南京大学 NANJING UNIVERSITY

奋进行动 团结奋斗 争先进位 坐言起行 应势而动

数 据 管 理 基 础

第3章 关系数据库标准语言SQL

SQL数据查询与数据操纵

（复习总结）

智能软件与工程学院

3.4 SQL数据查询

数据查询命令

❑语句格式

```
SELECT [ALL|DISTINCT] <目标列表达式>[,<目标列表达式>] …
FROM <表名或视图名>[,<表名或视图名> ]…|(SELECT 语句) [AS]<别名>
    [ WHERE <条件表达式> ]
    [GROUP BY <列名1> [ HAVING <条件表达式> ] ]
    [ORDER BY <列名2> [ ASC|DESC ] ];
```

❑SELECT子句(目标子句): 指定要显示的属性列（目标列的投影）

❑FROM子句(范围子句): 指定查询范围（基本表、视图、导出表/子查询）

❑WHERE子句(条件子句): 指定查询条件（表的连接、元组选择）

❑GROUP BY子句(分组子句): 对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数。

❑HAVING短语(分组查询子句): 只有满足指定条件的组才被用于目标列的投影

❑ORDER BY子句(排序输出子句): 对查询结果表按指定列值的升序或降序排序输出

SELECT & FROM 子句总结

- **SELECT [ALL|DISTINCT] <目标列表达式>[,<目标列表达式>] …**
  - 目标子句，用于投影生成结果关系；
  - 可以是单个属性投影，也可以是对一个表达式的计算结果进行投影；
  - 可以对投影得到的结果列进行重命名，改变查询结果的列标题；
  - 可以用 * 表示投影出表中的所有属性（按照创建表时的属性定义顺序显示）；
  - 可以用distinct谓词要求系统对结果元组进行唯一性检查（对结果元组进行去重处理）。
- **FROM <表名或视图名>[,<表名或视图名>]… | (SELECT 语句) [AS] <别名>**
  - 指定本次查询可以访问的范围（基本表、视图、导出表/子查询）；
  - 可以为表重命名（即定义一个别名）：<table_name> [AS] <alias_name>
  - 如果对表进行了重命名，那么必须通过定义的别名来访问对应的表；
  - 对表进行重命名的作用：便于实现表的‘自身连接’，简化语句的书写，提高语句的可读性；
  - FROM子句中的表（别名）不能重名，SELECT子句中的结果列不能重名；
  - 如果范围子句中的表存在同名的属性，可以通过‘表名.属性名’的方式来明确定义需要访问哪一张表中的列；否则可以直接通过属性名来访问相关的列。

选择表中的元组 1

- 条件子句： **WHERE <条件表达式>**
  - 是数据查询语句中的可选成分，用于指定查询条件（元组选择条件 & 表与表之间的连接条件）
  - 在指定元组选择条件时，除了常用的比较运算符和AND、OR、NOT逻辑运算外，引入了一些新的查询谓词：

查 询 条 件	谓　　词
比　　较	=, >, <, >=, <=, !=, <>, !>, !<
确定范围	BETWEEN … AND … , NOT BETWEEN … AND …
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（逻辑运算）	AND, OR, NOT

选择表中的元组 2

❑字符匹配

<colname> [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]

- ➢<匹配串>可以是一个完整的字符串，也可以含有通配符
- ➢存在两种类型的通配符：
  - % 匹配任意长度（长度可以为0）的字符串
  - _ 匹配任意的单个字符
- ➢使用方式：
  - 匹配串为固定字符串（LIKE 等价于 ‘相等比较’）
  - 匹配串为含通配符的字符串（也称为‘模糊查询’）
  - 使用换码字符将通配符转义为普通字符

