# 第三章 关系数据库标准语言SQL

# 本章内容

数据库标准语言SQL介绍

- 数据定义
- 数据查询
- 数据更新

# §3.1 SQL概述

- SQL的发展
  - 1974年，由Boyce和Chamberlin提出
  - 1975~1979，IBM San Jose Research Lab的关系数据库管理系统原型System R实施了这种语言
  - SQL-86是第一个SQL标准
  - SQL-89、SQL-92(SQL2)、SQL-99(SQL3)
  - SQL2003, SQL2008, SQL2011, SQL2016

# SQL的发展(续)

- SQL-99(SQL3)特点
- 关系特征:新的数据类型,例如大对象类型,集合类型 等,增加了递归查询等
- 面向对象特征:用户自定义的结构化类型,实现了函数和方法,有继承关系,对象ID的实现(REF类型)

# SQL的发展(续)

- SQL2003特点
  - ✓ 支持新的数据类型和相应的操作,例如:MULTISET
  - ✓ 支持数据仓库操作,例如MERGE，MERGE = UPDATE + INSERT
  - ✓ 支持 XML
- SQL2016特点
  - * 行模式识别：分析时间序列数据，例如股票行情等
  - * 支持JSON对象
  - * 多态表函数：用动态SQL创建强大复杂的自定义函数
  - * 额外的分析功能：增加三角函数，为多维数组提供支持

# SQL概述

- 现状
  - 大部分DBMS产品都支持SQL，成为操作数据库的标准语言
  - 商业数据库软件对SQL的支持程度不同

本章内容为SQL的核心内容，不涉及扩展内容。

# SQL概述——SQL的功能

- 数据定义（DDL）
  - 定义、删除、修改关系模式（基本表）
  - 定义、删除视图（View）
  - 定义、删除索引（Index）
- 数据操纵（DML）
  - 数据查询
  - 数据增、删、改
- 数据控制功能

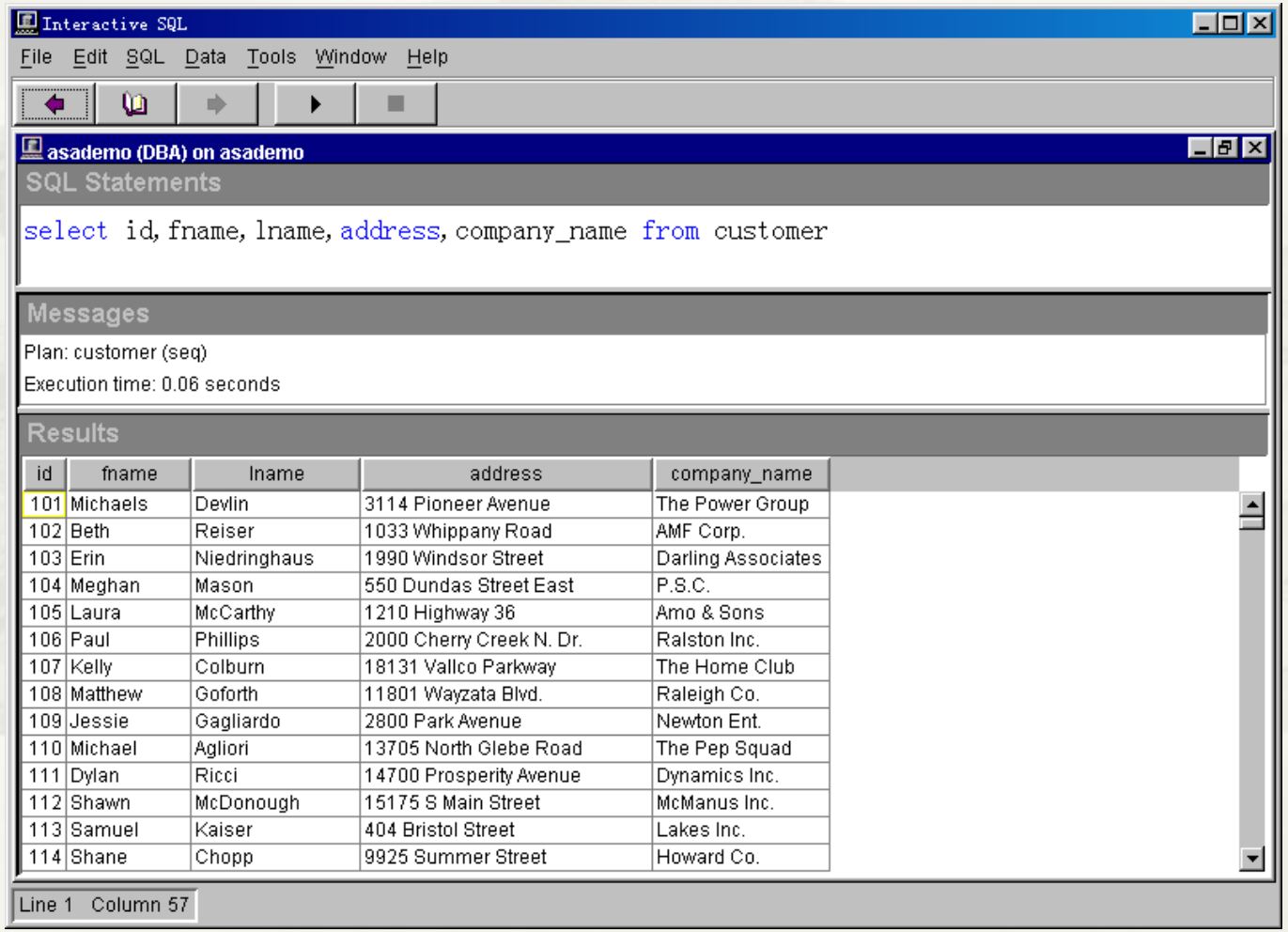
# SQL的特点

- 综合统一（数据定义，数据查询，数据操纵和数据控制功能）
- 非过程化语言（提出做什么，而无须指明怎么做）
- 面向集合的操作
- SQL多种使用形式（独立执行和嵌入在高级语言中）
- 简单，易学

