# SQL基础

单世民

大连理工大学

# SQL概述

- SQL是一种介于关系代数与关系演算之间的结构化查询语言。SQL是一个通用的、功能极强的数据库语言。

- SQL集数据查询（data query）、数据操作（data manipulation）、数据定义（data definition）、数据控制(data control)于一体。

# SQL概述

- SQL语言包括的内容
  1. **SQL DDL**：定义关系模式、删除关系、建立索引以及修改关系模式；
  2. **SQL DML**：查询、插入、删除和修改；
  3. **嵌入式DML**：嵌入在Pascal、C等宿主语言；
  4. **视图定义**：创建视图；
  5. **权限管理**：对关系和视图的访问进行授权；
  6. **完整性**：定义数据必须满足的完整性约束条件；
  7. **事务控制**：定义事务的开始、提交、和结束等。

# 数据定义与操纵

# SQL数据定义语言

- 基本的数据库对象：
  **表(table)**、视图(view)、索引(index)
- 基本表的创建

```
CREATE TABLE table_name
             (col_name datatype [column level constraint]
             [,col_name datatype [column level constraint]
             ,...]
             [,table level constraint]
             );
```

# SQL数据定义语言

- 基本表的创建-说明
  - 建表的同时可以定义与该表有关的完整性约束,这些约束条件被存入系统的数据字典中,当用户操作表时,DBMS会自动检查该操作是否有违背完整约束条件.
  - 建立约束的考虑：如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上；否则既可以定义在列级上也可以定义在表级上。
  - 表名、列名是不区分大小写的。
  - 对一个用户而言，表名必须唯一；一个表中，列名必须唯一。
  - 表名、列名必须以字母开头，长度不超过30个字符。

# SQL数据定义语言

- 例：创建一个学生表
  建立一个“学生”表Student，它由学号Sno、姓名Sname、性别Ssex、年龄Sage、所在系Sdept五个属性组成。其中学号不能为空，值是唯一的，并且姓名取值也唯一。

```
CREATE TABLE
Student(
        Sno CHAR(9) NOT NULL UNIQUE,
        Sname CHAR(20) UNIQUE,
        Ssex CHAR(2),
        Sage NUMERIC(2,0),
        Sdept CHAR(20)
        );
```

# SQL数据定义语言

- 练习
  创建一个课程表
  建立一个“课程”表Course，它由课程号Cno、课程名Cname、先行课Cpno、学分Ccredit四个属性组成。其中课程号不能为空，值是唯一的，并且课程名取值也唯一。

```
CREATE TABLE
Course(
      Cno CHAR(9) NOT NULL UNIQUE,
      Cname CHAR(20) UNIQUE,
      Cpno CHAR(2),
      Credit NUMERIC(2,0)
      );
```

# SQL数据定义语言

- 常用完整性约束
  - 主码约束：　PRIMARY KEY
  - 唯一性约束：UNIQUE
  - 非空值约束：NOT NULL
  - 参照完整性约束：FOREIGN KEY

# SQL数据定义语言

```
CREATE TABLE
Student(
        Sno CHAR(9) NOT NULL UNIQUE,
        Sname CHAR(20) UNIQUE,
        Ssex CHAR(2),
        Sage NUMERIC(2,0),
        Sdept CHAR(20)
        );
```

```
CREATE TABLE
Student(
        Sno CHAR(9) PRIMARY KEY,
        Sname CHAR(20) UNIQUE,
        Ssex CHAR(2),
        Sage NUMERIC(2,0),
        Sdept CHAR(20)
        );
```

# SQL数据定义语言

```
CREATE TABLE
Course(
      Cno CHAR(9) NOT NULL UNIQUE,
      Cname CHAR(20) UNIQUE,
      Cpno CHAR(2),
      Credit NUMERIC(2,0)
      );
```

```
CREATE TABLE
Course(
      Cno CHAR(9) PRIMARY KEY,
      Cname CHAR(20) UNIQUE,
      Cpno CHAR(2),
      Credit NUMERIC(2,0),
      FOREIGN KEY (Cpno) REFERENCES Course(Cno)
      );
```

# SQL数据定义语言

- 例：建立一个“学生选课”表SC，它由学号Sno、课程号Cno，修课成绩Grade组成，其中(Sno, Cno)为主码。

```
CREATE TABLE SC(
              Sno CHAR(7) ,
              Cno CHAR(4) ,
              Grade   int,
              Primary key (Sno, Cno),
              Foreign key (Sno) References student(Sno),
              Foreign key (Cno) References Course(Cno));
```

# SQL数据定义语言

- 删除基本表

```
DROP TABLE table_name
```

- 练习：
  删除Student表

```
DROP TABLE Student
```

# SQL数据定义语言

- 修改基本表-添加列

```
ALTER TABLE table_name
<modification>;
```

- 例：
向Student表增加“入学时间”（SCome），数据类型为日期型。

```
ALTER TABLE Student
ADD Scome DATE;
```

# SQL数据定义语言

- 修改基本表-改变列的数据类型

```
ALTER TABLE table_name
MODIFY column_name new_datatype;
```

- 例：
  将年龄的数据类型改为半字长整数。

```
ALTER TABLE Student
MODIFY Sage SMALLINT;
```

# SQL数据定义语言

- 修改基本表-删除完整性约束

```
ALTER TABLE table_name
DROP constraint;
```

- 例：
  删除学生姓名必须取唯一值的约束。

```
ALTER TABLE student
DROP unique(sname);
```

# 数据操纵语言

- 插入数据
  两种插入数据方式

  ¤ 插入单个元组

  ¤ 插入子查询结果（参见子查询内容）

# 数据操纵语言

- 插入数据-单条数据

```
INSERT INTO table_name
[(col_name [, col_name...])]
VALUES
(value [, value...]);
```