❑空值查询

<colname> IS [NOT] NULL

- ➢不能用 = NULL 代替 IS NULL（但在有些DBMS中允许）

聚集函数

聚集函数	功能描述
COUNT(*)	统计元组个数
COUNT([DISTINCT|ALL] <列名>)	统计一列中值的个数
SUM([DISTINCT|ALL] <列名>)	计算一列值的总和（此列必须为数值型）
AVG([DISTINCT|ALL] <列名>)	计算一列值的平均值（此列必须为数值型）
MAX([DISTINCT|ALL] <列名>)	求一列值中的最大值
MIN([DISTINCT|ALL] <列名>)	求一列值中的最小值

- 聚集函数的使用：只能用在 SELECT子句 或 HAVING子句 中
- 聚集函数的作用对象：元组集合
  - 如果没有使用GROUP BY子句对查询结果分组，聚集函数将作用于整个查询结果；
  - 使用GROUP BY子句对查询结果分组后，聚集函数将分别作用于每个组。

统计查询 & 分组统计查询

- [例3.42]查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC ;
```

- [例3.43]计算1号课程的学生平均成绩。

```
SELECT  AVG(Grade)
FROM   SC
WHERE Cno = '1' ;
```

- 查询每一门课程的选修学生人数与平均成绩。

```
SELECT  Cno, COUNT(*), AVG(Grade)
FROM  SC
GROUP BY  Cno ;
```

- 查询选修学生人数少于10人的课程的课程号。

```
SELECT  Cno
FROM  SC
GROUP BY  Cno
HAVING  COUNT(*) < 10 ;
```

- [例3.48]查询平均成绩大于等于90分的学生学号和平均成绩

```
SELECT  Sno, AVG(Grade)
FROM  SC
GROUP BY  Sno
HAVING  AVG(Grade) >= 90 ;
```

连接查询

- **连接查询**：同时涉及两个或两个以上的表的查询

- **连接条件**或**连接谓词**：用来连接两个表的条件
  - 一般格式：
    - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
    - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

- **连接字段**：连接谓词中的列名称
  - 连接条件中的各连接字段类型必须是可比的，但名字不必相同

- **等值连接**，**自然连接**，**外联接**

连接操作的执行过程

- 多表连接查询在DBMS内的实现方式：嵌套循环 或 索引连接
- 嵌套顺序由DBMS来决定，与在FROM子句中的书写顺序无关，与WHERE子句中的条件定义顺序也无关

- 三张表连接查询

```
SELECT  o.ordno, c.cname, p.pname
FROM  customers c, orders o, products p
WHERE  c.cid=o.cid  and  o.pid=p.pid ;
```

- 嵌套循环执行过程

```
FOR c FROM ROWS 1 TO LAST OF customers
    FOR o FROM ROWS 1 TO LAST OF orders
        FOR p FROM ROWS 1 TO LAST OF products
            if (c.cid=o.cid and o.pid=p.pid)
            {   /* 投影产生一条结果元组 */                }
        END FOR p
    END FOR o
END FOR c
```

自身连接

- **自身连接**：一个表与其自己进行连接
  - 需要给表起别名以示区别
  - 由于所有属性名都是同名属性，因此必须使用别名前缀

- [例3.52]查询每一门课的间接先修课（即先修课的先修课）

```
SELECT  FIRST.Cno, SECOND.Cpno
FROM  Course  FIRST, Course  SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

外连接

- 外连接与普通连接的区别
  - 普通连接操作只输出满足连接条件的元组
  - 外连接操作以指定表为连接主体，将主体表中不满足连接条件的元组一并输出
    - 左外连接
      - 列出左边关系中所有的元组
    - 右外连接
      - 列出右边关系中所有的元组

外连接 2

- [例3.53] 用外联接改写[例3.49]

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
FROM  Student  LEFT OUT JOIN SC ON (Student.Sno=SC.Sno);
```

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80
201215123	王敏	女	18	MA	NULL	NULL
201215125	张立	男	19	IS	NULL	NULL

嵌套查询概述 1

- 一个 SELECT-FROM语句 称为一个 查询块
- 将一个查询块嵌套在另一个查询块中的查询称为嵌套查询（一个查询块成为另一个查询块的组成成分）