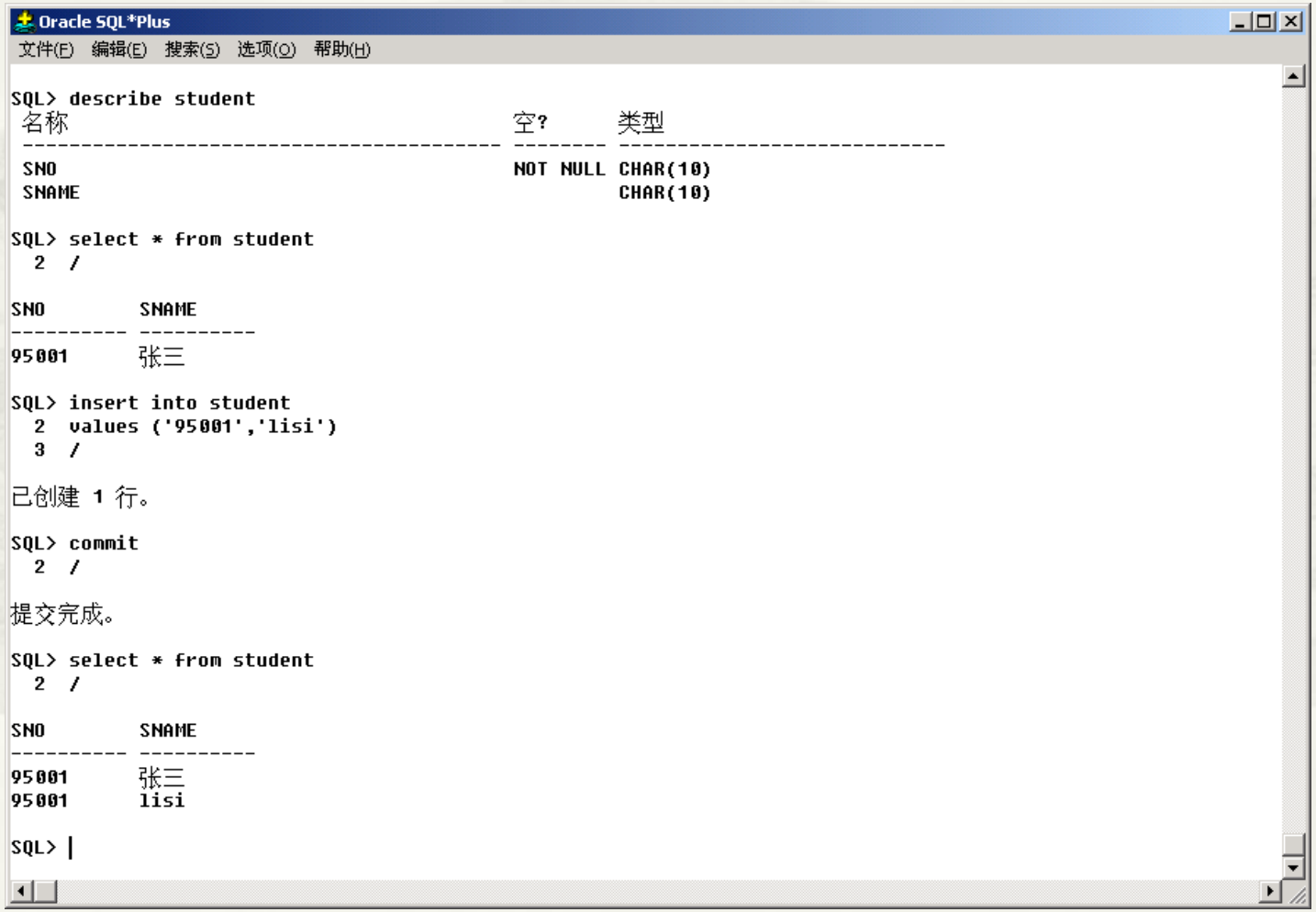
# SQL概述——SQL的形式

- 交互式SQL
  - 一般DBMS都提供联机交互工具
  - 用户可直接键入SQL命令对数据库进行操作
  - 由DBMS来进行解释

# SQL的形式——交互式


Interactive SQL
File Edit SQL Data Tools Window Help
asademo (DBA) on asademo
SQL Statements
select id, fname, lname, address, company_name from customer
Messages
Plan: customer (seq)
Execution time: 0.06 seconds
Results
id fname lname address company_name
101 Michaels Devlin 3114 Pioneer Avenue The Power Group
102 Beth Reiser 1033 Whippany Road AMF Corp.
103 Erin Niedringhaus 1990 Windsor Street Darling Associates
104 Meghan Mason 550 Dundas Street East P.S.C.
105 Laura McCarthy 1210 Highway 36 Amo & Sons
106 Paul Phillips 2000 Cherry Creek N. Dr. Ralston Inc.
107 Kelly Colburn 18131 Vallco Parkway The Home Club
108 Matthew Goforth 11801 Wayzata Blvd. Raleigh Co.
109 Jessie Gagliardo 2800 Park Avenue Newton Ent.
110 Michael Agliori 13705 North Glebe Road The Pep Squad
111 Dylan Ricci 14700 Prosperity Avenue Dynamics Inc.
112 Shawn McDonough 15175 S Main Street McManus Inc.
113 Samuel Kaiser 404 Bristol Street Lakes Inc.
114 Shane Chopp 9925 Summer Street Howard Co.
Line 1 Column 57

# SQL的形式——交互式


Oracle SQL*Plus
文件(F) 编辑(E) 搜索(S) 选项(O) 帮助(H)
SQL> describe student
名称 空? 类型
SNO NOT NULL CHAR(10)
SNAME CHAR(10)
SQL> select * from student
2 /
SNO SNAME
95001 张三
SQL> insert into student
2 values ('95001','lisi')
3 /
已创建 1 行。
SQL> commit
2 /
提交完成。
SQL> select * from student
2 /
SNO SNAME
95001 张三
95001 lisi
SQL>

# SQL概述——SQL的形式

- 嵌入式SQL
  - 能将SQL语句嵌入到高级语言（宿主语言）
  - 使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力
  - 一般需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句

# SQL的形式——嵌入式

```
Main(){
...
exec sql begin declare section;
char co[10];
int id;
exec sql end declare section;
…
exec sql select company_name
          from customer
          where id = :id
          into :co;
…
}
```

- SQL语句前要有标记。
- 共同使用的变量（共享变量）需要事先说明，在SQL语句中用冒号开头。
- 使用光标进行变量与表的数据交换。

# SQL概述——SQL的形式

- SQL/API（Application Programming Interface）——组函数和程序
  - 从宿主语言主程序中调用一个SQL DBMS库，而SQL语句是这个调用的参数
  - 目前更多的数据库编程在使用这种方法
    - ODBC
    - JDBC
    - SQL/CLI

# SQL的形式——SQL/API

```
#include “sqlcli.h”
SQLHSTMT        hstmt;
…
SQLPrepare(hstmt,“Insert Into customer values(……)”);
SQLExcute(hstmt);
…
```

# 交互式：SQL概述——SQL的动词

- 数据查询
  - SELECT
- 数据定义
  - CREATE、DROP、ALTER
- 数据操纵
  - INSERT、UPDATE、DELETE
- 数据控制
  - GRANT、REVOKE

# §3.2 数据定义

- 数据定义语言（Data Definition Language）
- 模式、基本表、视图、索引
  - 创建——Create
  - 修改——Alter（基本表）
  - 删除——Drop

# 数据定义——模式的定义与删除

* 定义模式

CREATE SCHEMA <模式名> AUTHORIZATION<用户名>

  * 如没有指定模式名,则隐含为用户名.
  * 定义模式,实际上是定义了一个命名空间,在该空间中,进一步定义基本表,索引等.

