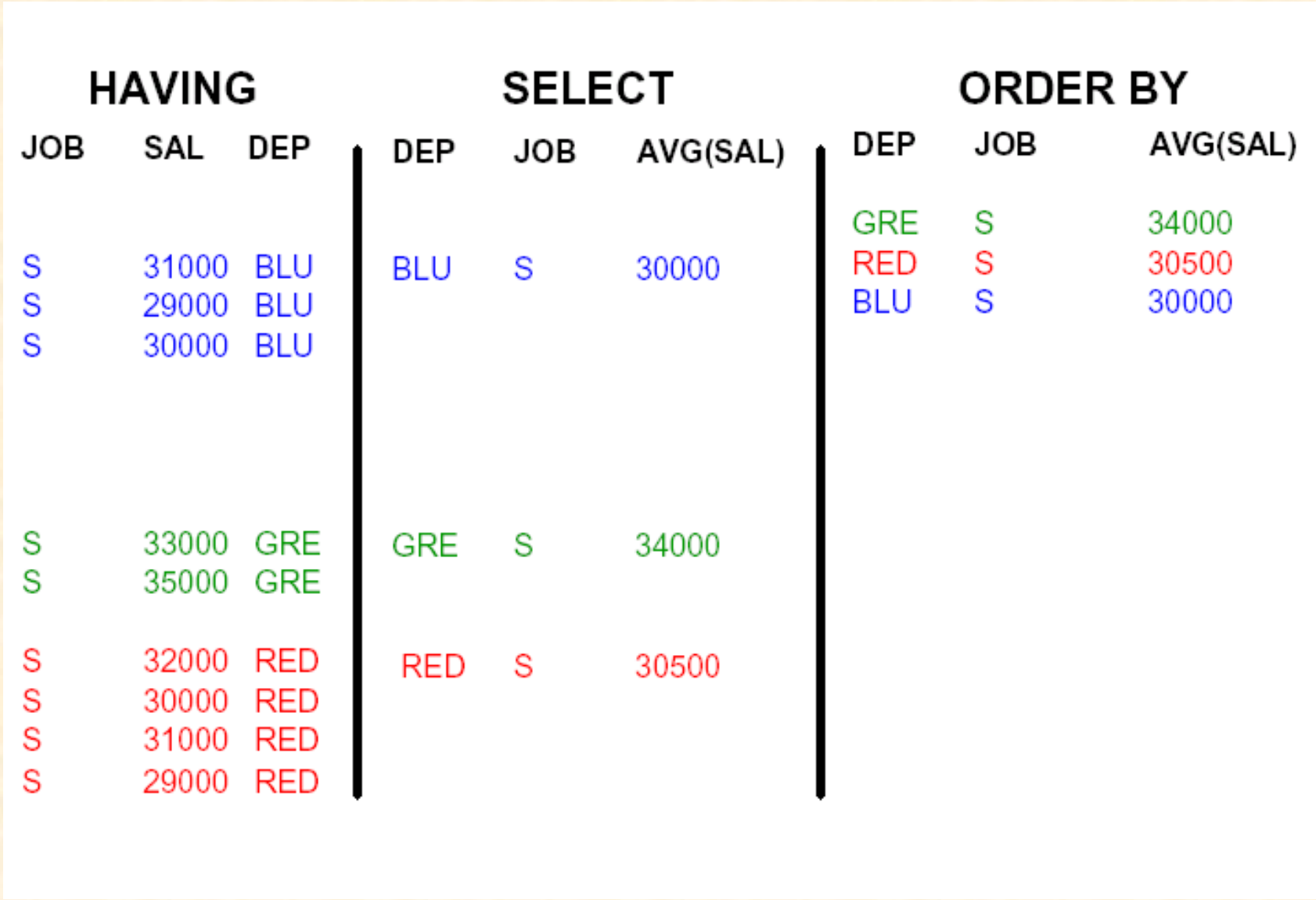
- GROUP BY & HAVING此两子句可以对映像语句所得到的集合元组分组（用GROUP BY子句），并还可利用HAVING子句设置逻辑条件

```
SELECT DEP, JOB, AVG(SAL)
FROM EMPL
WHERE JOB <> 'M'
GROUP BY DEP, JOB
   HAVING AVG(SAL) > 28000
ORDER BY AVG(SAL) DESC
```