```
SELECT Sname                          /*外层查询/父查询*/
FROM Student
WHERE Sno IN
      ( SELECT Sno                    /*内层查询/子查询*/
        FROM SC
        WHERE Cno = '2');
```

  - 最常见的是在WHERE子句、FROM子句中嵌入子查询
  - 有些数据管理系统也允许在SELECT子句、HAVING子句中嵌入子查询
  - 可以用创建的视图来代替FROM子句中的子查询

嵌套查询概述 2

- 上层的查询块称为外层查询或父查询
- 下层查询块称为内层查询或子查询

```
SELECT Sname          /*外层查询/父查询*/
FROM Student
WHERE Sno IN
        ( SELECT Sno  /*内层查询/子查询*/
          FROM SC
          WHERE Cno = '2');
```

- SQL语言允许多层嵌套查询
  - 即一个子查询中还可以嵌套其他子查询
- 子查询的限制
  - 不能使用ORDER BY子句
- 有些嵌套查询可以用连接运算替代
  - 谨慎使用嵌套查询

嵌套查询求解方法

不相关子查询

- 子查询的查询条件不依赖于父查询
- 又被称为‘独立子查询’
- 执行过程：由里向外，逐层处理
  - 每个不相关子查询在上一级查询处理之前求解，其执行结果与父查询无关；
  - 不相关子查询的执行结果被用于建立其父查询的查找条件；
  - 在父查询的执行过程中，不需要再去执行它的不相关子查询。

相关子查询

- 子查询的查询条件依赖于父查询
- 执行过程：由外向里，嵌套循环
  - 首先取外层父查询中表的第一个元组t，根据它与内层子查询相关的列的值去处理内层的子查询。若内层子查询的执行结果能够使得外层父查询的WHERE条件表达式成立，则将此元组t放入结果表（用于投影生成最终的查询结果表）；
  - 然后再取父查询中表的下一个元组；
  - 重复上述过程，直至外层查询中表的元组全部检查完为止。

带有IN谓词的子查询 2

- [例3.55] 查询与“刘晨”在同一个系学习的学生。

- 用‘IN+嵌套子查询’完成此查询

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sdept  IN (
      SELECT Sdept
      FROM Student
      WHERE Sname='刘晨');
   /* 此处为不相关子查询 */
```

- 用‘自身连接’完成此查询

```
SELECT  S1.Sno,S1.Sname,S1.Sdept
FROM    Student  S1, Student  S2
WHERE  S1.Sdept=S2.Sdept  AND
       S2.Sname='刘晨';
```

带有IN谓词的子查询 3

❑[例3.55] 查询与“刘晨”在同一个系学习的学生。

❑ 使用 ‘IN + 不相关子查询’

```
SELECT  Sno, Sname, Sdept
FROM  Student
WHERE  Sdept  IN (
        SELECT  Sdept
        FROM  Student
        WHERE  Sname='刘晨');
```

❑ 使用 ‘IN + 相关子查询’

```
SELECT  x.Sno, x.Sname, x.Sdept
FROM  Student  x
WHERE  '刘晨'  IN (
        SELECT  y.Sname
        FROM  Student  y
        WHERE y.Sdept=x.Sdept);
```

❑也可以用‘相关子查询’来表示查询[例3.55]！

带有IN谓词的子查询 4

- [例3.56]查询选修了课程名为“信息系统”的学生学号和姓名

```
SELECT   Sno,Sname
FROM     Student
WHERE   Sno   IN (
        SELECT Sno
        FROM    SC
        WHERE   Cno IN (
                SELECT Cno
                FROM    Course
                WHERE   Cname='信息系统')
);
/* 使用嵌套不相关子查询的执行过程 */
```