* 删除模式:

DROP SCHEMA <模式名><CASCADE I RESTRICT>

# 数据定义——创建基本表

* 定义一组关系（基本表）、说明各关系的信息
  * 各关系的模式
  * 各属性的值域
  * 完整性约束

# 数据定义——SQL中的域类型

- **数值型**
  - **int (32)**
  - **smallint (16)**
  - **tinyint (8)**
  - **numeric(p,d)**
  - **real、double**
  - **float(n)**

* 字符型
  * char(n)
  * varchar(n)
* 日期/时间型
  * date
  * time
  * datetime

# 数据定义——SQL的模式定义

Create Table R ($A_1D_1C_1$， $A_2D_2C_2$，…， $A_nD_nC_n$，

<表级完整性约束1>，

…

<表级完整性约束n>）

其中：

R　关系名（表名）

$A_i$　关系 r 的一个属性名

$D_n$　属性$A_i$域值的域类型

$C_n$　属性$A_i$的完整性约束

# 数据定义——SQL的模式定义

主键声明：

单属性主键：PRIMARY KEY

多属性主键：primary key ($A_{j1}$，$A_{j2}$，...，$A_{jm}$)

外键声明：

Foreign key ($A_{f1}$，$A_{f2}$，...，$A_{fp}$) Reference S

# 数据定义——创建基本表

* 例

```
Create Table Student (
        sno             char(5) primary key,
        sname           char(10),
        ssex            char(1),
        sage            int,
        sdept           char(2))
```

# 数据定义——创建基本表

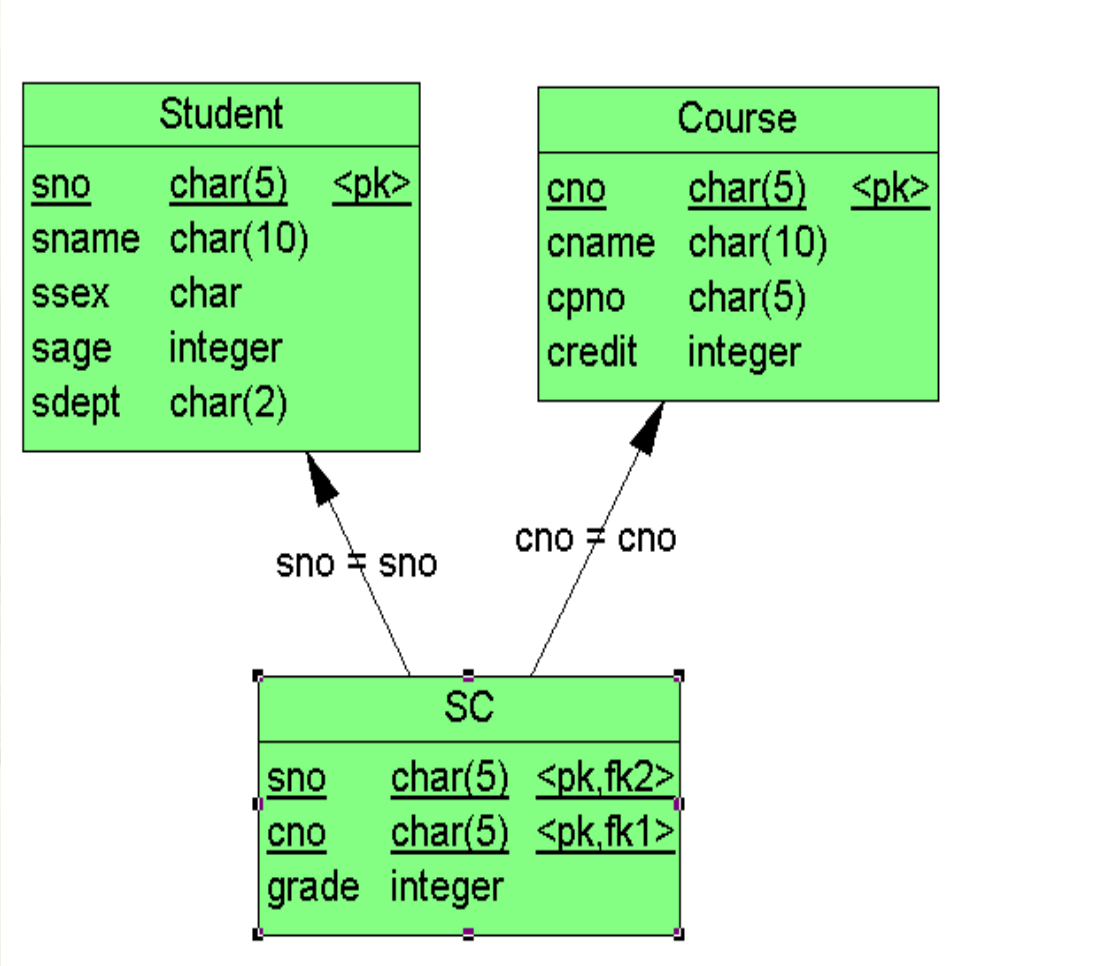
* 例

```
Create Table Course (
        cno             char(5) primary key,
        cname           char(10),
        cpno            char(5),
        credit          int)
```

# 数据定义——创建基本表

* 例

```
Create Table SC (
        sno             char(5),
        cno             char(5),
        grade           int,
        primary key(sno,cno),
        foreign key (sno) references student,
        foreign key (cno) references course)
```



# 数据定义——删除基本表

* 用SQL删除关系（表）
  * 将整个关系模式（表结构）彻底删除
  * Drop Table

# 数据定义——修改基本表结构

* 删除表中的某属性
  * 去除属性及相应的数据
  * Alter Table R Drop A

```
Alter Table student Drop sdept;
```

# 数据定义——修改基本表结构

* 增加表中的属性
  * 向已经存在的表中添加属性
  * 已有的元组中该属性的值被置为Null
  * Alter Table R Add A D

```
Alter Table student Add address char(30)
```

# 数据定义——修改基本表结构

* 修改表中属性
  * 修改表中属性的数据类型
  * 可能破坏原有的数据
  * Alter Table r Modify A D

```
Alter Table student Modify sname varchar(30)
```

# 数据定义——索引的建立与删除

* 建立索引

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

UNIQUE:　每一个索引值只对应唯一的数据记录

CLUSTER: 聚簇索引,索引项顺序与记录的物理顺序一致.