(3) 分类功能


BASE TABLE
JOB SAL DEP
S 31000 BLU
M 32000 RED
S 30000 BLU
C 27000 GRE
S 33000 GRE
M 31000 BLU
S 32000 RED
C 28000 GRE
S 30000 RED
M 33000 GRE
S 31000 RED
S 35000 GRE
C 27000 BLU
S 29000 RED
S 29000 BLU
WHERE
JOB SAL DEP
S 31000 BLU
S 30000 BLU
C 27000 GRE
S 33000 GRE
S 32000 RED
C 28000 GRE
S 30000 RED
S 31000 RED
S 35000 GRE
C 27000 BLU
S 29000 RED
S 29000 BLU
GROUP BY
JOB SAL DEP
C 27000 BLU
S 31000 BLU
S 29000 BLU
S 30000 BLU
C 27000 GRE
C 28000 GRE
S 33000 GRE
S 35000 GRE
S 32000 RED
S 30000 RED
S 31000 RED
S 29000 RED

(3) 分类功能


HAVING
JOB SAL DEP
S 31000 BLU
S 29000 BLU
S 30000 BLU
S 33000 GRE
S 35000 GRE
S 32000 RED
S 30000 RED
S 31000 RED
S 29000 RED
SELECT
DEP JOB AVG(SAL)
BLU S 30000
GRE S 34000
RED S 30500
ORDER BY
DEP JOB AVG(SAL)
GRE S 34000
RED S 30500
BLU S 30000

(3) 分类功能

例3.60：给出选修人数超过5个的课程的课程号及其选修人数

```
SELECT      cno,  COUNT(sno)
FROM        SC
GROUP BY  cno
HAVING      COUNT( * )  >  5
```

(3) 分类功能

例3.61：按总平均值降序给出所有课程均及格但不包括C_8的所有学生总平均成绩

```
SELECT      sno,  AVG( G )
FROM        SC
WHERE       cno <> C8
GROUP BY    sno
HAVING      MIN( G ) >= 60
ORDER BY    AVG( G )  DESC
```

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例7】查询每一种被订购过的商品的销售总数量（结果给出商品的编号及其被订购的总数量）

```
SELECT    pid, SUM( qty ) AS total
FROM      Orders
GROUP BY  pid
```

【思考】查询每一种商品的销售总数量（包括从没有被订购过的商品）？

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例8】查询每一个供应商在每一种商品上的销售总数量

```
SELECT    aid, pid, SUM(qty) AS total
FROM      Orders
GROUP BY  aid, pid
ORDER BY  aid, pid
```

该子句不是必须的，只是为了方便用户对于查询结果的浏览

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例9】查询每一个供应商在每一种商品上为‘c002’和‘c003’两位客户订购的总数量（结果给出供应商的编号和名称、商品的编号和名称以及销售总数量）

```
SELECT    a.aid, a.aname, p.pid, p.pname, sum(qty)
FROM      Agents a, Products p, Orders o
WHERE     a.aid=o.aid and p.pid=o.pid and
          (o.cid = 'c002' OR o.cid = 'c003')
GROUP BY  a.aid, a.aname, p.pid, p.pname
```

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例10】统计查询每个供应商在每一种商品上的销售总数量，只返回销售总数超过1000的统计结果（商品编号、供应商编号及其销售总数量）

```
SELECT    pid, aid, sum(qty)
FROM      Orders
GROUP BY  pid, aid
HAVING    sum(qty) > 1000
```

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例11】至少被两个客户购买过的商品的编号（参见前面的例6）

```
SELECT      pid
FROM        Orders
GROUP BY    pid
HAVING      COUNT( distinct cid )  >=  2
```

分类统计查询的例子

Customers (cid, cname, city, discnt)
Agents (aid, aname, city, percent)
Products (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, dollars)

【例12】每个供应商单笔销售最高金额的平均值

```
SELECT  avg ( t.x )
FROM    ( select  aid, max(dollars)  as  x
          from   orders
          group by  aid )  t ;
```