③ 最后在Student关系中取出选修了3号课程的学生的Sno和Sname

② 然后在SC关系中找出选修了3号课程的学生学号

① 首先在Course关系中找出“信息系统”的课程号，为3号

带有IN谓词的子查询 5

- 用连接查询实现[例3.56]

```
SELECT Student.Sno, Student.Sname
FROM   Student, SC, Course
WHERE  Student.Sno = SC.Sno  AND
       SC.Cno = Course.Cno  AND
       Course.Cname = '信息系统';
```

- 巧用表的换名，可将该查询简写如下：

```
SELECT S.Sno, S.Sname
FROM   Student S, SC, Course C
WHERE  S.Sno=SC.Sno AND SC.Cno=C.Cno AND C.Cname='信息系统';
```

- 三张表之间的连接查询，常见的执行过程是三层嵌套循环连接！

带有比较运算符的子查询 1

- 当能确切知道内层查询返回单值时，可用比较运算符>，<，=，>=，<=，!=或< >来代替IN谓词。（但该用法不是通用的！）
- 在[例3.55]中，由于一个学生只可能在一个系学习，则可以用 = 代替IN ：

```
SELECT Sno,Sname,Sdept
FROM   Student
WHERE Sdept = (SELECT Sdept
               FROM   Student
               WHERE  Sname = '刘晨');
```

- 说明：由于姓名Sname不是学生关系Student的码，并不能保证此子查询一定返回单值，所以要慎用属性与子查询之间的直接比较！

带有比较运算符的子查询 2

- [例3.57]找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT   Sno, Cno
FROM     SC  x
WHERE    Grade >= ( SELECT AVG(Grade)
                    FROM   SC y
                    WHERE  y.Sno = x.Sno );
```

相关子查询

- 说明：
  - 在本例中，子查询肯定只返回一个统计结果（单值），可放心直接比较；
  - 即使子查询仅返回单值，也不是所有数据库系统都支持这种类型的表示方式，建议使用IN或带有SOME、ANY、ALL量词的量化比较！

[例3.57] 可能的执行过程

① 从外层查询中取出SC的一个元组x(201215121, 1, 92)，将元组x的Sno值传送给内层查询（用201215121代替内层查询中的x. Sno）

```
SELECT  AVG(Grade)
FROM    SC  y
WHERE  y.Sno='201215121';
```

② 执行这个内层查询，得到值 88

③ 用该值代替内层查询，得到外层查询的WHERE子句：

```
WHERE   Grade >= 88;
```

④ 将元组x的Grade属性值92代入上述的条件表达式进行判断。

- 条件成立，则按照SELECT子句的要求对元组x进行投影，得到一条结果元组(201215121, 1)
- 如果条件不成立，则忽略外层查询的当前元组x

- 不断重复上述步骤①至步骤④的过程，直到外层的SC元组全部处理完毕。最终的查询结果为：

Sno	Cno
201215121	1
201215121	3
201215122	2

带有 ANY（SOME）或 ALL 谓词的子查询 1

- 使用ANY或ALL谓词时必须同时使用比较运算（SOME与ANY是同义词）

谓词	语义	谓词	语义
> ANY	大于子查询结果中的某个值	> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值	< ALL	小于子查询结果中的所有值
>= ANY	大于等于子查询结果中的某个值	>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值	<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值	= ALL	等于子查询结果中的所有值（）
!= ANY	不等于子查询结果中的某个值	!= ALL	不等于子查询结果中的任何一个值
<> ANY		<> ALL	

- ‘ = ALL ’与‘ != ANY ’通常没有实际使用价值。

带有 ANY（SOME）或 ALL 谓词的子查询 2

- [例3.58]查询非计算机科学系中比计算机科学系任意一个学生年龄小的学生姓名和年龄