* 删除索引

DROP INDEX <索引名>

# 数据定义——数据添加

- 用SQL的插入语句，向数据库表中添加数据
  - 按关系模式的属性顺序

    Insert Into Student Values（‘95001’,‘张三’,‘M’,27,‘CS’）
  - 按指定的属性顺序，也可以只添加部分属性（非Null属性为必需）

    Insert Into Student（sno, sname, sage）Values（‘95002’,‘李四’,26）

# §3.3 查询

- 数据查询是数据库的核心操作
- Select

# 查询

只有查询条件为真，select 语句才有结果

* 基本结构

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

等价于：

$$\prod_{A_1, A_2, \ldots, A_n}(\sigma_{\rho}(R_1 \times R_1 \times \ldots \times R_m))$$

**Select** **Where** **From**

# 查询

- Select语句的含义
  - 对 From 子句中的各关系，作笛卡儿积
  - 对 Where 子句中的逻辑表达式进行选择（σ）运算，找出符合条件的元组
  - 根据 Select 子句中的属性列表，对上述结果作投影操作

# 查询

* 结果集
  * 查询操作的对象是关系，结果还是一个关系，是一个结果集，是一个动态数据集

# 查询

* 例：列出所有学生的学号及姓名

```
Select   sno,sname
From     student;
```

# 查询——单表查询

- 仅涉及一个表的查询
  - 从一个基本表中产生所需要的结果集
  - From子句中仅有一个表名
  - 无须进行笛卡儿积

# 单表查询——选择若干列

* 查询指定的列

  Select〈目标列表达式〉

可按照需求排列属性的顺序

例：查询全体学生的学号与姓名

```
Select    sno, sname
From student;
```

# 单表查询——选择若干列

* 查询全部列

  Select〈目标列表达式〉

  Select *

两者有何差异？

例：查询全体学生的详细信息

```
Select    *
From student;
```

# 单表查询——选择若干列

* 查询计算列

  Select〈目标列表达式（含有计算表达式）〉

例：查询学生的学号、姓名及出生年份

```
Select    sno, sname,  2020-sage
From student;
```

# 单表查询——选择若干列

* 改变结果集的列名

  Select 列名 as 别名,...

例：查询学生的学号、姓名

Select sno as ‘学号’ ,sname as ‘姓名’ From student;

用 as 让输出可读性增强。

# 单表查询——选择若干列

* 函数的使用
  * 在Select子句中，可使用相应的SQL函数（不同的DBMS可能有不同的函数集）

例：列出学生的学号、姓氏

```
Select sno as '学号',left(sname,1) as '姓氏'
From student;
```

# 单表查询——选择若干元组

- 根据条件进行选择操作
- 集合(Set)与包(Bag)
- Select结果集
  - 包（缺省）
  - 集合（使用Distinct短语），耗时

```
Select Distinct sno
From SC;
```

# 单表查询——选择若干元组

- 查询满足条件的元组：WHERE子句
- 形式：

  Select 列名,列名,...

  From 表名

  Where 条件表达式

- 使条件表达式的结果为真的元组才会被输出。

结果为‘真’或‘假’或“unknown”

# 单表查询——选择若干元组

| 查 询 条 件 | 谓　　词 |
| --- | --- |
| 比　　较 | =，>，<，>=，<=，!=，<>，!>，!<；<br>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 |

# 单表查询——选择若干元组

- 确定集合(组成员测试)
  - where 字段 in (value1,value2,……)
  - 等价或运算(OR)
- 例找出CS系或MA系或IS系的学生

Select * From student

Where sdept IN (‘CS’,‘MA’,‘IS’)

# 单表查询——选择若干元组

- 字符匹配(Like, %, _)
  - 检查包含字符串数据字段的值是否与指定的样式匹配
  - 使用Like运算符
  - %表示任何顺序的0个或多个字符
  - _表示任何一个单个字符

# 单表查询——选择若干元组

* 找出所有数据库课程

```
Select *
From course
Where cname Like '%数据库%'
```

# 单表查询——选择若干元组

* 找出所有张姓单名的学生

```
Select *
From student
Where sname Like '张_'
```

# 单表查询——空值

- 空值Null
  - 表示信息短缺、不知道、未提供
  - Null算术运算为Null
  - Null比较运算为unknown
  - 判断
    - 字段名 Is Null/Is Not Null
- 逻辑运算
  - True(1)、False(0)、Unknown1(1/2)
  - and(取小)、or(取大)、not(1-x)

如果 x= null, y=false,　(x > 1) OR y = ?

# 单表查询——选择若干元组

* 找出所有不需要先修课的课程

```
Select *
From course
Where cpno is null
```

* 找出所有提供了年龄的学生

```
Select *
From student
Where sage is not null
```

# 课堂问题：两个查询结果一样吗?

```
SELECT b
FROM R
WHERE a<10 OR a>=10;
```

| 1 |
|---|
| 2 |
| 4 |

| a | b |
|---|---|
| 5 | 1 |
| 11 | 2 |
| null | 3 |
| 0 | 4 |

```
SELECT b
FROM R;
```

| 1 |
|---|
| 2 |
| 3 |
| 4 |

# 单表查询——结果集的排序

* 对查询结果进行排序
* ORDER BY子句
  * ASC/DESC
* 子句中指明列名/列号
* 需要临时表空间的支持，耗时

# 单表查询

* 例:生成CS系学生的名册(要求包含学号、姓名、年龄,依次按照年龄、姓名排序)

```
Select sno,sname,sage
From student
Where sdept = 'CS'
Order by sage,sname
```

| SNO | sname | dept | sage |
|---|---|---|---|
| 001 | david | CS | 19 |
| 002 | mary | Null | 19 |
| 003 | lucy | EE | 20 |