- 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。
  **若未指定属性列**：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。否则，**若指定部分属性列**：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

# 数据操纵语言

- 例：将一个新学生记录（学号：95020；姓名：陈冬；性别：男；所在系：IS；年龄：18岁）插入到Student表中。

```
INSERT INTO Student
VALUES ('95020','陈冬' ,'男' ,'IS',18);
```

- 例：插入一条选课记录( '95020'，'1 ')。

```
INSERT INTO SC(Sno, Cno)
VALUES ('95020','1');
```

*新插入的记录在Grade列上取空值*

# 数据操纵语言

- 修改数据

```
UPDATE table_name
SET col_name= value[, col_name = value, ...]
[WHERE condition];
```

- 三种修改方式
  - 修改某一个元组的值
  - 修改多个元组的值
  - 带子查询的修改语句（参见子查询内容）

# 数据操纵语言

- 修改数据

例：将学生95001的年龄改为22岁。

```
UPDATE Student
SET Sage=22
WHERE Sno='95001';
```

例：将所有学生的年龄增加1岁。

```
UPDATE Student
SET Sage= Sage+1;
```

例：将信息系所有学生的年龄增加1岁

```
UPDATE Student
SET Sage= Sage+1
WHERE Sdept='IS';
```

# 数据操纵语言

- 删除数据

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

- 三种删除方式
  - 删除某一个元组的值
  - 删除多个元组的值
  - 带子查询的删除语句（参见子查询内容）

# 数据操纵语言

- 删除数据

例：删除学号为95019的学生记录。

```
DELETE FROM Student WHERE Sno='95019';
```

例：删除2号课程的所有选课记录。

```
DELETE FROM SC WHERE Cno='2'; ;
```

例：删除所有的学生选课记录。

```
DELETE FROM SC;
```

# 数据查询

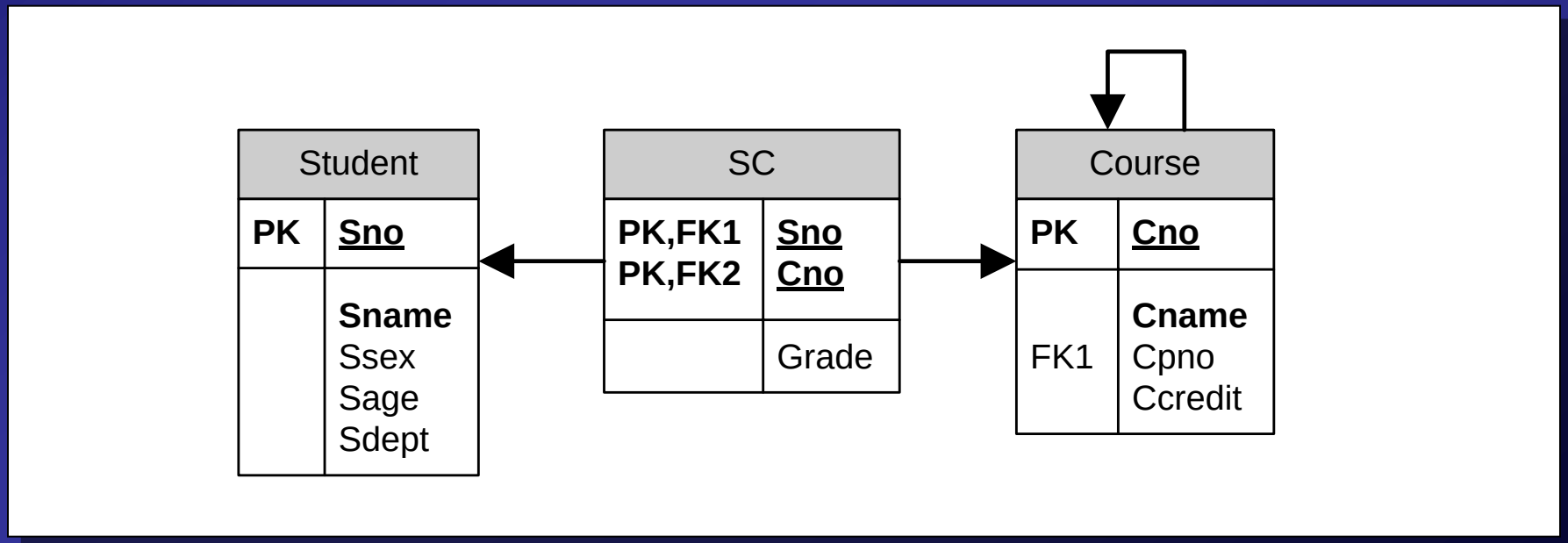
# 数据查询

- 基本语法

```
SELECT
[ALL|DISTINCT]
<column|expression> [alias][, <column|expression> [alias]]···
FROM
<table|view> [tab_alias][, <table|view> [tab_alias]]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

# 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)



# 单表查询

- 基本含义
  查询仅涉及一个表，是一种最简单的查询操作
  - 选择表中的若干列
  - 选择表中的若干元组
  - 对查询结果排序
  - 使用集函数
  - 对查询结果分组

# 单表查询

例：查询全体员工的员工号与姓名。

```
SELECT empno,ename FROM emp;
```

例：查询全体员工的姓名、员工号、所在部门编号。

```
SELECT ename,empno,deptno FROM emp;
```

例：查询全体员工的全部详细记录。

```
SELECT empno,ename,job,hiredate,
       mgr,sal,comm,zip,deptno
FROM emp;
```

