



第三章 SQL 语言



- 学习内容

- 3.1 SQL概述
- 3.2 SQL数据定义功能
- 3.3 SQL数据查询功能
- 3.4 SQL数据修改功能
- 3.5 SQL数据控制功能
- 3.6 嵌入式SQL



- 学习目标

- 掌握SQL语言的数据定义功能，能够增加、修改、删除表（模式）
- 掌握SQL语言的数据查询功能，包括基本查询、复杂查询以及嵌套子查询的使用
- 掌握SQL语言的数据更新功能
- 掌握SQL语言的数据控制功能
- 了解嵌入式SQL语言



3.1 SQL概述

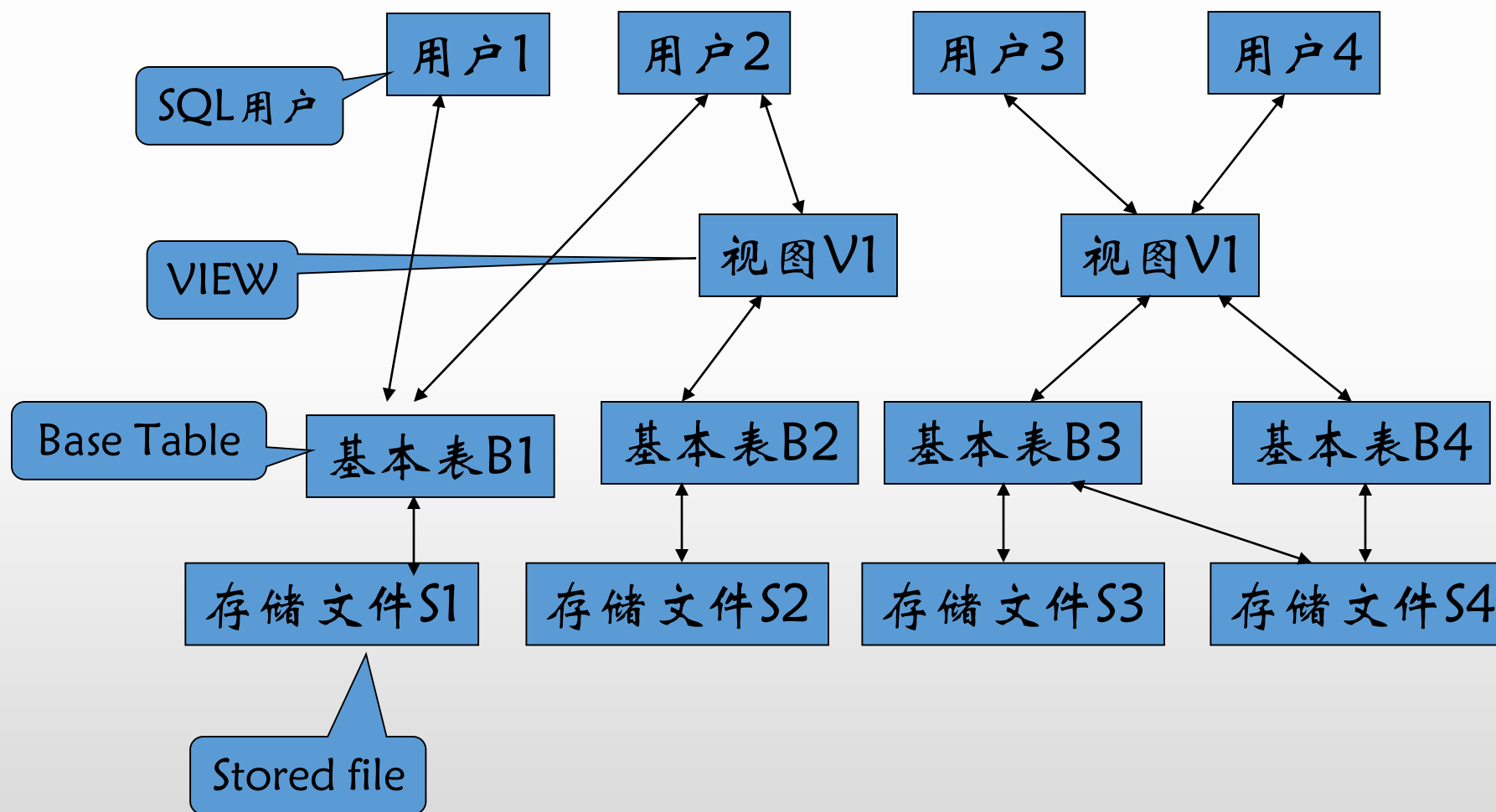
- 3.1.1 历史及其相关标准
- 3.1.2 SQL数据库体系结构
- 3.1.3 SQL 特点



3.1.1 SQL历史及其相关标准

- SQL: Structured Query Language
 - 交互式SQL
 - 嵌入式SQL
 - 调用式SQL（存储过程）
- 标准化的有关组织：
 - ANSI(American National Standard Institute)
 - ISO(International Organization for Standardization)
- 相关标准
 - SQL-86
 - SQL-89
 - SQL-92
 - SQL-3

3.1.2 SQL数据库体系结构





3.1.3 SQL 语言特点

- 一体化
- 面向集合的操作方式
- 高度非过程化
- 两种使用方式，统一的语法结构
- 语言简洁，易学易用

SQL功能	操作符
数据查询	SELECT
数据定义	CREATE, ALTER, DROP
数据操纵	INSERT, UPDATE, DELETE
数据控制	GRANT, REVOKE



