数据库系统概论

An Introduction to Database System

第三章 关系数据库标准语言 SQL

第三章 关系数据库标准语言SQL

- SQL是英文（Structured Query Language）的缩写，意思为结构化查询语言，它包括了数据定义、查询、操纵和控制四种功能。
- 本章介绍了SQL语言的一些基本操作命令，包括数据定义语句（DDL）、数据操纵语句（DML）及权限的操作，重点和难点是数据操纵语句中的查询操作。

第三章 关系数据库标准语言SQL

3.1 SQL概述

- SQL的特点
  - 1. 综合统一
  - 2. 高度非过程化
  - 3. 面向集合的操作方式
  - 4. 以同一种语法结构提供两种使用方法
  - 5. 语言简洁，易学易用

第三章 关系数据库标准语言SQL

3.1 SQL概述
3.2 数据定义
3.3 查询
3.4 数据更新
3.5 视图
3.6 小结

3.2 数据定义

表 3.2　SQL 的数据定义语句

操作对象	操作方式		
	创　建	删　除	修　改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视　图	CREATE VIEW	DROP VIEW	
索　引	CREATE INDEX	DROP INDEX	

3.2.1 创建数据库

#创建数据库

- CREATE DATABASE <数据库名> [CHARACTER SET <字符集名> COLLATE <校对规则>]

默认utf8 字符集，默认 utf8_general_ci校对规则

#删除数据库

- DROP DATABASE <数据库名>

#查看数据库

- SHOW CREATE DATABASE <数据库名>

#备份数据库

- Mysqldump –u 用户名 -p 密码 <数据库名> <表名> > 绝对路径\\文件名.sql

一、创建基本表

CREATE TABLE <表名>

（<列名> <数据类型>[<列级完整性约束条件>]

[,<列名> <数据类型>[<列级完整性约束条件>]] … [，<表级完整性约束条件>] ）；

- <表名>：所要定义的基本表的名字
- <列名>：组成该表的各个属性（列）
- <列级完整性约束条件>：涉及相应属性列的完整性约束条件
- <表级完整性约束条件>：涉及一个或多个属性列的完整性约束条件 （省略则和数据库保持一致）

【例】 建立图书销售系统中的图书库存信息表。

```
CREATE TABLE BookRecord  (
        BookNo   char(30) not null UNIQUE,
        BookName char(200),
        Publisher Char(100),
        Author    Char(30),
        Quantity   Int,
        PRIMARY KEY(BookNo)
);
```

其中NOT NULL指的是该列的值不能为空值。

二、修改基本表

ALTER TABLE <表名>

[ADD <新列名> <数据类型> [完整性约束]]

[DROP <完整性约束名>]

[MODIFY <列名> <数据类型>];

- <表名>：要修改的基本表
- ADD子句：增加新列和新的完整性约束条件
- DROP子句：删除指定的完整性约束条件
- MODIFY子句：用于修改列名和数据类型

三、删除基本表

DROP TABLE <表名> CASCADE/RESTRICT;

基本表删除。数据、表上的索引都删除表上的视图往往仍然保留，但无法引用。

删除基本表时，系统会从数据字典中删去有关该基本表及其索引的描述

与DELETE的区别

【例】 向BookRecord表增加一个“出版日期”的PublishiDate列。

```
ALTER TABLE BookRecord
    ADD PublishiDate datetime NULL;
```

不论基本表中原来是否已有数据，新增加的列一律为空值。

【例】 将折扣的数据类型改为整数

```
ALTER TABLE BookRecord
    MODIFY Discount int;
```

修改原有的列定义有可能会破坏已有数据。

【例】 删除字段“出版日期” PublishiDate列

```
ALTER TABLE BookRecord
    DROP COLUMN PublishiDate
```

【例】 删除关于书号必须取唯一值的约束

```
ALTER TABLE BookRecord
    DROP   UNIQUE(BookNo);
```

3.2.2 建立与删除索引

- 建立索引是加快查询速度的有效手段
- 建立索引
  - DBA或表的属主（即建立表的人）根据需要建立
  - 有些DBMS自动建立以下列上的索引
    - PRIMARY KEY（主键：不可重复 ， not null ,一张表只有一个）
    - UNIQUE（不可重复）
- 维护索引
  - DBMS自动完成
- 使用索引
  - DBMS自动选择是否使用索引以及使用哪些索引