```
SELECT * FROM emp;
```

# 单表查询

- 查询经过计算的值

```
SELECT
[ALL|DISTINCT]
<column|expression>[, <column|expression>]···
FROM
<table|view>[, <table|view>]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

SELECT子句的<目标列表达式>为表达式

- 算术表达式
- 字符串常量
- 函数
- 列别名
- ……

# 单表查询

- 查询经过计算的值
  例：查询全体员工的姓名及其每月的税额。

```
SELECT ename,Sal*0.12 tax FROM emp;
```

例：查询全体员工的员工号、雇佣年份和员工姓名，要求员工姓名首字母大写。

```
SELECT empno EmployeeNo,'Hiredate is:' HireMemo,
   hiredate,InitCap(ename)
FROM emp;
```

# 单表查询

- 选择表中的若干元组
  - 消除取值重复的行
  - 查询满足条件的元组

# 单表查询

- 消除取值重复的行

```
SELECT
[ALL|DISTINCT]
<column|expression>[, <column|expression>]···
FROM
<table|view>[, <table|view>]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

# 单表查询

- 消除取值重复的行
  例：查询选修了课程的学生学号。

```
SELECT DISTINCT Sno FROM SC;
```

例：查询选修课程的各种成绩

```
SELECT DISTINCT Cno, Grade FROM SC;
```

*注意：DISTINCT短语的作用范围是所有目标列*

# 单表查询

- 查询满足条件的元组

```
SELECT
[ALL|DISTINCT]
<column|expression>[, <column|expression>]···
FROM
<table|view>[, <table|view>]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

# 单表查询

- 查询满足条件的元组
  Where子句常用查询条件

| 查询条件 | 谓词 |
| --- | --- |
| 比较 | =,<,>,<=,>=,!,!=,<>,!<,!>,NOT+前述操作符 |
| 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 |

# 单表查询

- 基于数值的过滤条件
  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;
```

**OR**

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

# 单表查询

- 基于范围的过滤条件
  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;
```

练习：查询年龄不在20~23岁之间的学生姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage FROM Student
WHERE Sage NOT BETWEEN 20 AND 23;
```

# 单表查询

- 基于给定集合的过滤条件
  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' );
```

# 单表查询

- 基于文本的过滤条件

格式： [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**

- 用 = 运算符取代 LIKE 谓词
- 用 != 或 < >运算符取代 NOT LIKE 谓词

¤ **当模糊查询时**

- % (百分号) 代表任意长度（长度可以为0）的字符串
- _ (下横线) 代表任意单个字符
- 当用户要查询的字符串本身就含有 % 或 _ 时，要使用ESCAPE '<换码字符>' 短语对通配符进行转义。

# 单表查询

- 基于文本的过滤条件
  例：查询学号为95001的学生的详细情况

```
SELECT *
FROM Student
WHERE Sno LIKE '95001';
```

等价于

```
SELECT *
FROM Student
WHERE Sno = '95001';
```

# 单表查询

- 基于文本的过滤条件
  例：查询所有姓刘学生的姓名、学号和性别

```
SELECT Sname, Sno, Ssex
FROM Student
WHERE  Sname LIKE '刘%';
```

例：查询姓"欧阳"且全名为三个汉字的学生的姓名

```
SELECT Sname
FROM   Student
WHERE  Sname LIKE '欧阳__';
```

# 单表查询

- 基于文本的过滤条件
  例：查询名字中第2个字为“阳”字的学生的姓名和学号

```
SELECT Sname, Sno
FROM Student
WHERE Sname LIKE '__阳%';
```

例：查询所有不姓刘的学生姓名

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

# 单表查询

- 基于文本的过滤条件
  例：查询DB_Design课程的课程号和学分

```
SELECT Cno, Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

例：查询以"DB_"开头，且倒数第3个字符为 i 的课程的详细情况。

```
SELECT  *
FROM   Course
WHERE  Cname LIKE  'DB\_%i_ _' ESCAPE '\';
```

# 单表查询

- 针对空值的过滤条件

  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;
```

  例：查所有有成绩的学生学号和课程号。

```
SELECT Sno, Cno FROM SC WHERE Grade IS NOT NULL;
```

*注意：“IS NULL”不能用“= NULL”代替*

# 单表查询

- 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
  - 可用来实现多种其他谓词
    - [NOT] IN
    - [NOT] BETWEEN … AND …

# 单表查询

- 多重条件查询
  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;
```

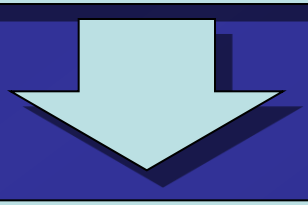
例[illegible]计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname, Ssex
FROM Student
WHERE Sdept='IS' OR Sdept='MA' OR Sdept='CS';
```

# 单表查询

- 多重条件查询
  练习：下面语句如何改写成AND或OR形式？

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



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

# 单表查询

- 查询结果排序

```
SELECT
[ALL|DISTINCT]
<column|expression>[, <column|expression>]···
FROM
<table|view>[, <table|view>]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

- 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

# 单表查询

- 查询结果排序

  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;
```

  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;
```

# 单表查询

- 聚集函数
  - SUM（[DISTINCT|ALL] *）
  - SUM（ [DISTINCT|ALL] <column>）
    计算值的总和并返回总数
  - COUNT（ [DISTINCT|ALL] <column> ）
    计算记录数
  - AVG（ [DISTINCT|ALL] <column> ）
    返回指定列中的平均值
  - MIN（ [DISTINCT|ALL] <column> ）
    返回自变量中指定列的最小值
  - MAX（ [DISTINCT|ALL] <column> ）
    返回自变量中指定列的最大值

  - 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

# 单表查询

- 聚集函数
  例：查询学生总人数。