和其他语言比较示例

Suppliers(sno,sname,status,city)

查询不在伦敦居住的供应商的情况

SQL实现:

SELECT *

```
// take all columns
```

FROM S

```
// from table s
```

```
WHERE city<>'London'
```

// and show all rows in which the city is

not London

QUEL实现:

RANGE OF s IS S

```
// variable 's' works on
```

table S

RETRIEVE (s.all)

```
// take all columns from
```

range of s

```
WHERE (s.city<>"London")
```

// and show all rows in which the

city is not London



3.2 SQL数据定义功能

- 3.2.1 域定义
- 3.2.2 基本表的定义
- 3.2.3 索引的定义
- 3.2.4 数据库的建立与撤消
- 3.2.5 SQL数据定义特点



3.2 SQL数据定义功能(续)

• 3.2.1 域定义

• 1. 域类型 (SQL 92)

- char (n) : 固定长度的字符串
- varchar (n) : 可变长字符串
- int: 整数
- smallint: 小整数类型
- numeric (p, d) : 定点数, 小数点左边p位, 右边q位
- real: 浮点数
- double precision: 双精度浮点数
- date: 日期 (年、月、日)
- time: 时间 (小时、分、秒)
- interval: 两个date或time类型数据之间的差



3.2 SQL数据定义功能(续)

- 3.2.1 域定义

- 2. 定义

- 格式

create domain 域名 数据类型

- 示例

create domain *person-name* char (20)

- 类似C语言中:

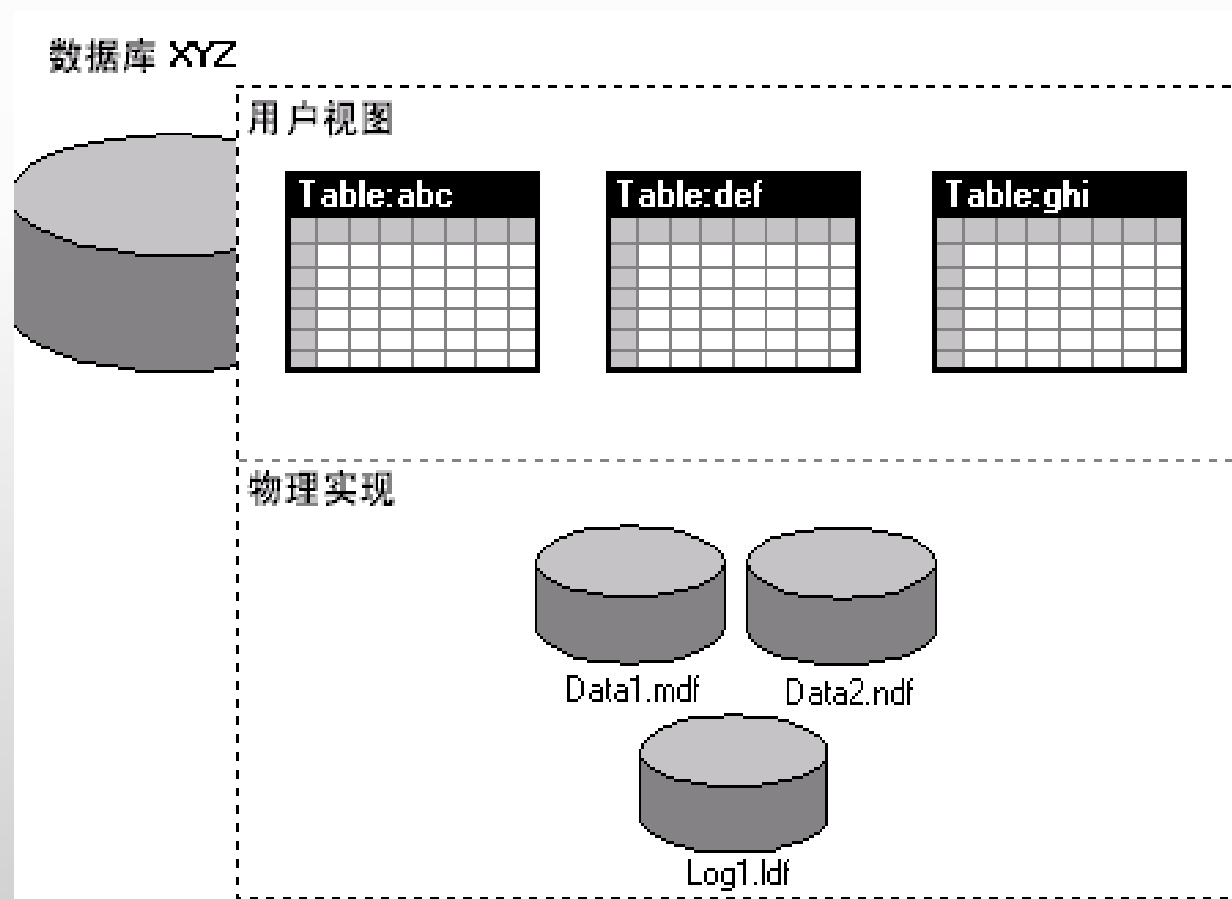
```
typedef ADDRESS_LIST{  
    char name[10];  
    char telephone[20];  
    char location[20];  
    char email[20];  
}
```

ADDRESS_LIST tom;

3.2 SQL数据定义功能(续)

- 3.2.2 基本表的定义

- 1. 基本表的概念





3.2 SQL数据定义功能(续)

- 3.2.2 基本表的定义

- 2. 基本表定义格式 (SQL 92)

create table 表名 (

列名 数据类型 [***default*** 缺省值] [***not null***]

[, 列名 数据类型 [***default*** 缺省值] [***not null***]]

.....