默认是升序排列，降序要写DESC。

# 单表查询——聚集函数

* 主要用于数据的统计计算
* COUNT
* SUM
* AVG
* MAX
* MIN

# 单表查询

- COUNT(*)
  - 行数
- COUNT(distinct 字段名)
  - 指定字段中的不同取值的个数
- NULL
  - 被忽略、不计

# 单表查询

- SUM/AVG
  - 对数值型字段进行计算
  - NULL
    - 不参与、不计入
- MAX/MIN可以是数值、字符串、日期
  - 大小、字符码之大小、早晚
  - NULL不参与

# 单表查询——结果分组

- GROUP BY子句
  - 将结果表按一列或多列的值进行分组，值相等的为一组
  - 便于按分组的形式进行信息的统计
  - 空值可以成为一个组
- 大部分DBMS要求
  - Group By中的项，必须出现在Select子句中

# 单表查询

* 例：计算各系的学生人数

```
Select sdept, count(*)
From student
Group by sdept;
```

* 例：计算各系女生的人数

```
Select sdept, count(*)
From student
Where ssex='f'
Group by sdept;
```

Select 子句中出现的要么是group by属性，要么是聚集函数

# 单表查询——分组的筛选

* HAVING子句
  * 对分组后的结果表，按各组的统计值进行筛选，符合条件的组就是最后结果表中的元组
  * 以集函数作为表达式的主体
  * Having COUNT(*) >= 50

# 单表查询

* 例：找出学生人数超过50人的系以及人数

```
Select sdept, count(*)
From student
Group by sdept
Having count(*)>50
```

having 子句中出现的要么是group by属性，要么是聚集函数

# Where与Having的区别

* Where作用于基本表，从中选出符合条件的行
* Having作用于组，从中选出符合条件的组

# 查询语句——小结#

Select ...

From ...

Where ...

Group By ...

Having ...

Order By ...

# 课堂练习

* 建立下面4个关系表

**S(sno,sname,status,city) 供应商**
**P(pno,pname,color,weight) 零件**
**J(jno,jname,city) 工程项目**
**SPJ(sno,pno,jno,qty)**

* 求供应工程J1 零件P1 的供应商号码SNO
* 求供应工程J1 零件为红色的供应商号码SNO

* Create table S(sno char(3) primary key,sname char(10),status char(2),city char(10));
* Create table P(pno char(3) primary key,pname char(10),color char(4),weight int)
* Create table J(jno char(3) primary key,jname char(10),city char(10))
* Create table SPJ(sno char(3),pno char(3),jno char(3),qty int, primary key(sno,pno,jno), foreign key (sno) references s, foreign key(pno) references p)

* Select sno from spj where jno= ‘j1’ and pno= ‘p1’ ;

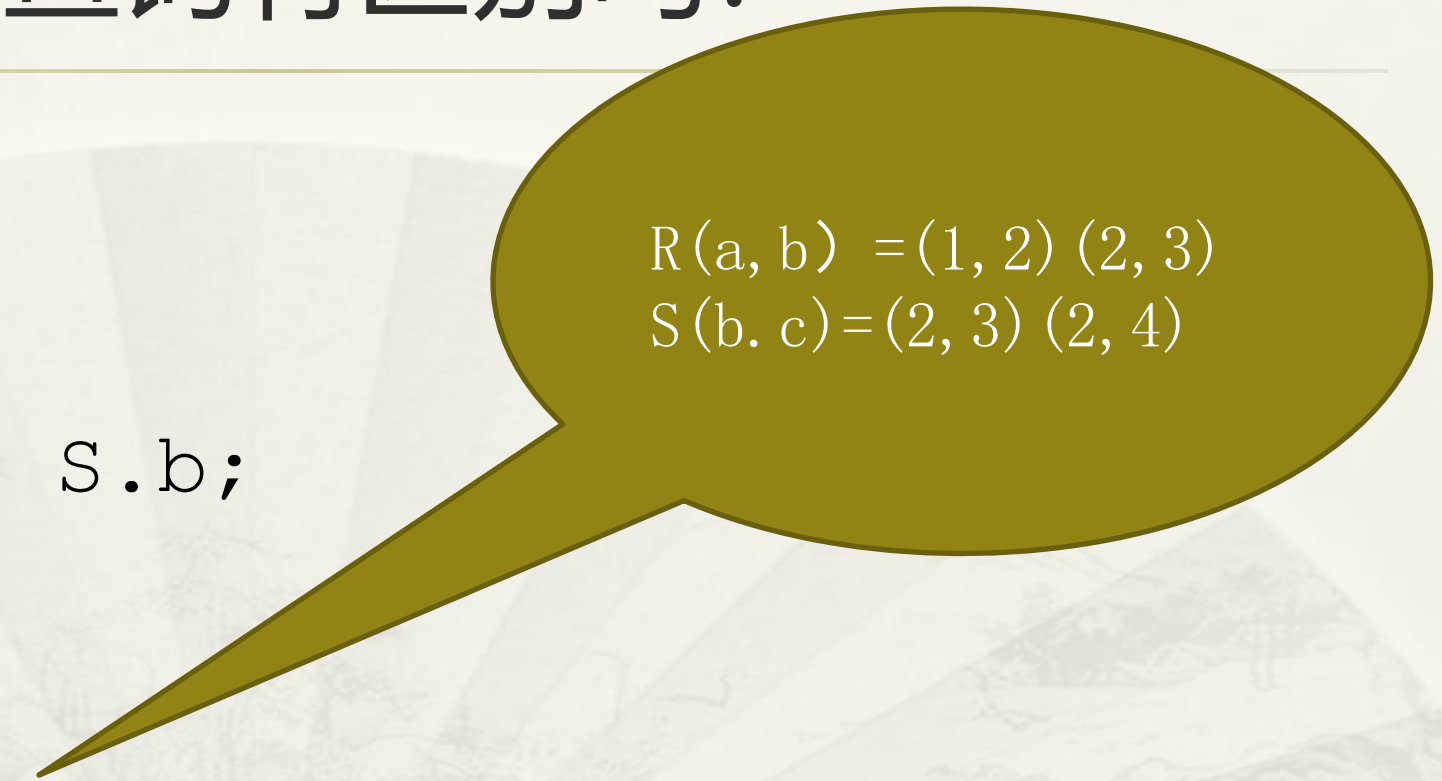
* Select sno from spj where jno= ‘j1’ and pno in (select pno from p where color= ‘red’ )

Or select sno from SPJ,P where jno= ‘j1’ and spj.pno=p.pno and color= ‘red’ ;