```
SELECT COUNT(*) FROM Student;
```

例：查询选修了课程的学生人数。

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

*注意：用DISTINCT以避免重复计算学生人数*

# 单表查询

- 聚集函数

例：计算1号课程的学生平均成绩。

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

例：查询选修1号课程的学生最高分数。

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

# 单表查询

- 查询结果分组

```
SELECT
[ALL|DISTINCT]
<column|expression>[, <column|expression>]···
FROM
<table|view>[, <table|view>]···
[WHERE <condition(s)>]
[GROUP BY <column1>[HAVING<condition(s)_1>]]
[ORDER BY <column2>[ASC|DESC]]
```

- 使用GROUP BY子句分组
  - 细化聚集函数的作用对象
    - 未对查询结果分组，聚集函数将作用于整个查询结果
    - 对查询结果分组后，聚集函数将分别作用于每个组
- 只有满足HAVING短语指定条件的组才输出

# 单表查询

- 查询结果分组
  例：求各个课程号及相应的选课人数。

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

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;
```

# 单表查询

- 查询结果分组
  - 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聚集函数

# 单表查询

- 查询结果分组
  练习：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;
```

# 单表查询

- 查询结果分组
  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。

# 连接查询

- 基本含义
  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

- 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

# 连接查询

- 广义笛卡尔积
  不带连接谓词的连接，很少使用

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

# 连接查询

- 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。

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;
```

# 连接查询

- 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：查询每个学生及其选修课程的情况。

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

# 连接查询

- 非等值连接
  连接运算符 不是 = 的连接操作

```
[<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

- 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

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;
```

# 连接查询

- 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

例：查询每个学生及其选修课程的情况，即使学生一门课也没有选也要输出学生信息。

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

# 连接查询

- 复合条件连接
  WHERE子句中含多个连接条件时，称为复合条件连接
  例：查询选修2号课程且其成绩在90分以上的所有学生的学号、姓名

```
SELECT Student.Sno, student.Sname
FROM   Student, SC
WHERE  Student.Sno = SC.Sno AND    /* 连接谓词*/
       SC.Cno= '2'    AND    /* 其他限定条件 */
       SC.Grade > 90;
```

# 连接查询

- 多表连接

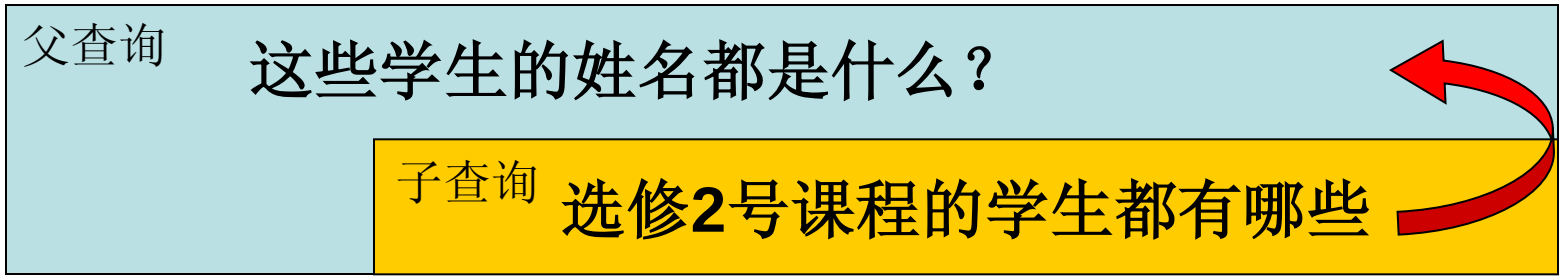
  例：查询每个选修了课程的学生的学号、姓名、选修的课程名及成绩。

```
SELECT Student.Sno, Sname, Cname, Grade
FROM   Student, SC, Course
WHERE  Student.Sno = SC.Sno AND SC.Cno = Course.Cno;
```

# 嵌套查询（子查询）

- 基本含义
  - ¤ 一个SELECT-FROM-WHERE语句称为一个查询块
  - ¤ 将一个查询块嵌套在另一个查询块的WHERE子句或HAVING短语的条件中的查询称为嵌套查询

例：查询选修了2号课程的学生的姓名



# 嵌套查询（子查询）

- 例：查询选修了2号课程的学生的姓名

```
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');
```

¤ 子查询的限制：不能使用ORDER BY子句

*注意： 有些嵌套查询可以用连接运算替代*

# 嵌套查询（子查询）

- 带有IN谓词的子查询

  例：查询选修课程名为“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' ) );
```

① 首先找出Course 关系 中 “信息系统”的 课程号（3）

② 然后在SC关系中找出选修了3号课程的学生学号

③ 最后在Student关系中取出Sno和Sname

# 嵌套查询（子查询）

- 例：查询选修课程名为“信息系统”的学生学号和姓名-使用连接方法

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

# 嵌套查询（子查询）

- 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< >）。
  - ¤ 可与ANY或ALL谓词配合使用

  例：查询与“刘晨”在同一个系学习的学生。假设一个学生只可能在一个系学习，并且必须属于一个系.

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

*注意： 子查询一定要跟在比较符之后*

# 嵌套查询（子查询）

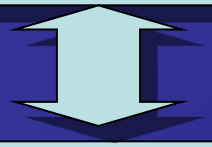
- 带有ANY或ALL谓词的子查询
  - ANY：任意一个值
  - ALL：所有值
  - 配合使用比较运算符的含义

| | |
|---|---|
| > 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（没有实际意义） |
| !=（或<>）ANY | 不等于子查询结果中的某个值 |
| !=（或<>）ALL | 不等于子查询结果中的任何一个值 |

# 嵌套查询（子查询）

- 带有ANY或ALL谓词的子查询
  例：查询其他系中比信息系中某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= 'IS')
      AND Sdept <> 'IS' ;
```