[, ***primary key*** (列名 [, 列名] ...)]

[, ***foreign key*** (列名 [, 列名] ...)

references 表名 (列名 [, 列名] ...)]

[, ***check*** (条件)]]



基本表的定义示例 I

S:

学号: char 4; 姓名: char 8; 年龄: smallint; 性别: char 1;

主关键字: 学号; 性别: 只能取 0 或者 1

学生基本表: **CREATE TABLE** S
(Sno **CHAR** (4) ,
SNAME **CHAR** (8) **NOT NULL**,
AGE **SMALLINT**,
SEX **CHAR** (1) ,
PRIMARY KEY(Sno),
CHECK (SEX='0' OR SEX='1')
)



基本表的定义示例 II

关系课程C:

课程号: char 4; 课程名称: char 10 非空;

任课教师姓名: char 8;

主关键字: 课程号

课程基本表: **CREATE TABLE C**
 (Cno **CHAR** (4) ,
 CNAME **CHAR** (10) **NOT NULL**,
 TEACHER **CHAR** (8) ,
 PRIMARY KEY(Cno),
)



基本表的定义示例 III

选课关系SC:

学生编号Sno, char 4; 课程编号Cno, char 4, 成绩Grade smallint;

主关键字: Sno,Cno; 外键: 关系S的属性Sno, 关系C的属性Cno;

约束: $0 \leq \text{成绩} \leq 100$ 或者为Null

```
CREATE TABLE SC  
  (Sno CHAR (4) ,  
   Cno CHAR (4) ,  
   GRADE SAMLLINT,  
   PRIMARY KEY(Sno, Cno),  
   FOREIGN KEY (Sno) REFERENCES S(Sno),  
   FOREIGN KEY(Cno) REFERENCES C(Cno),  
   CHECK((GRADE IS NULL) OR  
          GRADE BETWEEN 0 AND 100))
```




基本表的定义示例 IV

```
CREATE DOMAIN person_name char(20)
```

```
CREATE TABLE PROF  
  ( PNO  char[10],  
    person_name PNAME not null,  
    SAL  int,  
    AGE  int,  
    DNO  char[10],  
    primary key (PNO),  
    foreign key (DNO)  
      references DEPT(DNO),  
    check (SAL > 0)  
  )
```

SQL SERVER 参考



CREATE TABLE

```
[ database_name. [ owner ] . | owner. ] table_name  
( { < column_definition >  
    | column_name AS computed_column_expression  
    | < table_constraint > ::= [ CONSTRAINT constraint_name ] }  
    | [ { PRIMARY KEY | UNIQUE } [ , ... n ]  
)
```

```
[ ON { filegroup | DEFAULT } ]
```

```
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]
```

SQL SERVER 参考



```
< column_definition > ::= { column_name data_type }  
    [ COLLATE < collation_name > ]  
    [ [ DEFAULT constant_expression ]  
      | [ IDENTITY [ ( seed , increment ) [ NOT FOR  
REPLICATION ] ] ]  
    ]  
    [ ROWGUIDCOL ]  
    [ < column_constraint > ] [ ...n ]
```



SQL SERVER 参考

- `< column_constraint > ::= [CONSTRAINT constraint_name]`
 `{ [NULL | NOT NULL]`
 `| [{ PRIMARY KEY | UNIQUE }`
 `[CLUSTERED | NONCLUSTERED]`
 `[WITH FILLFACTOR = fillfactor]`
 `[ON { filegroup | DEFAULT }]]`
 `]`
 `| [[FOREIGN KEY]`
 `REFERENCES ref_table [(ref_column)]`
 `[ON DELETE { CASCADE | NO ACTION }]`
 `[ON UPDATE { CASCADE | NO ACTION }]`
 `[NOT FOR REPLICATION]`
 `]`
 `| CHECK [NOT FOR REPLICATION]`
 `(logical_expression)`
 `}`



3.2 SQL数据定义功能(续)

- 3.2.2 基本表的定义

- 3. 说明

- 命名规范，约定俗成
 - 恰当的数据类型
 - 体现数据完整性



3.2 SQL数据定义功能(续)

- 3.2.2 基本表的定义

- 4. 修改基本表定义 (ALTER)

- 格式

ALTER TABLE 表名

[**ADD** 子句]

增加新列

[**DROP** 子句]

删除列, 约束

[**MODIFY** 子句]

修改列定义

- 示例

ALTER TABLE Prof **ADD** Location char[30]

ALTER TABLE Prof **DROP** Location

- 说明

3.2 SQL数据定义功能(续)

• 3.2.2 基本表的定义

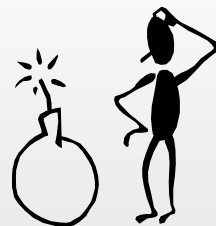
• 5. 删除基本表定义 (DROP)

• 格式

DROP TABLE 表名

• 示例

DROP TABLE DEPT



• 说明

- 撤消基本表后，基本表的定义、表中数据、索引、以及由此表导出的视图的定义、触发器、权限等都被删除



3.2 SQL数据定义功能(续)

- 3.2.3 索引的定义

- 1. 索引的概念

- **簇索引**

- **非簇索引**

- 2. 索引定义格式

- 3. 索引的删除

- 4. 其他说明



3.2 SQL数据定义功能(续)

• 3.2.3 索引的定义

• 2. 索引定义格式

CREATE [**UNIQUE**] [**CLUSTER**] **INDEX** 索引名
ON 表名 (列名 [**asc**/**desc**] [, 列名 **asc**/**desc**]]…)