# 这两个查询有区别吗?

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S)
```



# 查询——多表查询

- 数据来自多表，查询涉及两个或以上的表，必须将多个表进行连接
- 广义连接
  - 将相关的表进行组合
  - 所有行进行组合，字段拼接，行交叉组合
  - 一般无意义

# 多表查询

- 条件连接
  - 在广义连接的结果中，施加条件，加以选择，留下符合要求的元组
- 条件连接、自然连接、自身连接、外连接、……

# 多表查询——条件连接

* 例：列出每个学生及其选修课程的情况

  学生情况←student

  选修课程情况←SC

```
Select student.*,SC.*
From student,SC
Where student.sno = SC.sno
```

# 多表查询——自身连接

* 例：列出每门课程的间接先修课程号

```
Select first.cno,second.cpno
From course first, course second
Where first.cpno = second.cno
```

# 多表查询——自身连接

* 例：列出年龄比95001大的学生的学号、姓名

```
Select s1.sno, s1.sname
From student s1, student s2
Where s2.sno='95001' AND s1.sage>s2.sage;
```

# 多表查询——自然连接

* 参与连接的表，在某些公共属性上具有相同值的元组
* 必须具有相同的属性列
* 在公共属性列上的等值连接

# 多表查询——自然连接

* 例：列出选修C01课程的学生的名册

Select sno,sname
From student, SC
Where student.sno = SC.sno and cno = ‘C01’

或者 select sno,sname from student natural join SC
where cno=‘C01’;

# 多表查询——外连接

* 不匹配连接，含有NULL信息的处理
* 例：列出学生及其选课情况

# 多表查询——外连接

* 例：列出学生及其选课情况

  所有的学生情况以及他们的选课情况（含未选课）

```
Select student.*, SC.cno, SC.grade
From student left outer join SC
```

| sno | sname | ssex | sage | sdept |
|---|---|---|---|---|
| 95001 | ZHANGSAN | M | 24 | CS |
| 95002 | LISI | F | 25 | CS |
| 95003 | ZHANGSAN | M | 27 | CS |
| 95004 | WANGWU | F | 24 | CS |
| 95005 | LISAN | M | 23 | CS |
| 96001 | JOHN | M | 24 | MA |
| 96002 | TOMAS | F | 21 | MA |
| 96003 | SMITH | M | 24 | MA |
| 96004 | LISI | M | 22 | MA |
| 96005 | JUILA | F | 24 | MA |

| sno | cno | grade |
|---|---|---|
| 95001 | C01 | 85 |
| 95001 | C02 | (NULL) |
| 95001 | C03 | (NULL) |
| 95001 | C04 | (NULL) |
| 95002 | C01 | 90 |
| 95003 | C01 | 78 |
| 95004 | C01 | 88 |
| 95002 | C02 | (NULL) |

# 多表查询——外连接

- 主要用于主表-从表之间信息短缺的处理
- 注意
  - 不同的数据库产品有不同的方言，甚至不支持

# 嵌套查询

* 查询块
  * Select-From-Where
* 嵌套查询(SubQuery)
  * 查询块的Where或Having中含有另一个查询块

# 嵌套查询——一个简单例子

* 找出与95001同岁的学生学号及姓名
  * 95001的年龄？

```
Select sage
From student
Where sno = '95001'
```

  * 与上述年龄相等的学生？

```
Select sno,sname From student
Where sage = (Select sage From student Where sno = '95001')
```

# 嵌套查询——带IN的子查询

- 子查询返回结果集（多行、单列）
- 主查询与子查询之间由IN连接
- 几乎所有的DBMS均支持多行单列的IN操作
- 有些DBMS支持多行多列的IN操作
  - 需要了解DBMS的性能

# 嵌套查询——带IN的子查询

* 例：找出选修数据库课程的学生的学号
  * 数据库课程←course表
    * cname Like ‘%数据库%’
    * 获得相应课程的课程号
  * 选修课程信息←SC表
    * 由cno查询出相应的选课学号

# 嵌套查询——带IN的子查询

```
Select sno
From SC
Where cno IN (Select cno
              From course
              Where cname Like '%数据库%')
```

# 嵌套查询——带IN的子查询

- 例：找出选修数据库课程的学生的学号、姓名
  - 数据库课程←course表
    - cname Like ‘%数据库%’
    - 获得相应课程的课程号
  - 选修课程信息←SC表
    - 由cno查询出相应的选课学号
  - 学生的姓名←STUDENT表
    - 由sno查得相应的姓名

# 嵌套查询——带IN的子查询

```
Select sno,sname
From student
Where sno IN (
   Select sno
   From SC
   Where cno IN (
      Select cno
      From course
      Where cname Like '%数据库%'))
```

# 嵌套查询——带IN的子查询

* 表达查询最自然的方式
* 将复杂的查询切分成若干块，逐个解决
* Order By只能作用于主查询

# 嵌套查询——带比较的子查询

- 子查询与主查询之间由比较运算符相连接
- 单值（单行、单列）
  - 直接使用比较运算符
  - >、<、=、>=、<=、!=等
- 多值（多行、单列）
  - 比较运算符和ANY/ALL连用

# 嵌套查询——带比较的子查询

* 例：找出年龄比95001学生大的所有学生的学号、姓名
  * 95001的年龄
  * 年龄比他大的学生

```
Select sno,sname
From student
Where sage > (Select sage
              From student
              Where sno = '95001')
```

# 嵌套查询——带ANY/SOME/ALL的子查询

- 当子查询返回多值结果集（多行单列）时，比较操作须由比较运算符以及ANY/ ALL
- ANY/SOME
  - 子查询结果集中的某一个
- ALL
  - 子查询结果集中的所有

# 嵌套查询———带ANY/SOME/ALL的子查询

* 例：找出比最小年龄大的学生
  * 学生年龄←Select sage From student
  * 不是最小年龄←>some 学生年龄

```
Select * From student
Where sage >some (Select sage
                  From student)
```

# 嵌套查询——带ANY/SOME/ALL的子查询

* 例：找出年龄最小的学生
  * 学生年龄←Select sage From student
  * 最小年龄←<=All 学生年龄

```
Select * From student
Where sage <=All (Select sage
                  From student)
```

# 嵌套查询——带Exists的子查询

* 表示存在，判断子查询的返回结果集是否为空集
  * 子查询结果集为空集：False
  * 子查询结果集含有元组：True
  * 子查询只需使用 Select *

# 嵌套查询——带Exists的子查询

* 例：列出至少选修一门课程的学生的学号、姓名