```
SELECT  Sname,Sage
FROM    Student
WHERE   Sage < ANY (
            SELECT  Sage
            FROM    Student
            WHERE Sdept = 'CS')
    AND Sdept <> 'CS' ;
```

/*父查询块中的条件 */

- 结果：

Sname	Sage
王敏	18
张立	19

- 执行过程：
  - 首先处理子查询，找出CS系中所有学生的年龄，构成一个集合 {20, 19}
  - 再处理父查询，找所有不是CS系且年龄 小于20 或 小于19 的学生

带有 ANY（SOME）或 ALL 谓词的子查询 4

- [例3.59]查询非计算机科学系中比计算机科学系所有学生年龄都小的学生姓名及年龄。

方法一：用ALL谓词

```
SELECT  Sname,Sage
FROM    Student
WHERE  Sage < ALL (
        SELECT  Sage
        FROM  Student
        WHERE  Sdept = 'CS')
    AND Sdept <> 'CS';
```

方法二：用聚集函数

```
SELECT  Sname,Sage
FROM    Student
WHERE  Sage < (
        SELECT  MIN(Sage)
        FROM  Student
        WHERE  Sdept = 'CS')
    AND Sdept <> 'CS';
```

带有 ANY（SOME）或 ALL 谓词的子查询 5

表3.7 ANY(或SOME)、ALL谓词与聚集函数、IN谓词的等价转换关系

	=	<> 或 !=	<	<=	>	>=
ANY	IN		< MAX	<= MAX	> MIN	>= MIN
ALL		NOT IN	< MIN	<= MIN	> MAX	>= MAX

带有EXISTS谓词的子查询 1

- **EXISTS谓词**
  - 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”或逻辑假值“false”。
    - 若内层查询结果非空，则外层的WHERE子句返回真值
    - 若内层查询结果为空，则外层的WHERE子句返回假值
  - 由EXISTS引出的子查询，其目标列表达式通常都用 * ，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义。
- **NOT EXISTS谓词**
  - 若内层查询结果非空，则外层的WHERE子句返回假值
  - 若内层查询结果为空，则外层的WHERE子句返回真值

带有EXISTS谓词的子查询 2

- [例3.60]查询所有选修了1号课程的学生姓名。

```
SELECT  Sname
FROM    Student  S
WHERE  EXISTS ( SELECT  *
                FROM  SC
                WHERE  SC.Sno = S.Sno  AND  Cno = '1');
```

- [例3.61]查询没有选修1号课程的学生姓名。

```
SELECT  Sname
FROM    Student  S
WHERE NOT EXISTS (
                SELECT  *
                FROM   SC
                WHERE SC.Sno=S.Sno  AND  Cno='1');
```

带有EXISTS谓词的子查询 4

❑不同形式的查询间的替换

- ➢一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换
- ➢所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换

❑用EXISTS/NOT EXISTS实现全称量词

- ➢SQL语言中没有全称量词∀ （For all）
- ➢可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$$(\forall x)\, P \equiv \neg\left(\exists x\, (\neg\, P)\right)$$

带有EXISTS谓词的子查询 5

- [例3.55]查询与“刘晨”在同一个系学习的学生。

- IN + 子查询

```
SELECT  Sno, Sname, Sdept
FROM  Student
WHERE  Sdept  IN (
    SELECT  Sdept
    FROM     Student
    WHERE  Sname = ‘刘晨’ );
```

可以用‘EXISTS+子查询’替换

```
SELECT  Sno, Sname, Sdept
FROM     Student S1
WHERE  EXISTS (
    SELECT  *
    FROM  Student S2
    WHERE  S2.Sdept=S1.Sdept
            AND S2.Sname = '刘晨');
```

带有EXISTS谓词的子查询 6

- [例3.62] 查询选修了全部课程的学生姓名。
  - “选修了全部课程”等价于“不存在一门课程是该学生不选修的”

```
SELECT  Sname
FROM  Student
WHERE  NOT EXISTS (
        SELECT  *
        FROM  Course
        WHERE  NOT EXISTS (
                SELECT  *
                FROM  SC
                WHERE  SC.Sno = Student.Sno
                        AND  SC.Cno = Course.Cno  ) );
```

带有EXISTS谓词的子查询 7

- [例3.62] 查询选修了全部课程的学生姓名。
- 最里面的 NOT EXISTS 谓词可以改用 NOT IN 谓词