- **unique (distinct)** : 唯一性索引, 不允许表中不同的行在索引列上取相同值。
- **cluster**: 聚簇索引(聚集索引)
- **asc/desc**: 索引表中索引值的排序次序, 缺省为asc



3.2 SQL数据定义功能(续)

- 3.2.3 索引的定义

- 2. 索引定义格式

- 示例

```
CREATE CLUSTER INDEX s-index on S (Sno)
```

- 3. 索引的删除

- 格式

drop index 索引名



3.2 SQL数据定义功能(续)

- 3.2.3 索引的定义

- 4. 其他说明

- 动态定义索引

- 应在使用频率高的、经常用于连接的列上建索引

- 一个表上可建多个索引

- 索引可以提高查询效率，但索引过多耗费空间，且降低了插入、删除、更新的效率



3.2 SQL数据定义功能(续)

• 3.2.4 数据库的建立与撤消

- 有的DBMS支持多库。
 - 通常，数据库由包含数据的表集合和其它对象（如[视图](#)、[索引](#)、[存储过程](#)和[触发器](#)）组成。

- 建立一个新数据库

create database 数据库名

- 撤消一个数据库

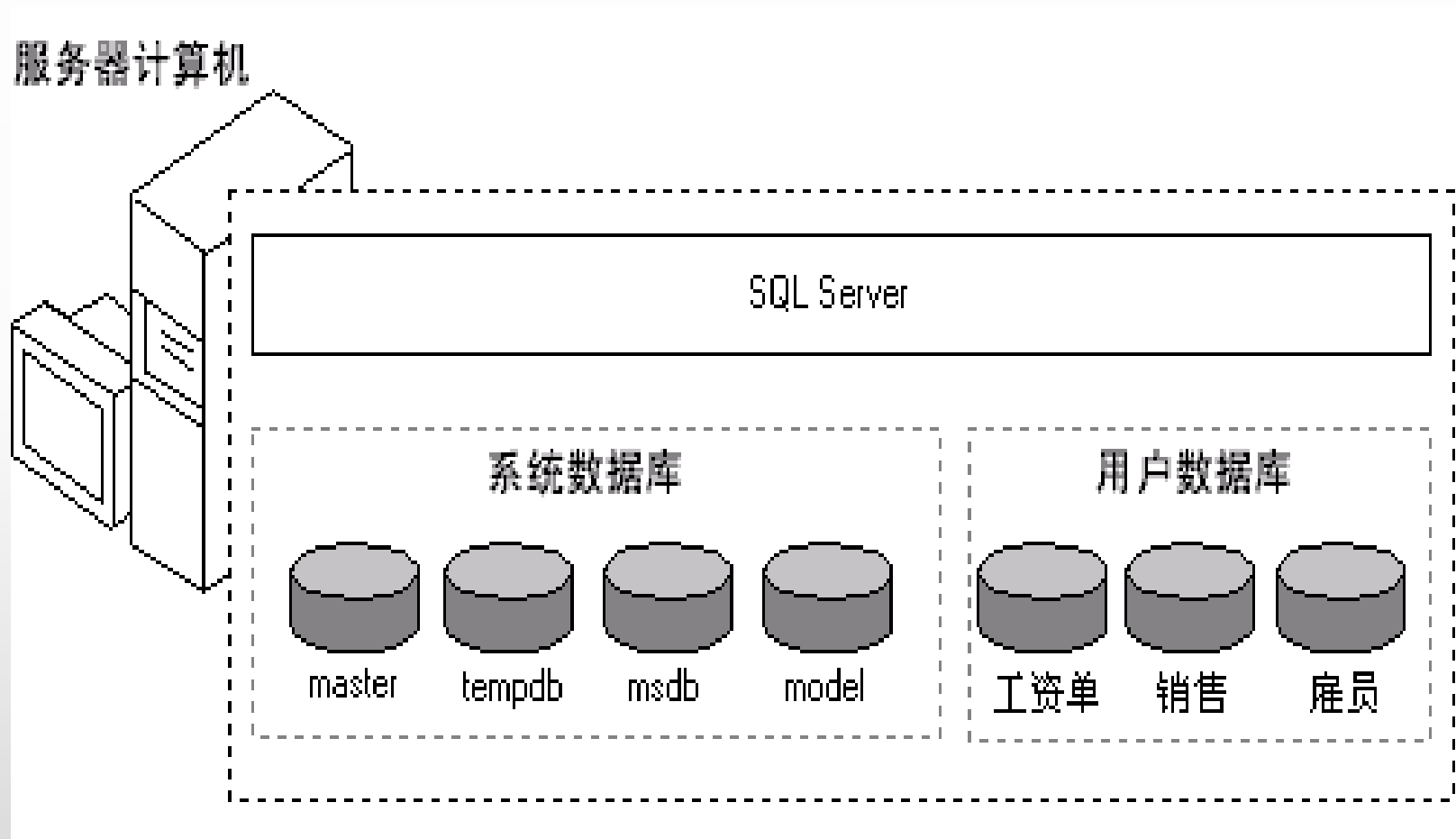
drop database 数据库名

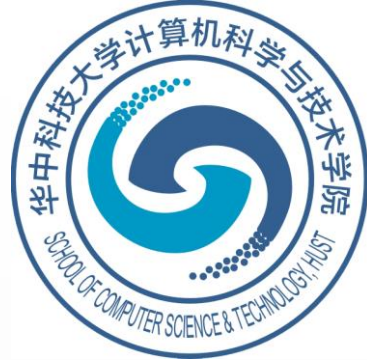
- 指定当前数据库

database 数据库名

use 数据库名

示例





SQL Server Enterprise Manager

控制台(C) 窗口(W) 帮助(H)