```
Select sno,sname
From student
Where Exists (
        Select *
        From SC
        Where student.sno = SC.sno )
```

# 嵌套查询——相关子查询

* 内层子查询的条件引用外层主查询的某些属性
* 在Exists运算中，更多到涉及到相关子查询的使用
* 前例便是相关子查询的典型使用

# 嵌套查询——相关子查询

* 找出与95001在同一个系的学生

```
Select *
From student st1
Where Exists (
      Select *
      From student st2
      Where st2.sdept = st1.sdept and  st2.sno = ‘95001’)
```

# 嵌套查询——相关子查询

* 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 student.sno = SC.sno and SC.cno = '1')
```

# 嵌套查询——相关子查询

* 查询选修了所有课程的学生姓名

```
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))
```

不存在这样的一门课程，这个学生没有选修。

# 课堂练习:
# 求没有使用天津供应商生产的红色零件的工程项目jno

```
Select jno from J
Where not exists
        (select * from spj
         where spj.jno=j.jno and sno in
                (select sno from s where city= ‘tianjing’ )
and pno in
                (select pno from p where color= ‘red’ ) );
```

# 集合查询

- 并、交、差集合操作。
- 对多个查询的结果集实施集合操作
- 属性必须相容
- ORDER BY只能施加在整个结果集中
- 并不是所有的DBMS均支持三个操作

# 集合查询

* 查询计算机系以及数学系的学生

```
Select * From student Where sdept = ‘CS’
Union
Select * From student Where sdept = ‘IS’;
```

# 集合查询

* 列出雇员和客户的姓名
  * 可以来自不同的表
  * 只要属性相容(列数和数据类型相同)即可

```
Select emp_name From employee
Union
Select cus_name From customer
```

# §3.4 数据更新

* 增（Insert）
* 删（Delete）
* 改（Update）

# 数据更新——增Insert

- 插入单行
  - 按关系模式的属性顺序

```
Insert Into Student
        Values ( '95001','张三' ,'M',27,'CS' )
```

  - 按指定的属性顺序，也可以只添加部分属性（非Null属性为必需），其余属性值为NULL

```
Insert Into Student ( sno, sname, sage)
        Values ( '95002','李四' ,26 )
```

# 数据更新——增Insert

* 插入多行
  * 将子查询的结果插入数据库表中

  Insert Into〈表名〉[属性列表] 子查询

```
Insert into stu_cs
  Select *
  From student
  Where sdept = 'CS'
```

# 缺省值

* Create table students (sno char(8) primary key, sname char(8), age int default 18, ssex char(2), dept char(2));
* Insert into students (sno,sname) values ('02','王东');

→ (02 王东 18 null null) 被插入到学生表中

# 数据更新——删除Delete

* 删除符合条件的元组(使得表达式为真的元组)

```
Delete
From Student
Where sage <=15
```

```
Delete
From Student
Where sage = (Select sage From student
                Where sno = '95001')
```

# 数据更新——删除Delete

* 删除所有元组

```
Delete
From Student
```

学生表仍然存在，只是没有元组。

# 数据更新——删除Delete

* 不能同时对多表进行删除操作
* 删除操作实现的步骤

1. 查找满足条件的元组
2. 删除该元组

# 例如

* create table j (jno char(2) primary key, jname string, city string);
* insert into j values ('j1','Jiaoda','shanghai');
* insert into j values ('j2','Fudan','shanghai');
* insert into j values ('j3','qinghua','Beijing');

* Delete from j where exists
(select * from j jj where jj.city=j.city and jj.jname<>j.jname);

* 删除一个元组还是二个元组?

# 数据更新——修改Update

* 格式

  update〈表名〉

  Set〈列名〉=〈表达式〉[, .....]

  Where〈条件表达式〉

# 数据更新——修改Update

* 修改符合条件的元组
  * 对使得条件表达式成立的元组的相应属性进行修改

例:将所有学号前缀为95的学生转至IS系

```
Update student
set sdept = 'IS'
Where sno Like '95%'
```

# 数据更新——修改Update

- 子查询在UPDATE中的使用
  - 作为Where子句中的子查询
  - 作为Set子句中的新值
- 例:将男生的年龄改为所有学生的平均年龄

```
Update student
set sage = (Select AVG(sage)
            From student)
Where ssex = 'M'
```

仅作举例

# 数据更新——修改Update

* Update不能对多表进行修改

# 有参照完整性情况下的数据更新

- 数据更新引起参照完整性违约的情况：
- 从表
  - 在从表中插入外码值
  - 在从表中修改外码值
- 主表
  - 在主表中修改主码（已被参照）
  - 在主表中删除元组（已被参照）

# 数据更新——异常

- 级联操作
  - 删除主表元组，级联删除从表的参照元组
  - 修改主表元组，级联修改从表中的外码值
- 禁止操作
  - 不得更改已具有参照关系的数据
- Set Null
  - 使得从表的外码被置为Null
- Set Default
  - 使得从表的外码被置为其缺省值