一、建立索引

- 语句格式

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

  - 用<表名>指定要建索引的基本表名字
  - 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
  - 用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
  - UNIQUE表明此索引的每一个索引值只对应唯一的数据记录
  - CLUSTER表示要建立的索引是聚簇索引

二、删除索引

DROP INDEX <索引名>;

- 删除索引时，系统会从数据字典中删去有关该索引的描述。

[例7] 删除Student表的Stusname索引。

DROP INDEX Stusname;

3.3 查　询

3.3.1 概述

3.3.2 单表查询

3.3.3 连接查询

3.3.4 嵌套查询

3.3.5 集合查询

3.3.6 小结

3.3.1 概述

- 语句格式

SELECT [DISTINCT] *|[<目标列名>
，<目标列名>] …
FROM <表名或视图名>[， <表名或视图名>] …
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]];

DISTINCT:取消重复行

语句格式

- SELECT子句：指定要显示的属性列
- FROM子句：指定查询对象(基本表或视图)
- WHERE子句：指定查询条件
- GROUP BY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集函数。
- HAVING短语：筛选出只有满足指定条件的组
- ORDER BY子句：对查询结果表按指定列值的升序或降序排序

示例数据库

学生-课程数据库

- 学生表：Student(Sno，Sname，Ssex，Sage，Sdept)
- 课程表：Course(Cno，Cname，Cpno，Ccredit)
- 学生选课表：SC(Sno，Cno，Grade)

3.3 查　询

3.3.1 概述

3.3.2 单表查询

3.3.3 连接查询

3.3.4 嵌套查询

3.3.5 集合查询

3.3.6 小结

3.3.2 单表查询

查询仅涉及一个表，是一种最简单的查询操作

一、选择表中的若干列

二、选择表中的若干元组

三、对查询结果排序

四、使用集函数

五、对查询结果分组

查询指定列

[例] 查询全体学生的学号与姓名。

```
SELECT Sno，Sname
FROM Student;
```

[例] 查询全体学生的姓名、学号、所在系。

```
SELECT Sname，Sno，Sdept
FROM Student;
```

2.查询满足条件的元组

WHERE子句常用的查询条件

表 3.3　常用的查询条件

查 询 条 件	谓　　词
比　　较	=，>，<，>=，<=，!=，<>，!>，!<； NOT + 上述比较运算符
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	AND，OR

(1) 比较大小

在WHERE子句的<比较条件>中使用比较运算符

- =，>，<，>=，<=，!= 或 <>，!>，!<，
- 逻辑运算符NOT + 比较运算符

  [例] 查询所有年龄在20岁以下的学生姓名及其年龄。

```
SELECT Sname，Sage
FROM   Student
WHERE Sage < 20;
```

或

```
SELECT Sname，Sage
FROM   Student
WHERE NOT Sage >= 20;
```

(2) 确定范围

- 使用谓词 BETWEEN … AND …

  NOT BETWEEN … AND …

[例] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

```
SELECT Sname，Sdept，Sage
FROM  Student
WHERE Sage BETWEEN 20 AND 23;
```

(3) 确定集合

使用谓词　IN <值表>，NOT IN <值表>

<值表>：用逗号分隔的一组取值

[例]查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname，Ssex
FROM  Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' );
```

[例]查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' );
```

(4) 字符串匹配

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

  <匹配串>：指定匹配模板

  匹配模板：固定字符串或含通配符的字符串.当匹配模板为固定字符串时，可以用 = 运算符取代 LIKE 谓词,用 != 或 < >运算符取代 NOT LIKE 谓词

[例] 查询所有不姓刘的学生姓名、学号和性别。

```
SELECT Sname，Sno，Ssex
FROM Student
WHERE Sname NOT LIKE '刘%';
```