控制台根目录\Microsoft SQL Servers\SQL Server 组\DATA-SERVER (Windows NT)\数据库\pytemp\表

操作(A) 查看(V) 工具(T)

树

- Microsoft SQL Servers
 - SQL Server 组
 - 202.114.20.224 (Windows NT)
 - DATA-SERVER (Windows NT)
 - 数据库
 - 1009
 - 20021111_evening
 - 20021111evening
 - 200211202test_xwzdk
 - 20021203test
 - 20021203test-xwzdk
 - 20021203test_xwzdk
 - 20030102jhb
 - cj_test
 - dbforer
 - ems
 - jcj
 - jcpj
 - library
 - master
 - model
 - msdb
 - Northwind
 - origin
 - originyuanxi
 - ourgroup
 - pubs
 - pyjh
 - pytemp
 - 关系图
 - 表
 - 视图

表 21个项目

名称	所有者	类型	创建日期
dtproperties	dbo	系统	2003-3-20 20:31:54
SC	dbo	用户	2003-3-20 20:32:29
syscolumns	dbo	系统	2000-8-6 1:29:12
syscomments	dbo	系统	2000-8-6 1:29:12
sysdepends	dbo	系统	2000-8-6 1:29:12
sysfilegroups	dbo	系统	2000-8-6 1:29:12
sysfiles	dbo	系统	2000-8-6 1:29:12
sysfiles1	dbo	系统	2000-8-6 1:29:12
sysforeignkeys	dbo	系统	2000-8-6 1:29:12
sysfulltextcatalogs	dbo	系统	2000-8-6 1:29:12
sysfulltextnotify	dbo	系统	2000-8-6 1:29:12
sysindexes	dbo	系统	2000-8-6 1:29:12
sysindexkeys	dbo	系统	2000-8-6 1:29:12
sysmembers	dbo	系统	2000-8-6 1:29:12
sysobjects	dbo	系统	2000-8-6 1:29:12
syspermissions	dbo	系统	2000-8-6 1:29:12
sysproperties	dbo	系统	2000-8-6 1:29:12
sysprotects	dbo	系统	2000-8-6 1:29:12
sysreferences	dbo	系统	2000-8-6 1:29:12
systypes	dbo	系统	2000-8-6 1:29:12
sysusers	dbo	系统	2000-8-6 1:29:12



SQL Server 参考

```
CREATE DATABASE database_name
[ ON
    [ < filespec > [ ,...n ] ]
    [ , < filegroup > [ ,...n ] ]
]
[ LOG ON { < filespec > [ ,...n ] } ]
[ COLLATE collation_name ]
[ FOR LOAD | FOR ATTACH ]
```

< filespec > ::=

```
[ PRIMARY ]
( [ NAME = logical_file_name , ]
  FILENAME = 'os_file_name'
  [ , SIZE = size ]
  [ , MAXSIZE = { max_size | UNLIMITED } ]
  [ , FILEGROWTH = growth_increment ] ) [ ,...n ]
```

< filegroup > ::=

```
FILEGROUP filegroup_name < filespec > [ ,...n ]
```



3.2 SQL数据定义功能(续)

• 3.2.5 SQL数据定义特点

- 任何时候都可以执行一个数据定义语句，随时修改数据库结构。
- 数据库定义不断增长
- 数据库定义随时修改
- 可进行增加索引、撤消索引的实验，检验其对效率的影响



3.3 SQL数据查询功能

- 3.3.1 概述
- 3.3.2 简单查询
- 3.3.3 嵌套查询
- 3.3.4 集合查询
- 3.3.5 视图



3.3.1 概述

$$\text{subquery} ::=$$
[illegible]

FROM tableref{, tableref...}

[WHERE search_condition]

[GROUP BY colname{, colname...}]

```
[HAVING search_condition];
```

$$| \text{ subquery } \{ \text{UNION } [\text{ALL}] \mid \text{INTERSECT } [\text{ALL}] \mid \text{EXCEPT } [\text{ALL}] \} \text{ subquery}$$

select statement ::=

```
subquery [ORDER BY result_column [ASC| DESC] {, result_column[ASC | DESC]...}]
```



示例关系

DEPT(Dno , DNAME , DEAN)

S(Sno , SNAME , SEX , AGE , Dno)

COURSE(Cno , CN , PCno , CREDIT)

SC(Sno , Cno , SCORE)

PROF(Pno , PNAME, AGE, Dno , SAL)

PC(Pno , Cno)



3.3.2 简单查询

- 1. SQL数据查询基本结构
- 2. SELECT子句
- 3. 重复元组的处理
- 4. FROM子句
- 5. WHERE子句
- 6. 更名运算
- 7. 字符串操作
- 8. 元组显示顺序
- 9. 分组与聚集函数
- 10. 空值处理
- 11. TopK



3.3.2 简单查询 (续)