```
SELECT Sname
FROM Student S
WHERE NOT EXISTS (
    SELECT *
    FROM Course C
    WHERE NOT EXISTS (
        SELECT *
        FROM SC
        WHERE SC.Sno = S.Sno
            AND SC.Cno = C.Cno
    )
);
```

```
SELECT Sname
FROM Student S
WHERE NOT EXISTS (
    SELECT *
    FROM Course C
    WHERE C.Cno NOT IN (
        SELECT SC.Cno
        FROM SC
        WHERE SC.Sno = S.Sno
    )
);
```

带有EXISTS谓词的子查询 8

- [例3.63]查询至少选修了学生201215122选修的全部课程的学生号码。
- 解题思路：
  - 查询语义的解析：如果学号为x的学生满足查询条件，那么对所有的课程y，只要201215122学生选修了课程y，则x也选修了y。
  - 形式化表示：
    - 用p表示谓词 “学生201215122选修了课程y”
    - 用q表示谓词 “学生x选修了课程y”
    - 则上述查询为：（∀y） p → q
    - 等价变换：

$$
\begin{aligned}
(\forall y)\, p \rightarrow q &\equiv \neg\ (\exists y\ (\neg\ (p \rightarrow q\ )\ ) \\
&\equiv \neg\ (\exists y\ (\neg\ (\neg\ p \vee\ q)\ )\ ) \\
&\equiv \neg\ \exists y\ (p \wedge \neg q)
\end{aligned}
$$

  - 变换后语义如下：

    不存在这样的课程y，学生201215122选修了y，而学生x没有选。

带有EXISTS谓词的子查询 8

- [例3.63]查询至少选修了学生201215122选修的全部课程的学生号码。

- 用NOT EXISTS谓词表示如下：

```
SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS (
        SELECT *
        FROM SC SCY
        WHERE SCY.Sno='201215122' AND
              NOT EXISTS (
                    SELECT *
                    FROM SC SCZ
                    WHERE SCZ.Sno=SCX.Sno AND
                          SCZ.Cno=SCY.Cno ) );
```

集合查询

- 集合操作的种类
  - 并操作 UNION [ALL]
  - 交操作 INTERSECT [ALL]
  - 差操作 EXCEPT [ALL]

- 参加集合操作的各个子查询需要满足
  - 结果的列数必须相同
  - 对应列的数据类型也必须相同

基于派生表的查询

- 子查询不仅可以出现在WHERE子句中，还可以出现在FROM子句中，这时子查询生成的临时派生表（Derived Table）成为主查询的查询对象

- [例3.57]找出每个学生超过他自己选修课程平均成绩的课程号

```
SELECT  Sno, Cno
FROM  SC,  (SELECT Sno, Avg(Grade)
                FROM SC
                GROUP BY Sno)
                AS   Avg_sc(avg_sno, avg_grade)
WHERE  SC.Sno = Avg_sc.avg_sno
           and  SC.Grade >= Avg_sc.avg_grade
```

- 可以用创建视图来代替在FROM子句中嵌入子查询。

数据管理基础

数据更新 -- 插入、修改、删除
空值的处理
视图

智能软件与工程学院

插入元组 1

- 将新元组插入指定表中，语句格式