# Creating Foreign Keys

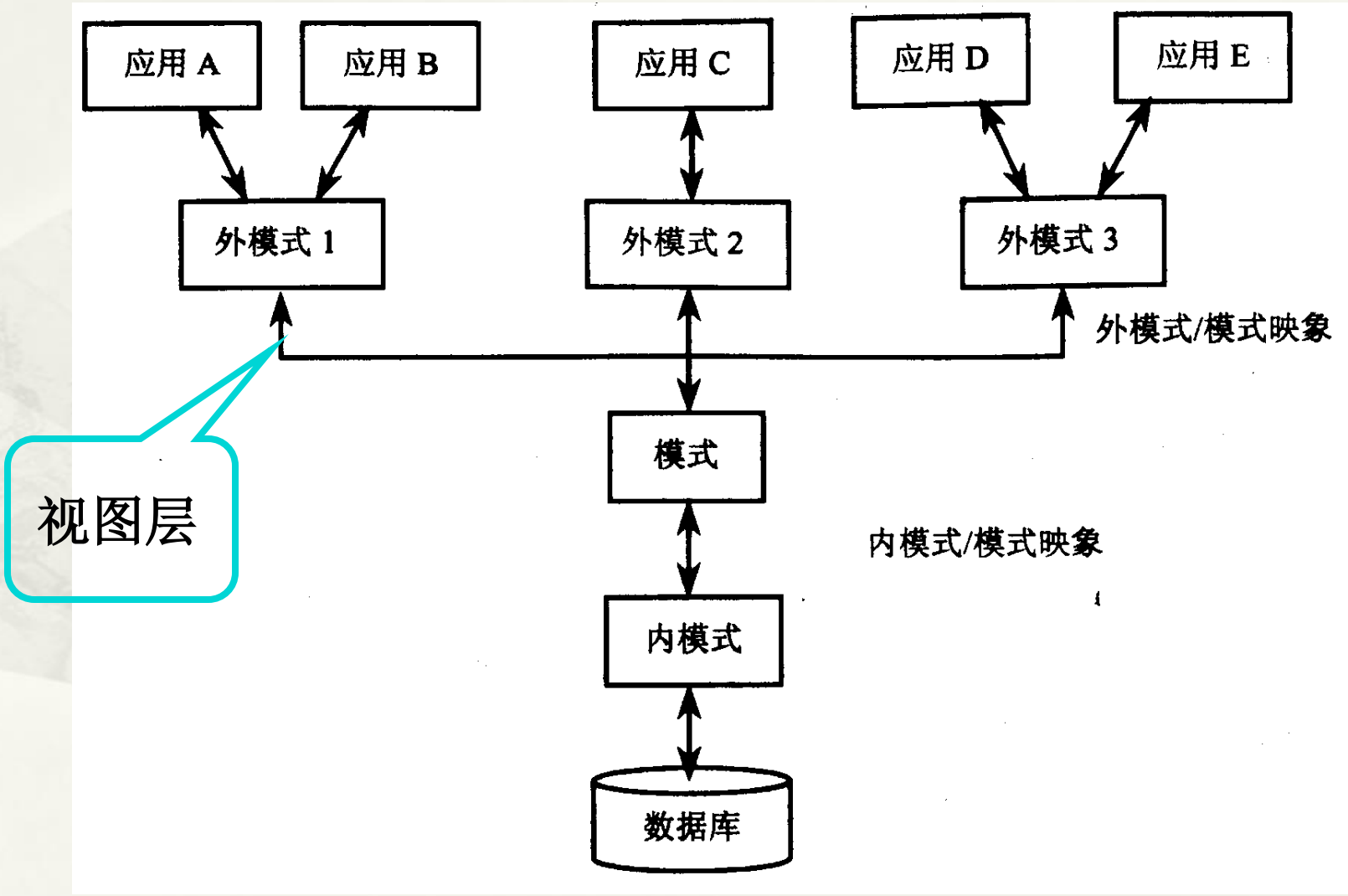
* Example

```
CREATE TABLE department
   (dept_id          INTEGER NOT NULL,
   dept_name         CHAR ( 20 ),
   dept_id_head      INTEGER,
   FOREIGN KEY (dept_id_head)
   REFERENCES employee
   ON UPDATE CASCADE
  ON DELETE SET NULL);
```

# §3.5 视图（View）

- 用户模式、外模式
- 视图是一个虚表，是从一个或几个基本表导出的表
- DB中存放的是视图的定义（数据来源于基本表的查询）

# 视图（View）


应用 A
应用 B
应用 C
应用 D
应用 E
外模式 1
外模式 2
外模式 3
外模式/模式映象
模式
内模式/模式映象
内模式
数据库
视图层

# 视图 —— 定义格式

Create View <视图名> [(<列名>,...)]
As <子查询>

# 视图——例子

* Create View stu_cs(no,name) as

  Select sno,sname

  From student

  Where sdept = 'CS';

- *Select * From stu_cs*
  * 将对视图的操作，转换成基本表的操作

    Select sno,sname

    From student

    Where sdept = 'CS'

# 视图——查询

* 在查询中，视图基本可以作为基本表使用
  * 大部分DBMS不支持聚集函数字段的视图

# 视图——更新

- 来自单表的基本属性可以修改
- 来自多表的视图不得更新
- 计算属性构成的视图，不得修改
- 视图中的元组非自身元组、属性均不得改

# 视图——更新 (实例)

```
INSERT INTO  Stu_cs
Values ( “905234”,”张三”)
```

**系统转换为:**

```
INSERT INTO  student (sno, sname, sdept)
Values ( “905234”,”张三”, “CS”)
```

# 视图——作用

* 安全
* 简便
* 数据独立性

# 本章小结

关系数据库中, 如何进行

- 数据定义
- 数据更新
- 数据查询

# 课堂练习（更新）

- 假设已创建数据库：
- 供应商表S(SNO, SNAME, STATUS, CITY)
- 零件表 P(PNO, PNAME, COLOR, WEIGHT)
- 项目表 J(JNO, JNAME, CITY)
- 三者关系表 SPJ (SNO, PNO, JNO, QTY)

Insert into S values (‘s10’, ‘宏达’, null, ‘Shanghai’);

Update SPJ set QTY=350 where sno=‘s1’ and jno=‘j1’ and pno=‘p1’;

delete from s where sno = ‘s2’;
Delete from SPJ where sno=‘s2’;

更新操作：

- 新增供应商（S10，‘宏达’，null，‘Shanghai’）
- 更新S1供应商供给工程J1零件P1数量为350；
- 从供应商关系中删除S2记录，并从供应关系中删除相应的记录

# 实践操作

- http://www.cs.sjtu.edu.cn/~li-fang/DB2.htm 下载 sqlite 和 SPJ
- 双击SQLite，在命令行上输入：.read spj.sql
- 查询以下内容

# 课堂练习 (查询)

- 查询不使用天津’tianjing’供应商供应的红色’red’零件的工程项目(jno)
- 查询这样的工程项目号:

供给该工程项目的零件P1的平均供应量大于供给工程项目J1的任何一种零件的最大供应量.

- 定义一个视图,它由所有具有这种特点的工程项目(项目号,所在城市名称)所组成:它们由供应商S1供货且使用零件P1.

# 课堂练习答案

```
Select distinct jno from spj
 where pno= ‘p1’
 group by jno having avg(qty) >
   (select max(qty) from spj where jno= ‘j1’ );

Create view j_s1p1 as select j.jno, j.city from spj,j
where spj.jno=j.jno and spj.sno= ‘s1’  and
spj.pno= ‘p1’ ;
```

```
sqlite> select jno from j where not exists (select * from spj where spj.jno=j.jno and sno in (select
sno from s where city='tianjing') and pno in (select pno from p where color='red'));
jno
----------
j3
sqlite> select distinct jno from spj where pno='p1' group by jno having avg(qty)>(select max(qty) fro
m spj where jno='j1');
jno
----------
j2
sqlite>
```

# 系统演示

* 使用sqlite 开源软件,命令行方式
* 查询例题
* 插入数据
* 查询操作
* 更新操作