- 1. SQL数据查询基本结构

SELECT A_1, A_2, \dots, A_n

FROM R_1, R_2, \dots, R_m

WHERE P

\Leftrightarrow

$$\prod A_1, A_2, \dots, A_n(\sigma_p(R_1 \times R_2 \times \dots \times R_m))$$



3.3.2 简单查询（续）

- 示例

Ex: 查询所有老师的姓名

```
SELECT pname FROM Prof
```

\Leftrightarrow

$\bigwedge pname(Prof)$



3.3.2 简单查询 (续)

- 2. SELECT子句
 - 目标列形式
 - 列名, *, 算术表达式, 聚集函数

Ex 1: 查询所有老师的证件号, 姓名, 年龄

```
SELECT pno, pname, age  
FROM Prof
```

Ex 2: 查询所有老师的信息

```
SELECT * FROM Prof
```

Ex 3: 查询所有院系的信息

```
SELECT * FROM Dept
```



3.3.2 简单查询 (续)

- 2. SELECT子句

- 目标列形式

- 列名, *, 算术表达式, 聚集函数

Ex 4 : 查询所有老师的姓名及税后工资额

```
SELECT pname, sal * 0.9  
FROM Prof
```

Ex 5: 查询所有老师的平均工资

```
SELECT AVG(sal) FROM Prof
```




3.3.2 简单查询（续）

- 3. 重复元组的处理

- 语法约束

- 缺省为保留重复元组，也可用关键字**ALL**显式指明。
- 若要去掉重复元组，可用关键字**DISTINCT**指明

Ex: 查询所有选修课程的学生

```
SELECT DISTINCT sno  
FROM SC
```



3.3.2 简单查询 (续)

- 4. FROM 子句

- 说明

- FROM 子句列出查询的对象表
 - 当目标列取自多个表时，在不混淆的情况下可以不用显式指明来自哪个关系

–Ex: 查询职工的姓名、工资、系别

```
SELECT ALL pname , sal , dname  
FROM      Prof , Dept  
WHERE     Prof.DNO = Dept.DNO
```



3.3.2 简单查询 (续)

- 连接查询说明

- 示例

- Agents(aid, aname, city, percent)
- Customers(cid, cname, city, discount)
- Products(pid, pname, city, quantity, price)
- Orders(ordno, cid, aid, pid, qty)

- Ex: 查询所有订货顾客及其供应商的姓名

$\Pi_{\text{cname,aname}}((\Pi_{\text{cid,cname}}(\text{Customers}) \times \text{Orders}) \times \text{Agents})$

$\Pi_{\text{cname,aname}} (\sigma_{\text{Customers.cid} = \text{Orders.cid} \wedge \text{Orders.aid} = \text{Agents.aid}} ((\text{Customers} \times \text{Orders}) \times \text{Agents}))$



3.3.2 简单查询 (续)

—连接查询说明

- SELECT语句的执行步骤

$\Pi_{\text{cname,aname}} (\sigma_{\text{Customers.cid} = \text{orders.cid} \wedge \text{Orders.aid} = \text{Agents.aid}} ((\text{Customers} \times \text{Orders}) \times \text{Agents}))$

```
SELECT DISTINCT Customers.cname, Agents.aname
FROM Customers, Orders, Agents
WHERE Customers.cid = Orders.cid
      AND Orders.aid = Agents.aid
```



3.3.2 简单查询 (续)

- 4. FROM 子句

- *Ex:* 列出教授“哲学”课程的老师的教工号及姓名

```
SELECT Prof.pno , Prof.pname  
FROM Prof, PC, Course  
WHERE Prof.pno = PC.pno  
AND PC.Cno = Course.cno  
AND Course.cname = “哲学”
```



3.3.2 简单查询 (续)

- 5. WHERE 子句

- 语法成分

- 比较运算符

<、<=、>、>=、=、<>

- 逻辑运算符

AND, OR, NOT

- BETWEEN 条件

- 判断表达式的值是否在某范围内



3.3.2 简单查询 (续)

- 5. WHERE 子句

Ex 1: 查询工资低于2000的老师的姓名、工资、系别

```
SELECT pname , sal , dname  
FROM   Prof , Dept  
WHERE      sal < 2000  
          AND Prof.dno = Dept.dno
```

Ex 2: 查询工资在1000-2000之间的老师姓名

```
SELECT  pname  
FROM    Prof  
WHERE   sal BETWEEN 1000 AND 2000
```



3.3.2 简单查询 (续)

- 6. 更名运算

- 格式

old_name **AS** *new_name*

为关系和属性重新命名，可出现在SELECT和FROM子句中

注: **AS**可选

- 属性更名

Ex1 :给出所有老师的姓名、所纳税额及税后工资额

```
SELECT pname, sal*0.05 AS taxi,
```

```
       sal * 0.95 AS incoming
```

```
FROM   Prof
```




3.3.2 简单查询 (续)

- 6. 更名运算

- 关系更名

Ex 2: 查询工资比所在系主任工资高的老师姓名及工资

```
SELECT  P1.pname, P1.sal
FROM    Prof AS P1, Prof AS P2,
Dept
WHERE   P1.dno = Dept.dno
        AND   Dept.dean = P2.pno
        AND   P1.sal > P2.sal
```



3.3.2 简单查询（续）

• 7. 匹配查询

- 语法规则

- 列名 [NOT] LIKE '<匹配串>' [ESCAPE '<换码字符>']
- 找出满足给定匹配条件的字符串

- 匹配规则

- “%”
 - 匹配零个或多个字符
- “ ”
 - 匹配任意单个字符
- Escape
 - 定义转义字符，以去掉特殊字符的特定含义，使其被作为普通字符看待

思考：用什么去匹配\？



3.3.2 简单查询（续）

- 7. 匹配查询

- Ex 1: 查询姓名以“张”打头的教师的所有信息

```
SELECT *  
FROM Prof  
WHERE pname LIKE '张%'
```

- Ex 2: 查询姓‘司马’且全名为四个汉字的学生的姓名

```
SELECT sname  
FROM S  
WHERE sname LIKE '司马_ _ _ _'
```