(5) 涉及空值的查询

- 使用谓词 IS NULL 或 IS NOT NULL
- “IS NULL” 不能用 “= NULL” 代替

[例] 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno，Cno
 FROM SC
 WHERE Grade IS NULL;
```

(6) 多重条件查询

用逻辑运算符AND和 OR来联结多个查询条件

- AND的优先级高于OR
- 可以用括号改变优先级

可NOT用来实现多种其他谓词

- [NOT] IN
- [NOT] BETWEEN ... AND ...

[例] 查询计算机系年龄在20岁以下的学生姓名。

```
SELECT Sname
 FROM  Student
 WHERE Sdept= 'CS' AND Sage<20;
```

[例] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

```
SELECT Sname，Sdept，Sage
    FROM Student
    WHERE  Sage>=20 AND Sage<=23;
```

三、对查询结果排序

使用ORDER BY子句

- 可以按一个或多个属性列排序
- 升序：ASC；降序：DESC；缺省值为升序

当排序列含空值时

- ASC：排序列为空值的元组最后显示
- DESC：排序列为空值的元组最先显示

对查询结果排序（续）

[例] 查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列。

```
SELECT Sno，Grade
FROM  SC
WHERE  Cno='3'
ORDER BY Grade DESC;
```

查询结果

Sno	Grade
95010	
95024	
95007	92
95003	82
95010	82
95009	75
95014	61
95002	55

对查询结果排序（续）

[例] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT *
FROM Student
ORDER BY Sdept，Sage DESC;
```

四、使用集函数

5类主要集函数

- 计数

  COUNT（[DISTINCT|ALL] *）

  COUNT（[DISTINCT|ALL] <列名>）

  (COUNT*:返回满足条件的记录的行数

  COUNT列：统计满足条件的某列有多少个，会排除为空)

- 计算总和

  SUM（[DISTINCT|ALL] <列名>）

- 计算平均值

  AVG（[DISTINCT|ALL] <列名>）

使用集函数（续）

求最大值

MAX（[DISTINCT|ALL] <列名>）

求最小值

MIN（[DISTINCT|ALL] <列名>）

- DISTINCT短语：在计算时要取消指定列中的重复值
- ALL短语：不取消重复值
- ALL为缺省值

五、对查询结果分组

使用GROUP BY子句分组

细化集函数的作用对象

- 未对查询结果分组，集函数将作用于整个查询结果
- 对查询结果分组后，集函数将分别作用于每个组

使用GROUP BY子句分组

[例] 求各个课程号及相应的选课人数。

```
SELECT Cno，COUNT(Sno)
FROM   SC
GROUP BY Cno;
```

结果

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48

对查询结果分组（续）

- GROUP BY子句的作用对象是查询的中间结果表
- 分组方法：按指定的一列或多列值分组，值相等的为一组
- 使用GROUP BY子句后，SELECT子句的列名列表中只能出现分组属性和集函数

使用HAVING短语筛选最终输出结果

[例] 查询选修了3门以上课程的学生学号。

```
SELECT Sno
FROM  SC
GROUP BY Sno
HAVING  COUNT(*) >3;
```

例题

[例] 查询有3门以上课程是90分以上的学生的学号及（90分以上的）课程数

```
SELECT  Sno,  COUNT(*)
FROM   SC
WHERE Grade>=90
GROUP BY Sno
HAVING COUNT(*)>=3;
```

使用HAVING短语筛选最终输出结果

- 只有满足HAVING短语指定条件的组才输出
- HAVING短语与WHERE子句的区别：作用对象不同
  - WHERE子句作用于基表或视图，从中选择满足条件的元组。
  - HAVING短语作用于组，从中选择满足条件的组。