```
INSERT
INTO  <表名> [(<属性列1>[,<属性列2 >…)]
VALUES  (<常量1> [,<常量2>]… );
```

- INTO子句
  - 指定要插入数据的表名及属性列
  - 属性列的顺序可与表定义中的顺序不一致
  - 没有指定属性列：表示要插入的是一条完整的元组，且属性列属性与表定义中的顺序一致
  - 指定部分属性列：插入的元组在其余属性列上取空值

- VALUES子句
  - 提供的值必须与INTO子句匹配：值的个数 & 值的类型

插入元组 2

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

插入子查询结果 1

- 语句格式

  INSERT
  INTO <表名> [(<属性列1> [,<属性列2>…)]
  子查询;

- INTO子句

- 子查询
  - SELECT子句目标列必须与INTO子句匹配
    - 值的个数 & 值的类型

修改数据

❑语句格式

UPDATE <表名>
SET <列名> = <表达式> [, <列名> = <表达式>] ...
[WHERE <条件>] ;

❑功能

- ➢修改指定表中满足WHERE子句条件的元组
- ➢SET子句给出<表达式>的值用于取代相应的属性列
- ➢如果省略WHERE子句，表示要修改表中的所有元组

删除数据

- 语句格式

```
DELETE
FROM    <表名>
[ WHERE <条件> ];
```

- 功能
  - 删除指定表中满足WHERE子句条件的元组

- WHERE子句
  - 指定要删除的元组
  - 缺省表示要删除表中的全部元组，表的定义仍在字典中

插入、修改、删除操作与数据完整性

- 关系数据库管理系统在执行插入、修改、删除语句时会检查操作结果是否破坏表上已定义的完整性规则
  - 实体完整性
  - 参照完整性
  - 用户定义的完整性
    - NOT NULL约束
    - UNIQUE约束
    - 值域约束

数据管理基础

3.6 SQL中的空值

智能软件与工程学院

空值的处理

- 空值就是“不知道”或“不存在”或“无意义”的值。

- 一般有以下几种情况：
  - 该属性应该有一个值，但目前不知道它的具体值
  - 该属性不应该有值
  - 由于某种原因不便于填写

- 空值是一个很特殊的值，含有不确定性。对关系运算带来特殊的问题，需要做特殊的处理。下面从空值的产生、判断、完整性约束、运算等方面来阐述空值处理规则。

空值的产生 1

- [例3.79]向SC表中插入一个元组，学生的学号是“201215126”，课程号是“1”，成绩为空。

```
INSERT  INTO SC(Sno, Cno, Grade)
VALUES('201215126', '1', NULL);
/* 该学生还没有考试成绩，故在成绩字段上写入的是空值NULL */
```

- 或

```
INSERT  INTO  SC(Sno, Cno)
VALUES('201215126', '1');
/* 在插入元组时没有赋值的属性，DBMS自动将该属性赋为空值 */
```

空值的产生 2

- [例3.80]将Student表中学生号为“201215200”的学生所属的系改为空值。

```
UPDATE Student
SET Sdept = NULL
WHERE Sno='201215200';
```

空值的判断

- 判断一个属性的值是否为空值，用IS NULL或IS NOT NULL来表示。

- [例3.81]从Student表中找出漏填了数据的学生信息

```
SELECT *
FROM   Student
WHERE  Sname  IS NULL  or
       Ssex  IS NULL  or
       Sage  IS NULL  or
       Sdept  IS NULL;
```

空值相关的完整性约束

- 属性定义（或者域定义）中
  - 有 NOT NULL 约束条件的不能取空值
  - 主码（primary key）属性不能取空值
  - UNIQUE唯一性约束：当属性值非空时，其值在表中具有唯一性

- 可以通过 UNIQUE + NOT NULL 约束来确保表中元组的唯一性

空值的算术运算、比较运算和逻辑运算 1

- 运算规则
  - 空值与另一个值（包括另一个空值）的算术运算的结果为空值
  - 空值与另一个值（包括另一个空值）的比较运算的结果为UNKNOWN
  - 有UNKNOWN后，传统二值（TRUE，FALSE）逻辑就扩展成了三值逻辑

- 在数据查询语句的WHERE子句和HAVING子句中，只有条件表达式的判断结果为TRUE时，当前元组（或组GROUP）才被作为结果输出。

- 因此，在关系数据库管理系统中，一般不支持三值逻辑，统一规定：空值参与比较运算的结果为逻辑假FALSE！

数据管理基础

3.7 视图

智能软件与工程学院

视图

视图的特点

- 虚表，是从一个或几个基本表（或视图）导出的表
- 只存放视图的定义，不存放视图对应的数据
- 基表中的数据发生变化，从视图中查询出的数据也随之改变

定义视图-建立视图 1

- 语句格式