3.3.2 简单查询 (续)

- 7. 匹配查询

- Ex 3: 列出名称中含有4个字符以上，且倒数第3个字符是d，倒数第2个字符是_ 的系的所有信息

```
SELECT  *  
FROM    Prof  
WHERE   pname LIKE '%__d\__' escape '\'
```



3.3.2 简单查询（续）

- 8. 元组显示顺序

- 命令

- *ORDER BY* 列名 [*ASC* | *DESC*]

- Ex 1: 按系名升序列出老师姓名，所在系名，同一系中老师按姓名降序排列

```
SELECT dname, pname  
FROM Prof, Dept  
WHERE Prof.dno = Dept.dno  
ORDER BY dname ASC, pname DESC
```

3.3.2 简单查询 (续)

- 9. 分组与聚集函数

- 聚集函数

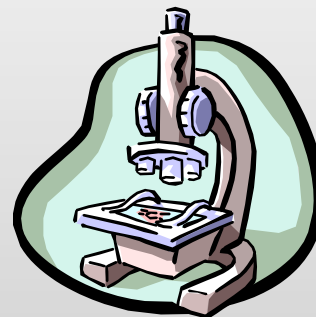
- $\text{COUNT}([\text{DISTINCT} \mid \text{ALL}] A) / \text{CONCT}([\text{DISTINCT} \mid \text{ALL}]*)$

- $\text{SUM}([\text{DISTINCT} \mid \text{ALL}] A)$

- $\text{AVG}([\text{DISTINCT} \mid \text{ALL}] A)$

- $\text{MAX}([\text{DISTINCT} \mid \text{ALL}] A)$

- $\text{MIN}([\text{DISTINCT} \mid \text{ALL}] A)$





3.3.2 简单查询（续）

- 9. 分组与聚集函数

- Ex 1: 查询教师的平均工资，最低工资，最高工资

```
SELECT AVG(sal), MIN(sal), MAX(sal)  
FROM Prof
```

- Ex 2: 查询年纪最大的教师年龄

```
SELECT MAX(age)  
FROM Prof
```



3.3.2 简单查询（续）

- 9. 分组与聚集函数

- 分组命令

GROUP BY 列名 [*HAVING* 条件表达式]

- *GROUP BY* + *Group-list*

- 将表中的元组按指定列上值相等的原则分组，然后在每一分组上使用聚集函数，得到单一值

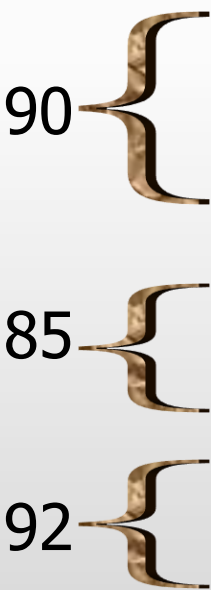
- *HAVING* + *Group-qualification*

- 对分组进行选择，只将聚集函数作用到满足条件的分组上

3.3.2 简单查询（续）

■9. 分组与聚集函数

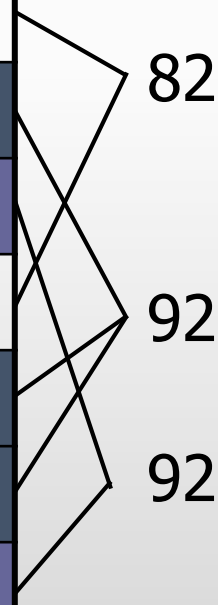
列出每个学生的平均成绩



Sno	Cno	score
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

GROUP BY Sno

列出每门课程的平均成绩



Sno	Cno	score
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

GROUP BY Cno



3.3.2 简单查询 (续)

- 9.分组与聚集函数

SC(Sno , Cno , SCORE)

- Ex 3: 列出每个学生的平均成绩

```
SELECT sno, AVG(score)
```

```
FROM SC
```

```
GROUP BY sno
```

- Ex 4: 列出每门课程的平均成绩

```
SELECT cno, AVG(score)
```

```
FROM SC
```

```
GROUP BY cno
```



3.3.2 简单查询 (续)

- 9.分组与聚集函数

- Ex 5: 查询各系的老师的最高、最低、平均工资

```
SELECT dno, MAX(sal), MIN(sal), AVG(sal)
FROM Prof
GROUP BY dno
```

- Ex 6: 统计男学生人数大于50人的小组的人数

```
SELECT age, COUNT(sno)
FROM S
WHERE sex = 'M'
GROUP BY age
HAVING COUNT(*) > 50
```

示例 I

Sn o	C no	score
100	8	90
100	1	90
100	3	50
200	8	60
200	1	70
200	3	90
300	1	70
300	3	50

– Ex 7: 查询每门课程的及格人数



```
SELECT  
  cno ,COUNT(sno)  
FROM SC  
GROUP BY cno  
HAVING score >= 60
```

```
SELECT cno, count(sno)  
FROM SC  
WHERE score >= 60  
GROUP BY cno
```

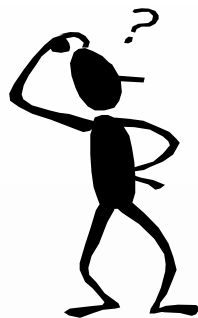
C no	Expr1
1	3
3	1
8	2

示例 II

- Ex 8: 列出所有课程都及格的学生
的平均成绩

```
SELECT sno, AVG(score)
FROM SC
GROUP BY sno
HAVING MIN(score) >= 60
```

```
SELECT sno, AVG(score)
FROM SC
WHERE score >= 60
GROUP BY sno
```



哪个正确?