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

# 嵌套查询（子查询）

- 带有ANY或ALL谓词的子查询
  ANY与ALL与集函数的对应关系

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

*用集函数实现子查询通常比直接用ANY或ALL查询效率要高，因为前者通常能够减少比较次数*

# 嵌套查询（子查询）

- 练习：查询其他系中比信息系所有学生年龄都小的学生姓名及年龄

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

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

# 嵌套查询（子查询）

- 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
    处理方式：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。
  - **相关子查询**
    子查询的查询条件依赖于父查询
    处理方式：
    - 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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  - ¤ 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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询所有选修了1号课程的学生姓名。

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

- 求解思路分析：
  - 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。

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询所有选修了1号课程的学生姓名。
  （用连接方法实现）

```
SELECT Sname FROM Student,SC
WHERE Student.Sno=SC.Sno AND SC.Cno= '1';
```

- 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');
```

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  不同形式的查询间的替换

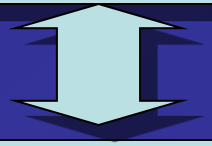
  ¤ 一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换

  ¤ 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。

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询与“刘晨”在同一个系学习的学生。

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



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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  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：

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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询选修了全部课程的学生姓名。

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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  用EXISTS/NOT EXISTS实现逻辑蕴涵*

¤ SQL语言中没有蕴涵(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：

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

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询（至少）选修了学生95002选修的全部课程的学生号码。
- 解题思路
  用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要95002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95002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: $(\forall y)\, p \rightarrow q$

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  等价变换：

$$(\forall y)p \rightarrow q \equiv \neg(\exists y(\neg(p \rightarrow q))$$
$$\equiv \neg(\exists y(\neg(\neg p \vee q)$$
$$\equiv \neg\exists y(p \wedge \neg q)$$

变换后语义：不存在这样的课程y，学生95002选修了y，而学生x没有选。

# 嵌套查询（子查询）

- 带有EXISTS谓词的子查询
  例：查询（至少）选修了学生95002选修的全部课程的学生号码。

```
SELECT DISTINCT Sno FROM SC SCX
WHERE NOT EXISTS
     (SELECT * FROM SC SCY
      WHERE SCY.Sno = '95002' AND
      NOT EXISTS(SELECT * FROM SC SCZ
                 WHERE SCZ.Sno=SCX.Sno
                       AND SCZ.Cno=SCY.Cno));
```

$$\neg\exists y(p \wedge \neg q)$$

# 嵌套查询（子查询）

- 子查询的其他应用
  运用到数据操纵语言中
  对INSERT语句：

```
INSERT INTO table_name (
        SELECT statement
        );
```

对UPDATE语句：

```
UPDATE table
SET column= value[, column = value, ...]
[WHERE condition];
```

# 嵌套查询（子查询）

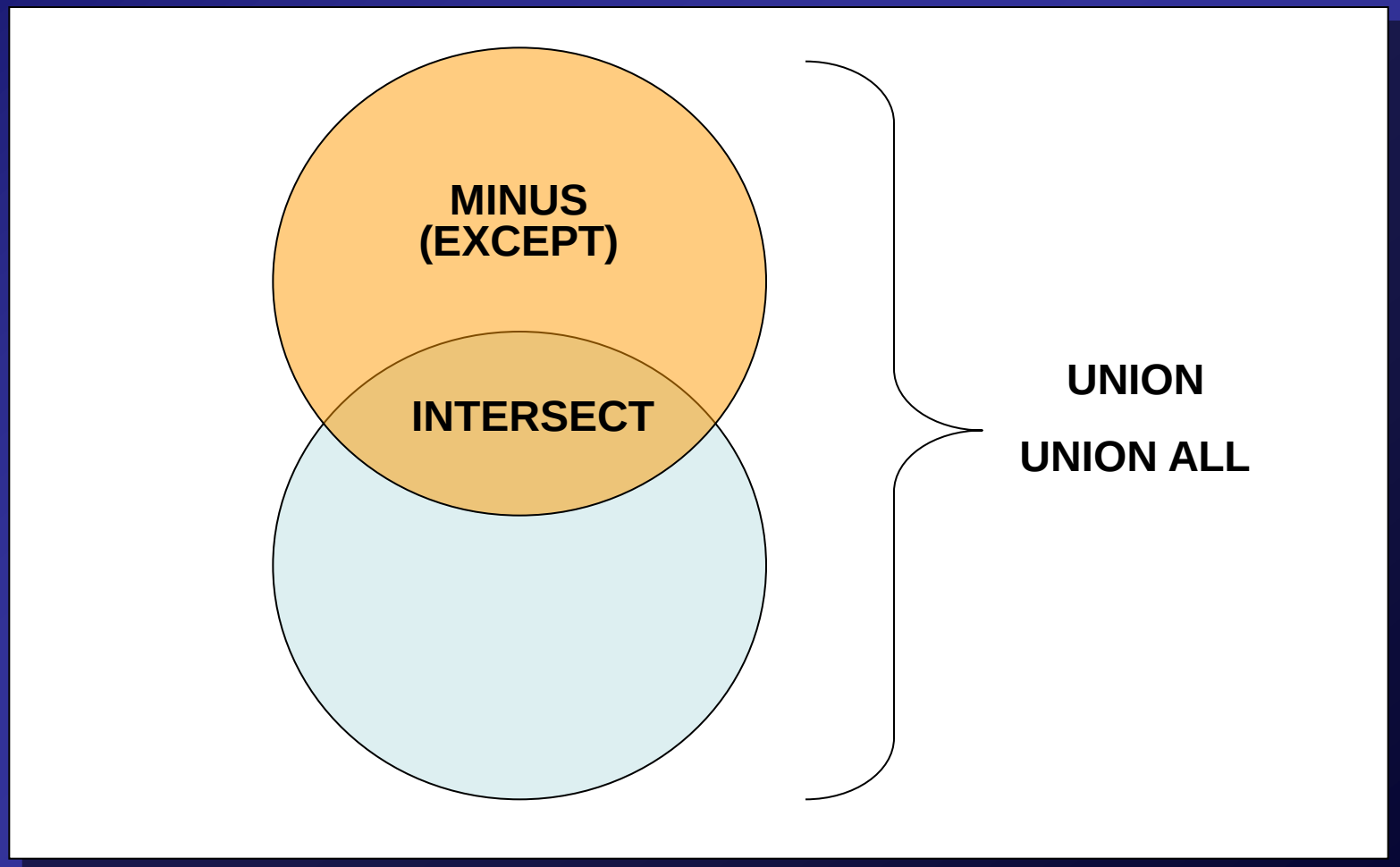
- 例：将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET Grade=0
WHERE exists (SELECT *
              FROM Student
              WHERE Student.Sno = SC.Sno
                    AND student.sdept='CS');
```