3. SQL数据操纵功能

3.1 SQL的基本查询功能

3.2 分层结构查询与集合谓词使用

3.3 SELECT语句间的运算

3.4 SQL计算、统计、分类的功能

3.5 SELECT语句使用的一般规则

3.5 SELECT语句使用的一般规则

- 映像语句的处理顺序

1. 合并FROM子句中的表（笛卡儿乘积）
2. 利用WHERE子句中的条件进行元组选择，抛弃不满足WHERE条件的那些元组
3. 根据GROUP BY子句对保留下来的元组进行分组
4. 利用HAVING子句中的条件对分组后的元组集合（group）进行选择，抛弃不满足HAVING条件的那些元组集合
5. 根据SELECT子句进行统计计算，生成结果关系中的元组：a group → a result row
6. 根据ORDER BY子句对查询结果进行排序

关系数据库系统数据子语言SQL

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图

4. SQL的更新功能

❑删除功能

```
DELETE FROM table_name
[ WHERE search_condition ] ;
```

- 在table_name表中删除符合条件search_condition的元组
  - 在省略WHERE子句的情况下，删除表table_name中的所有元组
- WHERE子句的构造方式与映像语句中的WHERE子句一样，也可以在其中嵌入子查询（subquery）

数据删除的例子

例3.62：删除姓名为wang的学生记录

```
DELETE  FROM  S
WHERE   sn = 'wang'
```

例3.63：删除计算机系（CS）全体学生的选课记录

```
DELETE
FROM    SC
WHERE   sno   IN  ( SELECT  sno
                    FROM  S
                    WHERE   sd = 'CS' ) ;
```

4. SQL的更新功能

- 元组插入功能

属性名列表可以被省略。在此情况下，属性的排列顺序采用基表定义中的顺序

```
INSERT
INTO  tabname [ ( colname { , colname … } ) ]
VALUES  ( expr | NULL { , expr | NULL … } ) | subquery;
```

属性值，NULL表示空值

被插入的常量元组值。其中属性值的数量及其排列顺序必须与INTO子句中的属性名列表一致

插入单个常量元组

将子查询的查询结果插入到表tabname中。子查询结果属性的排列顺序必须与INTO子句中的顺序一致

数据插入的例子

例3.64：插入一个选课记录（S_{10}, C_{15}, B）

```
INSERT  INTO  SC
VALUES ( 'S10', 'C15', 'B' )
```

【例】插入计算机系（CS）学生选修数据库（Database）课的选课记录

```
INSERT  INTO  SC ( sno,  cno )
        SELECT  sno,  cno
        FROM    S,  C
        WHERE   sd = 'CS' and cn = 'Database'
```

4. SQL的更新功能

❑修改功能

```
UPDATE  table_name
SET  colname = expr | NULL | subquery, ......
[ WHERE  search_condition ] ;
```

➢修改指定基表table_name中满足WHERE条件的元组

– 即：用SET子句中的赋值语句修改相关元组上的属性值

数据更新的例子

【例3.66】将学号为S_{16}的学生系别改为CS

```
UPDATE   S
SET  sd = 'CS'
WHERE   sno = 'S16'
```

数据更新的例子

【例3.67】将数学系学生的年龄均加1岁

```
UPDATE   S
SET      sa = sa+1
WHERE    sd = 'MA'
```

数据更新的例子

【例3.68】将计算机系学生的成绩全置零

```
UPDATE  SC
SET   G = 0
WHERE  sno  IN  ( SELECT   sno
                  FROM     S
                  WHERE    sd = 'CS' )
```

关系数据库系统数据子语言SQL

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图

5. 视图

- 视图（view）
  - 由若干张表经映像语句构筑而成的表
  - 又称为：导出表（drived table）

- 视图（view）与基表（base table）的区别
  - 被称为视图的二维表本身（结构与数据）并不实际存在于数据库内，而仅仅保留了其构造信息（有关视图的定义信息）。因此视图又被称为‘虚表’（virtual table）
  - 当用户执行视图上的访问操作时，数据库管理系统将根据视图的定义命令将用户对于视图的访问操作转换成相应基表上的访问操作

5. 视图

1. 视图定义

CREATE VIEW <视图名> [(<列名> { , <列名> … })]

AS <映像语句> [WITH CHECK OPTION]