3.3 查　询

3.3.1 概述

3.3.2 单表查询

3.3.3 连接查询

3.3.4 嵌套查询

3.3.5 集合查询

3.3.6 小结

3.3.3 连接查询

同时涉及多个表的查询称为连接查询

用来连接两个表的条件称为连接条件或连接谓词

一般格式：

- [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

  比较运算符：=、>、<、>=、<=、!=

- [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

连接查询（续）

- 连接字段
  - 连接谓词中的列名称为连接字段
  - 连接条件中的各连接字段类型必须是可比的，但不必是相同的

连接查询（续）

SQL中连接查询的主要类型

- 广义笛卡尔积
- 等值连接(含自然连接)
- 非等值连接查询
- 自身连接查询
- 外连接查询
- 复合条件连接查询

一、广义笛卡尔积

- 不带连接谓词的连接
- 很少使用

例：

```
SELECT  Student.* ,  SC.*
 FROM     Student, SC
```

二、等值与非等值连接查询

等值连接、自然连接、非等值连接

[例] 查询每个学生及其选修课程的情况。

```
SELECT  Student.*，SC.*
FROM    Student，SC
WHERE  Student.Sno = SC.Sno；
```

等值连接

- 连接运算符为 = 的连接操作
  - [<表名1>.]<列名1> = [<表名2>.]<列名2>
  - 任何子句中引用表1和表2中同名属性时，都必须加表名前缀。引用唯一属性名时可以加也可以省略表名前缀。

等值连接

假设Student表、SC表分别有下列数据：

Student表

Sno	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

等值连接

SC表

Sno	Cno	Grade
95001	1	92
95001	2	85
95001 95002 95002	3 2 3	88 90 80

等值连接

结果表

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
95001	李勇	男	20	CS	95001	1	92
95001	李勇	男	20	CS	95001	2	85
95001	李勇	男	20	CS	95001	3	88
95002	刘晨	女	19	IS	95002	2	90
95002	刘晨	女	19	IS	95002	3	80

自然连接

- 等值连接的一种特殊情况，把目标列中重复的属性列去掉。

[例] 对上例中查询条件用自然连接完成。

```
SELECT  Student.Sno，Sname，Ssex，Sage，
                Sdept，Cno，Grade
FROM    Student，SC
WHERE  Student.Sno = SC.Sno;
```

三、自身连接

- 一个表与其自己进行连接，称为表的**自身连接**
- 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

自身连接（续）

[例] 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;
```

自身连接（续）

FIRST表（Course表）

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

自身连接（续）

SECOND表（Course表）

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

自身连接（续）

查询结果

cno	cpno
1	7
3	5
5	6

四、外连接（Outer Join）

- 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

外连接（续）

[例] 查询每个学生及其选修课程的情况包括没有选修课程的学生----用外连接操作

```
SELECT  Student.Sno，Sname，Ssex，
                    Sage，Sdept，Cno，Grade
FROM    Student，SC
WHERE  Student.Sno = SC.Sno(*);
```

from 表1 left join 表2 on (正常连接条件)

right

外连接（续）

结果：

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
95001	李勇	男	20	CS	1	92
95001	李勇	男	20	CS	2	85
95001	李勇	男	20	CS	3	88
95002	刘晨	女	19	IS	2	90
95002	刘晨	女	19	IS	3	80
95003	王敏	女	18	MA		
95004	张立	男	19	IS		

外连接（续）

- 在表名后面加外连接操作符(*)或(+)指定非主体表
- 非主体表有一“万能”的虚行，该行全部由空值组成
- 虚行可以和主体表中所有不满足连接条件的元组进行连接
- 由于虚行各列全部是空值，因此与虚行连接的结果中，来自非主体表的属性值全部是空值

外连接（续）

- 左外连接（左侧表完全显示）
  - 外连接符出现在连接条件的左边
- 右外连接（右侧表完全显示）
  - 外连接符出现在连接条件的右边

3.3 查　询

- 3.3.1 概述
- 3.3.2 单表查询
- 3.3.3 连接查询
- 3.3.4 嵌套查询
- 3.3.5 集合查询
- 3.3.6 小结

3.3.4 嵌套查询

- 嵌套查询概述
- 嵌套查询分类
- 嵌套查询求解方法
- 引出子查询的谓词

嵌套查询(续)多行子查询