- 除了使用在嵌套式SELECT语句之中外，子查询还可以被放在 CREATE VIEW 语句中、CREATE TABLE 语句、UPDATE 语句、INSERT 语句和 DELETE语句中。

# 集合查询

- 基本含义

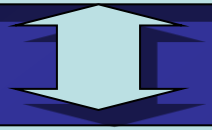


# 集合查询

- 并

例：查询计算机科学系的学生及年龄不大于19岁的学生。

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



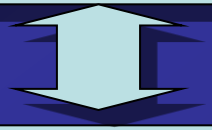
```
SELECT  DISTINCT  *
FROM Student
WHERE Sdept= 'CS' OR Sage<=19;
```

*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*

# 集合查询

- 交
  例：查询计算机科学系的学生及年龄不大于19岁的学生集合的交集

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



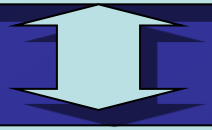
```
SELECT * FROM Student
WHERE Sdept= 'CS' AND Sage<=19;
```

# 集合查询

- 交

例：查询选修课程1的学生集合与选修课程2的学生集合的交集

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



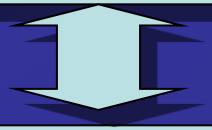
```
SELECT Sno FROM SC
WHERE Cno='1' AND Sno IN (SELECT Sno FROM SC
                          WHERE Cno='2');
```

# 集合查询

- 差

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。

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



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

# 集合查询

- 对集合操作结果的排序

```
SELECT * FROM Student WHERE Sdept= 'CS'
UNION
SELECT * FROM Student WHERE Sage<=19
ORDER BY 1;
```

  - 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子句中可用数字指定排序属性

# Oracle系统中的某些特殊语法

# DUAL表

- Dual表是 Oracle中的一个实际存在的表，任何用户均可读取，常用在没有目标表的Select语句块中

```
DESC DUAL;
SELECT * FROM DUAL;
```

```
SELECT 18*1.05 FROM DUAL;
```

# 表间数据转换

- 同时插入到多表-无条件插入

```
INSERT ALL
INTO first_table_name
      VALUES(first_column_name,...last_column_name)
...
INTO last_table_name
      VALUES(first_column_name,...last_column_name)
SELECT statement
;
```

# 表间数据转换

- 同时插入到多表-有条件插入

```
INSERT ALL
WHEN first_condition THEN INTO first_table_name
      VALUES(first_column_name,...last_column_name)
...
WHEN last_condition THEN INTO last_table_name
      VALUES(first_column_name,...last_column_name)
[ELSE INTO defaul_table_name
      VALUES(first_column_name,...last_column_name)]
SELECT statement
;
```

# 表间数据转换

- 同时插入到多表-有条件插入

```
INSERT FIRST
WHEN first_condition THEN INTO first_table_name
      VALUES(first_column_name,...last_column_name)
...
WHEN last_condition THEN INTO last_table_name
      VALUES(first_column_name,...last_column_name)
[ELSE INTO defaul_table_name
      VALUES(first_column_name,...last_column_name)]
SELECT statement
;
```

# 表间数据转换

- 条件插入-MERGE命令

```
MERGE INTO main_table
USING change_table
ON (main_table.primary_key=change_table.primary_key)
WHEN MATCHED THEN
  UPDATE
  SET
  main_table.first_column=change_table.first_column
  ...
  main_table.last_column=change_table.last_column
WHEN NOT MATCHED THEN
  INSERT(first_column,...last_column)
  VALUES(change_table.first_column,...last_column)
;
```

# 查询记录

- 将多个文本段结合起来

```
SELECT DName||'的地址是'||DAddr
   FROM Department
   WHERE DName LIKE '海源%';
```