- 创建一个以<视图名>为表名的视图，对应的数据查询语句是<映像语句>。以<映像语句>作查询所得到的查询结果即是该视图中的元组
- 如果没有给视图中的属性命名，则用<映像语句>的SELECT子句中的属性名作为视图属性的属性名。否则视图中的属性必需与<映像语句>的SELECT子句中的结果属性一一对应
- WITH CHECK OPTION用于约束视图上的修改操作：如果允许在该视图上执行更新操作，则其更新后的结果元组仍然必需满足视图的定义条件。即通过该视图插入或修改后的新元组能够通过该视图上的查询操作查出来

5. 视图

❑ 视图定义的例子

例3.69：定义一个计算机系学生的视图

```
CREATE   VIEW   CS_S
    AS   SELECT   *
         FROM     S
         WHERE    sd = 'CS'
```

例3.70：定义学生姓名和他修读的课程名及其成绩的视图

```
CREATE   VIEW   S_C_G ( sn, cn, G )
    AS  SELECT  sn, cn, G
        FROM    S, C, SC
        WHERE   S.sno = SC.sno and C.cno = SC.cno
```

❑ 视图定义的例子（cont.）

例3.71：定义学生的学号、姓名及其平均成绩的视图

```
CREATE   VIEW   S_G ( sno, sn, Avg_G )
AS  SELECT  sno,  sn,  AVG( G )
      FROM      S,  SC
      WHERE   S.sno = SC.sno
      GROUP BY   sno, sn
```

5. 视图

❑视图的嵌套定义

➢可以利用已有的视图定义新的视图

【例】定义一个由课程名及该课程的平均成绩构成的视图

```
CREATE  VIEW  C_G ( cn,  Cavg )
      AS  SELECT  cn,  AVG( G )
          FROM    S_C_G
          GROUP  BY  cn
```

经定义已经存在的视图

```
CREATE   VIEW   S_C_G ( sn, cn, G )
    AS  SELECT  sn, cn, G
        FROM    S, C, SC
        WHERE   S.sno = SC.sno and C.cno = SC.cno
```

5. 视图

2. 视图的删除

DROP VIEW <视图名>

- 在执行视图的删除操作时，将连带删除定义在该视图上的其它视图
- 例：执行 ‘DROP VIEW S_C_G’命令将连带删除之前用下述命令创建的视图 C_G 和视图 S_C_G

```
CREATE   VIEW   S_C_G ( sn, cn, G )
     AS  SELECT  sn, cn, G
           FROM    S, C, SC
           WHERE   S.sno = SC.sno and C.cno = SC.cno
```

```
CREATE  VIEW  C_G ( cn,  Cavg )
     AS  SELECT  cn,  AVG( G )
           FROM    S_C_G
           GROUP  BY  cn
```

3. 视图上的操作

- 对视图可以作查询操作
  - 视图上的查询操作将首先被改写为基表上的查询操作，然后才能得到执行

- 一般不允许执行视图上的更新操作，只有在特殊情况下才可以进行：
  - 视图的每一行必须对应基表的惟一一行
  - 视图的每一列必须对应基表的惟一一列

5. 视图

- **WITH CHECK OPTION**

```
CREATE VIEW authors_CA
    AS ( SELECT * FROM Authors WHERE state='CA'
    )

UPDATE authors_CA SET state='NJ'

CREATE VIEW authors_CA
    AS ( SELECT * FROM Authors WHERE state='CA'
    )
    WITH CHECK OPTION
```

5. 视图

- 视图的优点
  - 提高了数据独立性
  - 简化用户观点
  - 提供自动的安全保护功能

关系数据库系统数据子语言SQL

1. SQL概貌
2. SQL数据定义功能
3. SQL数据操纵功能
4. SQL的更新功能
5. 视图
6. DB2 SQL的扩展

日期的比较

- SELECT EMPNO, LASTNAME, BIRTHDATE
  FROM EMPLOYEE
  WHERE BIRTHDATE >= ‘1955-01-01’
  ORDER BY BIRTHDATE

日期时间的数学计算

- 时间相减：得到6位十进制数（decimal(6, 0)），表示时间差
- 日期相减：得到8位十进制数（decimal(8, 0)），表示日期差
- 时间戳相减：得到20位日期时间戳差
- 时间加时间差
- 日期加日期差
- 时间戳加时间戳差

6. DB2 SQL的扩展

- SELECT EMPNO, LASTNAME,
  CURRENT_DATE – BIRTHDATE AS DIFFER
  FROM EMPLOYEE
  WHERE CURRENT_DATE – BIRTHDATE > 650000
  ORDER BY DIFFER DESC