```
SELECT Sname                    外层查询/父查询
FROM Student
WHERE Sno IN
    （ no                内层查询/子查询
      FROM SC
      WHERE Cno= ' 2 '）；
```

嵌套查询(续)

- 子查询的限制
  - 不能使用ORDER BY子句
- 层层嵌套方式反映了 SQL语言的结构化
- 有些嵌套查询可以用连接运算替代

嵌套查询分类

- 不相关子查询

  子查询的查询条件不依赖于父查询

- 相关子查询

  子查询的查询条件依赖于父查询

嵌套查询求解方法

- 不相关子查询

  是由里向外逐层处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

嵌套查询求解方法（续）

- 相关子查询
  - 首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表；
  - 然后再取外层表的下一个元组；
  - 重复这一过程，直至外层表全部检查完为止。

引出子查询的谓词

- 带有IN谓词的子查询
- 带有比较运算符的子查询
- 带有ANY或ALL谓词的子查询
- 带有EXISTS谓词的子查询

一、带有IN谓词的子查询

[例] 查询与“刘晨”在同一个系学习的学生。

此查询要求可以分步来完成

① 确定“刘晨”所在系名

```
SELECT  Sdept
FROM    Student
WHERE  Sname= ' 刘晨 ';
```

结果为：

Sdept
IS

构造嵌套查询

将第一步查询嵌入到第二步查询的条件中

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

此查询为不相关子查询。DBMS求解该查询时也是分步去做的。

带有IN谓词的子查询（续）

父查询和子查询中的表均可以定义别名

```
SELECT Sno，Sname，Sdept
 FROM   Student  S1
 WHERE  S1.Sdept  IN
        (SELECT Sdept
         FROM    Student  S2
         WHERE S2.Sname=‘刘晨’);
```

带有IN谓词的子查询（续）

[例]查询选修了课程名为“信息系统”的学生学号和姓名

```
SELECT Sno，Snam
FROM    Student
WHERE Sno  IN
      (SELECT Sno
       FROM    SC
       WHERE  Cno IN
          (SELECT Cno
          FROM Course
          WHERE Cname= ‘信息系统’ ));
```

③ 最后在Student关系中取出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，结果为3号

带有IN谓词的子查询（续）

结果：

Sno	Sname
95001	李勇
95002	刘晨

带有IN谓词的子查询（续）

- 用连接查询

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

二、带有比较运算符的子查询

- 当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或<>）。
- 与ANY或ALL谓词配合使用

带有比较运算符的子查询（续）

例：假设一个学生只可能在一个系学习，并且必须属于一个系，则在上例中可以用 = 代替IN：

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

三、带有ANY或ALL谓词的子查询

谓词语义

- ANY：任意一个值
- ALL：所有值

四、带有EXISTS谓词的子查询

1. EXISTS谓词
2. NOT EXISTS谓词
3. 不同形式的查询间的替换
4. 相关子查询的效率
5. 用EXISTS/NOT EXISTS实现全称量词
6. 用EXISTS/NOT EXISTS实现逻辑蕴函

带有EXISTS谓词的子查询(续)

- 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，则返回真值
    - 若内层查询结果为空，则返回假值
  - 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
- 2. NOT EXISTS谓词

带有EXISTS谓词的子查询(续)

［例］ 查询所有选修了1号课程的学生姓名。

用嵌套查询

```
SELECT Sname
FROM Student
WHERE EXISTS
   (SELECT *
    FROM SC                    /*相关子查询*/
    WHERE Sno=Student.Sno AND  Cno= ’ 1 ’);
```

求解过程

带有EXISTS谓词的子查询(续)

思路分析：

- 本查询涉及Student和SC关系。
- 在Student中依次取每个元组的Sno值，用此值去检查SC关系。
- 若SC中存在这样的元组，其Sno值等于此Student.Sno值，并且其Cno= '1'，则取此Student.Sname送入结果关系。

带有EXISTS谓词的子查询(续)

[例] 查询没有选修1号课程的学生姓名。

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

此例用连接运算难于实现

带有EXISTS谓词的子查询(续)

3. 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。

带有EXISTS谓词的子查询(续)

5. 用EXISTS/NOT EXISTS实现全称量词(难点)

- 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：

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

带有EXISTS谓词的子查询(续)

6. 用EXISTS/NOT EXISTS实现逻辑蕴函(难点)

- SQL语言中没有蕴函(Implication)逻辑运算
- 可以利用谓词演算将逻辑蕴函谓词等价转换为：

$$p \rightarrow q \equiv \neg p \vee q$$

3.3 查　询

3.3.1 概述

3.3.2 单表查询

3.3.3 连接查询

3.3.4 嵌套查询

3.3.5 集合查询

3.3.6 小结

3.3.5 集合查询

标准SQL直接支持的集合操作种类

并操作(UNION)

一般商用数据库支持的集合操作种类

并操作(UNION)

交操作(INTERSECT)

差操作(MINUS)

1. 并操作

- 形式

  <查询块>

  UNION

  <查询块>

  - 参加UNION操作的各结果表的列数必须相同；对应项的数据类型也必须相同

并操作（续）

方法二：