# 查询记录

- 为列指定别名

```
SELECT DName||'的地址是'||DAddr 地址信息
    FROM Department
    WHERE DName LIKE '海源%';
```

# 过滤查询结果

- 其他过滤记录的方法
  - 基于日期过滤记录

```
SELECT EName,EHireDate
FROM Employee
WHERE EHireDate<TO_DATE('01/01/2000','MM/DD/YYYY');
```

# 过滤查询结果

- NOT IN与NULL

```
SELECT CID,CName
FROM Card
WHERE CID NOT IN (1,2,3,NULL);
```

*如果指定的值列表中有一个为空值，那么NOT IN就会返回false*

# 连接查询

- 自然连接（NATURAL JOIN）
  列出店面名称以及在店面中工作的员工名称

```
SELECT DName,EName
FROM Department NATURAL JOIN Employee
```

# 连接查询

- 外连接-（+）连接
  在Oracle 9i之前，当执行外连接时，都是使用操作符（+）来完成的。
- 注意事项
  - （+）操作符只能出现在WHERE子句中，并且不能与OUTER JOIN语法同时使用
  - 当使用（+）时，如果在WHERE子句中包含多个条件，则必须在所有条件中都包含（+）操作符
  - （+）只适用于列，而不能用在表达式上。
  - （+）不能与OR和IN操作符一起使用
  - （+）只能用于实现左外连接或右外连接，不能用于是实现完全外连接

# 连接查询

- （+）操作符

```
SELECT VTName, VName
FROM VideoType,Video
WHERE VideoType.VTID=Video.VTID(+);
```

*通常，（+）符号应放置在子表（而不是父表）名称的后面。这是因为：如果合理运用了引用完整性（即有适当的外键约束），那么父表可能包含子表没有的记录，而子表不应该包含与父表不匹配的记录*

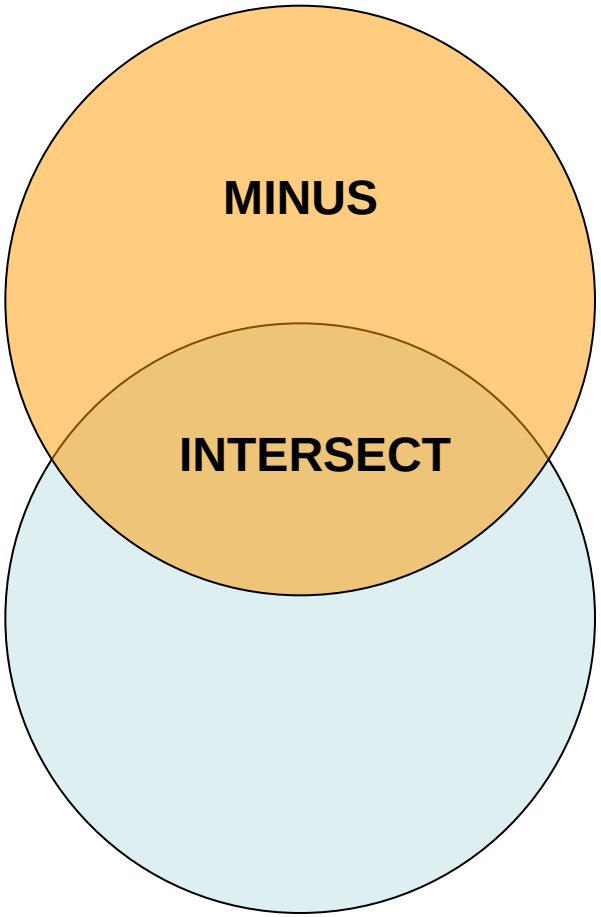
# 连接查询

- USING（SQL/92）
  在满足以下限制时可以使用USING子句对连接条件进行简化
  - 查询必须是等连接的
  - 等连接中的列必须是同名

```
SELECT VTName, VName, VTID
FROM VideoType LEFT JOIN Video
USING VTID;
```

*在using子句中引用列时不要使用表名或别名，否则会出错*

# 连接查询




UNION

UNION ALL

# SQL函数

- 单行函数
  - 系统变量
  - 数字函数
  - 字符函数
  - 日期函数
  - 数据类型转换函数
  - 其他函数
- 多行函数（组函数）

# 单行函数

- 系统变量
  - SYSDATE
  - SYSTIMESTAMP
  - CURRENT_DATE
  - CURRENT_TIMESTAMP
  - USER
  - USERENV

# 系统变量

- Sysdate

```
INSERT INTO PriceHis VALUES(15,SYSDATE,33.2,4);
```

# 系统变量

- USER

```
SELECT USER FROM dual;
```

# 单行函数

- 数字函数
  - ROUND

```
ROUND(input_value,decimal_places_of_pricision)
```

  - TRUNC

```
TRUNC(input_value,decimal_places_of_pricision)
```

# 单行函数

- ROUND

```
SELECT COUNT(*) 影片总数,
  COUNT(*)-COUNT(VDesc) 无介绍影片数,
  ROUND((COUNT(*)-COUNT(VDesc))/COUNT(*)),2) 比例
FROM Video;
```

# 单行函数

- TRUNC
  一种常见的应用是将TRUNC和SYSDATE结合使用

```
SELECT TRUNC(SYSDATE) FROM dual;
```

# 单行函数

- 字符函数
  - UPPER,LOWER,INITCAP
  - LENGTH
  - LTRIM,RTRIM
  - SUBSTR

```
SUBSTR(source_text,starting_character_postion,
                        number_of_characters)
```

  - INSTR

```
INSTR(source_text,text_to_locate,
                     starting_character_position)
```