```
CREATE VIEW <视图名> [(<列名> [,<列名>]...)]
    AS <子查询>
    [WITH CHECK OPTION];
```

- WITH CHECK OPTION
  - 对视图进行UPDATE，INSERT和DELETE操作时要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）

- 子查询可以是任意的SELECT语句，是否可以含有ORDER BY子句和DISTINCT短语，取决于具体系统的实现。
  - 在标准SQL中，视图对应的子查询中不允许有ORDER BY子句，但可以使用DISTINCT短语

定义视图-建立视图 2

- 组成视图的属性列名：全部省略 或 全部指定
  - 全部省略
    - 由子查询中SELECT目标列中的诸字段的名字作为视图中对应列的列名
  - 明确指定视图的所有列名
    - 具体到每一列，可以继续沿用子查询中对应目标列的列名，也可以启用一个新的更合适的列名，但必须保证视图中列名的唯一性
    - 当出现下列情况时，必须明确指定视图的所有列名
      - 某个目标列是聚集函数或列表达式
      - 多表连接时选出了几个同名列作为视图的字段
      - 需要在视图中为某个列启用新的更合适的名字

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，只有满足以下条件才可以：
  - ① 视图的每一行必须对应基表的惟一一行
  - ② 视图的每一列必须对应基表的惟一一列
- 同时满足上述两个条件的视图被称为‘可更新视图’(updateable view)。
- 视图上的数据更新操作需要被转化为基表上的数据更新操作，只有在可更新视图上才允许执行数据更新操作(insert, update, delete)。

查询视图 1

- 用户角度：查询视图与查询基本表相同

- 关系数据库管理系统实现视图查询的方法
  - 视图消解法（View Resolution）
    - 进行有效性检查
    - 转换成等价的对基本表的查询
    - 执行修正后的查询

查询视图 3

- [例3.94]在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩

```
SELECT  *
FROM    S_G
WHERE  Gavg>=90;
```

- S_G视图的子查询定义：

```
CREATE VIEW  S_G (Sno,Gavg)
AS
    SELECT  Sno, AVG(Grade)
    FROM   SC
    GROUP BY  Sno;
```

- 视图消解后得到的查询：

```
SELECT  Sno,AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

- 或者，直接在基本表上组织查询

```
SELECT *
FROM  (SELECT Sno,AVG(Grade)
        FROM  SC
        GROUP BY Sno)
        AS S_G(Sno,Gavg)
WHERE Gavg>=90;
```

更新视图 4

- 允许对行列子集视图进行更新
- 对其他类型视图的更新不同系统有不同限制
  - 更新视图的限制：一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新
- 例：例3.89定义的视图S_G为不可更新视图。

```
UPDATE  S_G
SET     Gavg = 90
WHERE  Sno = '201215121';
```

- 这个对视图的更新无法转换成对基本表SC的更新
- 在一个不允许更新的视图上定义的新视图也不允许更新。

视图的作用 - 视图能够简化用户的操作

- 当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作。
  - 基于多张表连接形成的视图
  - 基于复杂嵌套查询的视图
  - 含导出属性的视图
- 视图使用户能以多种角度看待同一数据。
  - 视图机制能使不同用户以不同方式看待同一数据，
  - 适应数据库共享的需要
- 视图对重构数据库提供了一定程度的逻辑独立性。
  - 使用户的外模式保持不变，用户应用程序通过视图仍然能够查找数据
  - 由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。
- 视图能够对机密数据提供安全保护。
  - 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据
- 适当的利用视图可以更清晰的表达查询。