```
SELECT  DISTINCT  *
 FROM Student
 WHERE Sdept= 'CS'  OR  Sage<=19;
```

并操作（续）

[例] 查询选修了课程1或者选修了课程2的学生。

方法一：

```
SELECT Sno
FROM SC
WHERE Cno='1'
UNION
SELECT Sno
FROM SC
WHERE Cno='2';
```

并操作（续）

方法二：

```
SELECT  DISTINCT  Sno
 FROM SC
 WHERE Cno='1'  OR  Cno='2';
```

并操作（续）

[例] 设数据库中有一教师表Teacher(Tno, Tname,...)。查询学校中所有师生的姓名。

```
SELECT Sname
 FROM   Student
 UNION
 SELECT  Tname
 FROM   Teacher;
```

2. 交操作

标准SQL中用**INTERSECT**实现交操作

也可用其他方法间接实现。

2. 交操作

[例] 查询计算机科学系的学生与年龄不大于19岁的学生的交集

本例实际上就是查询计算机科学系中年龄不大于19岁的学生

```
SELECT *
 FROM Student
 WHERE Sdept= 'CS' AND
      Sage<=19;
```

3. 差操作

标准SQL中用**EXCEPT**实现差操作

也可用其他方法间接实现。

3. 差操作

[例] 查询计算机科学系的学生与年龄不大于19岁的学生的差集。

本例实际上是查询计算机科学系中年龄大于19岁的学生

```
SELECT *
 FROM Student
 WHERE Sdept= 'CS' AND
       Sage>19;
```

4. 对集合操作结果的排序

- ORDER BY子句只能用于对最终查询结果排序，不能对中间结果排序
- 任何情况下，ORDER BY子句只能出现在最后
- 对集合操作结果排序时，ORDER BY子句中用数字指定排序属性

3.3.6 SELECT语句的一般格式

SELECT [ALL|DISTINCT]
<目标列表达式> [别名] [，<目标列表达式> [别名]] ...
FROM <表名或视图名> [别名]
[，<表名或视图名> [别名]] ...
[WHERE <条件表达式>]
[GROUP BY <列名1>[，<列名1'>] ...
[**HAVING** <条件表达式>]]
[**ORDER BY** <列名2> [ASC|DESC]
[，<列名2'> [ASC|DESC]] ...];

目标列表达式

- 目标列表达式格式

  (1) [<表名>.] *

  (2) [<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] ...

  <属性列名表达式>：由属性列、作用于属性列的集函数和常量的任意算术运算（+，-，*，/）组成的运算公式。

集函数格式

{ COUNT
SUM
AVG
MAX
MIN } ([DISTINCT|ALL] <列名>)

COUNT ([DISTINCT|ALL] *)

条件表达式格式

（1）

<属性列名> θ { <属性列名> | <常量> | [ANY|ALL] (SELECT语句) }

条件表达式格式

（2）

<属性列名> [NOT] BETWEEN { <属性列名> | <常量> | (SELECT 语句) } AND { <属性列名> | <常量> | (SELECT 语句) }

条件表达式格式

（3）

<属性列名> [NOT] IN { (<值1>[，<值2>] ...) | (SELECT语句) }

条件表达式格式

(4) <属性列名> [NOT] LIKE <匹配串>

(5) <属性列名> IS [NOT] NULL

(6) [NOT] EXISTS (SELECT语句)

条件表达式格式

（7）

<条件表达式> { AND / OR } <条件表达式> { AND / OR } <条件表达> ...