# 单行函数

- UPPER,LOWER,INITCAP
  示例：查找英文名称中包含stone的电影名称

```
SELECT VName, VEngName
FROM Video
WHERE UPPER(VEngName) LIKE '%STONE%';
```

# 单行函数

- LENGTH
  LENGTH函数在确定存储在数据库中某一列的数据长度时非常有效。

```
SELECT VName 片名,LENGTH(VName) 片名长度
FROM Video
WHERE LENGTH(VName)>10 ORDER BY VName;
```

# 单行函数

- SUBSTR
  在实际应用中经常会出现需要将一个文本串分成几个独立的文本串的需求，比如从一个标向另一个表复制记录。这个分解文本串的过程称为“解析（parse）串”
- 示例：查看各个打印号系列不同的会员卡数量

```
SELECT SUBSTR(CPrintID,1,2) series,COUNT(*) num
FROM Card
GROUP BY SUBSTR(CPrintID,1,2);
```

# 单行函数

- INSTR
  在实际应用中，INSTR经常和SUBSTR配合使用，即以“嵌套函数”的方式进行应用。
- 示例：列出每部影片的第一个演员名称

```
SELECT SUBSTR(VActor,1,INSTR(VActor,';',1)-1)
FROM Video
```

# 单行函数

- LTRIM,RTRIM
  主要用于处理CHAR类型的数据列

```
LTRIM(colmun_name)
RTRIM(colmun_name)
```

# 单行函数

- 日期处理
  - ADD_MONTHS

```
ADD_MONTHS('starting_date',number_of_months)
```

  - LAST_DAY

```
LAST_DAY('date')
```

  - MONTHS_BETWEEN

```
MONTHS_BETWEEN(later_date,earlier_date)
```

# 单行函数

- ADD_MONTHS

```
SELECT ADD_MONTHS(TO_DATE('01/31/2007','MM/DD/YYYY'),1)
FROM dual;
```

```
SELECT ADD_MONTHS(TO_DATE('01/31/2008','MM/DD/YYYY'),1)
FROM dual;
```

```
SELECT ADD_MONTHS(TO_DATE(`12/31/2007','MM/DD/YYYY'),-1)
FROM dual;
```

# 单行行数

- LAST_DAY
  示例：得到新员工领取第一次工资的日期

```
SELECT LAST_DAY(SYSDATE)+10 FROM dual;
```

# 单行函数

- MONTHS_BETWEEN
  示例：查看自己已经出生了多少个月

```
SELECT MONTH_BETWEEN(SYSDATE,
                     TO_DATE('03/02/1986','MM/DD/YYYY'))
FROM dual;
```

# 单行函数

- 数据类型转换
  - TO_CHAR

```
TO_CHAR(input_value,'format_code')
```

  - TO_DATE

```
TO_DATE(input_value,'format_code')
```

# 单行函数

- TO_CHAR
  示例：查看商品的价格，要求价格以小数点对齐。

```
SELECT VName, TO_CHAR(VOutPrice,'L9,999.00') Price
FROM Video
```

# 单行函数

- TO_DATE

```
INSERT INTO EMPLOYEE VALUES
(
 7369,
 20,
 '丘处机',
 '12312341235',
 '副经理',
 7902,
 TO_DATE('17/12/1995','DD/MM/YYYY'),
 800
);
```

# 单行函数

- 其他函数
  - DECODE

```
DECODE(incoming_source,
       incoming_value_1,outgoing_result_1,
       incoming_value_2,outgoing_result_2,
       ……
       last_incoming_value,last_outgoing_result
       [,default_outgoing_result_if_no_match]
       )
```

  - NVL

```
NVL(input_value,result_if_value_is_null)
```

# 单行函数

- DECODE

  示例：查看库存情况，要求输出是“可销售”商品还是“废品”

```
SELECT VName,DName,DECODE(DVType,
                          0,'可销售',
                          1,'废品'),
       DVCount
FROM Dept_Video a, Video b, Department c
WHERE a.DID=c.DID
      AND a.VID=b.VID
```

# 单行函数

- NVL

```
NVL(input_value,result_if_value_is_null)
```

*input_value*通常是某个列名。
*result_if_input_value_is_null*可以使字面值，列的引用或其他任何表达式

- nvl2(input,value1,value2)函数与nvl()不同。在nvl2()函数中，如果表达式input不是空值，则返回value1值；否则返回value2值。
- 示例：显示员工的电话

```
SELECT EName,NVL(EPhone,'N/A') FROM Employee;
```

# 单行函数

- 注意事项：
  NVL要求两个参数必须属于同一数据类型
- 示例：显示每个员工的领导编号

```
SELECT EName,NVL(TO_CHAR(EManager),'N/A') FROM Employee;
```

# 表的维护

- 在当前表的基础上创建新表

```
CREATE TABLE new_table_name AS(
       SELECT statement
       );
```

# 小结

- SQL语言概述
- 数据定义语言（DDL）
  - 建立基本表
  - 删除基本表
  - 更改基本表
    - 添加列
    - 改变列的数据类型
    - 删除完整性约束
- 数据操纵语言（DML）
  - 插入数据
  - 修改数据
  - 删除数据

# 小结

- 单表查询
  - 选择表中的若干列
  - 选择表中的若干元组
  - 对查询结果排序
  - 使用集函数
  - 对查询结果分组
- 连接查询
  - 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

# 小结

- 嵌套查询
  - 不相关子查询与相关子查询
  - 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
- 集合查询
  - 交，差，并
- Oracle的某些特殊语法

# *The End*