EMPNO	LASTNAME	DIFFER
000130	QUINTANA	721116
000050	GEYER	721116
000340	GOUNOT	720314
000110	LUCCHESI	680926
000310	SETRIGHT	670410
000320	MEHTA	660020

6. DB2 SQL的扩展

- SELECT LASTNAME, FIRSTNAME,
  CURRENT_DATE – BIRTHDATE AS AGE,
  YEAR(CURRENT_DATE – BIRTHDATE) AS YEARS,
  MONTH(CURRENT_DATA – BIRTHDATE) AS MONTHS,
  DAY(CURRENT_DATE – BIRTHDATE) AS DAYS
  FROM EMPLOYEE
  WHERE YEAR(CURRENT_DATE – BIRTHDATE) > 65
  ORDER BY AGE DESC, LASTNAME

6. DB2 SQL的扩展

```
SELECT PROJNO,
  DAYS(PRENDATE) – DAYS(PRSTDATE) AS DAYS
FROM PROJECT
WHERE DAYS(PRENDATE) – DAYS(PRSTDATE)
  <= 300
ORDER BY DAYS
```

* DAYS：将日期值转换为自从12/31/0000以来的天数

6. DB2 SQL的扩展

- SELECT PROJNO, PRENDATE,
  PRENDATE + 2 MONTHS + 15 DAYS
  FROM PROJECT
  WHERE PROJNO = 'AD3100'
  ORDER BY PROJNO

PROJNO	PRENDATE	
AD3100	1983-02-01	1983-04-16

6. DB2 SQL的扩展

- SELECT LASTNAME, FIRSTNAME, WORKDEPT, JOB, SEX FROM EMPLOY ORDER BY WORKDEPT DESC, JOB, LASTNAME, SEX DESC
- 等价的ORDER BY子句
  - ORDER BY WORKDEPT DESC, JOB ASC, LASTNAME ASC, SEX DESC
  - ORDER BY 3 DESC, 4, 1, 5 DESC
  - ORDER BY 3 DESC, 4 ASC, 1 ASC, 5 DESC
  - ORDER BY 3 DESC, JOB, LASTNAME, 5 DESC
  - ORDER BY WORKDEPT DESC, 4 ASC, 1 ASC, SEX DESC

表的连接

❑ 内连接

- 内连接只从叉积中返回满足连接条件的行。如果某一行在一个表中存在，但不在另一张表中，那么结果集将不包括这一行
- 内连接可以通过在**FROM**子句中用**INNER JOIN**操作符来实现

表的连接

❑ 外连接

- ➢ 外连接返回的是内连接操作生成的行，以及内连接操作无法返回的行
- ➢ 外连接共有三类：
  - – 左外连接：包括内连接和在左表中但内连接不会返回的那些行。这类连接在FROM子句中使用LEFT OUTER JOIN（或LEFT JOIN）操作符
  - – 右外连接：RIGHT OUTER JOIN（或RIGHT JOIN）操作符
  - – 全外连接：FULL OUTER JOIN（或FULL JOIN）操作符

表的连接

C11	C12
A	11
B	12
C	13

C21	C22
A	21
C	22
D	23

表的连接

- **SELECT ***
  FROM TAB1 INNER JOIN TAB2 ON C11=C21

C11	C12	C21	C22
A	11	A	21
C	13	C	22

表的连接

- **SELECT ***
  FROM TAB1 RIGHT OUTER JOIN TAB2
  ON C11=C21

C11	C12	C21	C22
A	11	A	21
C	13	C	22
----	----	D	23

表的连接

- **SELECT ***
  FROM TAB1 LEFT OUTER JOIN TAB2
  ON C11=C21

C11	C12	C21	C22
A	11	A	21
B	12	----	----
C	13	C	22

表的连接

- **SELECT ***
  FROM TAB1 FULL OUTER JOIN TAB2
  ON C11=C21

C11	C12	C21	C22
A	11	A	21
C	13	C	22
B	12	----	----
----	----	D	23

UPDATE

- UPDATE语句用于改变表或视图中的数据
  - UPDATE TESTEMP
    SET bonus = 500, salary = 26000
    WHERE empno = '000111'
  - UPDATE STAFF
    SET (dept, salary) = (51, 70000)
    WHERE id = 750