Sn o	Expr1
200	73

Sn o	Expr1
100	90
200	73
300	70



3.3.2 简单查询（续）

- 10. 空值

- 格式

IS [*NOT*] *NULL*

- 测试指定列的值是否为空值

- = NULL



- Ex 1: 找出年龄值为空的老师姓名

```
SELECT pname
FROM Prof
WHERE age IS NULL
```



3.3.2 简单查询 (续)

- 10. 空值

- 说明

```
SELECT SUM (score )  
FROM SC
```

350

```
SELECT COUNT(*)  
FROM SC
```

6

Sno	Cno	score
s1	c1	80
s1	c2	90
s1	c3	95
s2	c1	85
s2	c2	null
s3	c2	null



3.3.2 简单查询（续）

- 11. TopK查询

- 格式

SELECT TOP (expression) [PERCENT] [WITH TIES]
[ORDER BY (expression)]

- 输出查询结果的前N条记录
 - 输出查询结果前百分之N的记录
 - 例子： 查询“001”号课程前十名同学的学号和分数
 - Select TOP(10) sno,grade from sc where cno = '001'
 - Order by grade desc



3.3.3 嵌套查询

- 1. 概念
- 2. 集合成员资格 (IN)
- 3. 集合之间的比较(比较运算符)
- 4. 相关嵌套查询



3.3.3 嵌套查询(续)

- 1. 概念
 - 查询块
 - 嵌套查询
 - 父查询
 - 子查询



3.3.3 嵌套查询(续)

- 2. 集合成员资格 (IN)

- 格式

表达式 [*NOT*] *IN* (子查询)

- 判断表达式的值是否在子查询的结果中

- Ex 1: 查询预定了103号船的水手的姓名

Sailor(sid,sname)
Boat(bid,bname,color)
Reserve(sid,bid)

方案1: 连接运算

```
SELECT S.sname  
FROM Sailor as S Reserve as R  
WHERE S.sid = R.sid  
AND R.bid = '103'
```



3.3.3 嵌套查询(续)

- 2. 集合成员资格 (IN)

- Ex 1: 查询预定了103号船的水手的姓名

第一步：先找出预定船只的水手的编号

```
SELECT R.sid  
FROM Reserve as R  
WHERE R.bid = '103'           ( '22' , '31' )
```

第二步：根据编号查询姓名

```
SELECT S.sname  
FROM Sailor S  
WHERE S.sid = '22'  
OR S.sid = '31'
```



3.3.3 嵌套查询(续)

- 2. 集合成员资格 (IN)
 - Ex 1: 查询预定了103号船的水手的姓名

方案2: 嵌套子查询

```
SELECT S.sname  
FROM Sailor as S  
WHERE S.sid IN  
    ( SELECT R.sid  
      FROM Reserve as R  
      WHERE R.bid = '103'  
    )
```



3.3.3 嵌套查询(续)

- 2. 集合成员资格 (IN)
 - Ex 1: 查询预定了所有红颜色船只的水手的姓名

```
SELECT S.sname
FROM Sailor as S
WHERE S.sid IN
(SELECT R.sid
FROM Reserve as R
WHERE R.bid IN
( SELECT B.bid
FROM Boat as B
WHERE B.color = 'red' ) )
```



3.3.3 嵌套查询(续)

- 3. 集合之间的比较
 - 格式
 - 表达式 比较运算符 θ *ANY* (子查询)
 - 表达式的值至少与子查询结果中的一个值相比满足比较运算符 θ
 - 表达式 比较运算符 θ *ALL* (子查询)
 - 表达式的值与子查询结果中的所有的值相比都满足比较运算符 θ



3.3.3 嵌套查询(续)

- 3. 集合之间的比较
 - Ex 2: 查询与王红在同一个系学习的同学的姓名
SELECT sname,
FROM S
WHERE S.Dno = (SELECT Dno
FROM S
WHERE sname = '王红')



3.3.3 嵌套查询(续)

- 3. 集合之间的比较

- Ex 3: 查询比某一个名叫Horatio的水手级别高的水手的姓名

```
SELECT S.sname  
FROM Sailor S  
WHERE S.rating > [ANY ( SELECT S2.raing  
                        FROM Sailor S2  
                        WHERE S2.sname = 'Horatio' )
```

- Ex 4: 查询比所有名叫Horatio的水手级别高的水手的姓名

```
SELECT S.sid  
FROM Sailor S  
WHERE S.rating > ALL ( SELECT S2.raing  
                      FROM Sailor S2  
                      WHERE S2.sname = 'Horatio' )
```



3.3.3 嵌套查询(续)

- 3. 集合之间的比较
 - ANY , ALL谓词与集函数以及IN谓词的等价转换关系
 - Ex 4: 查询比所有名叫Horatio的水手级别高的水手的姓名

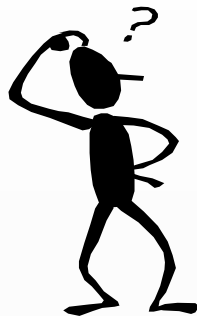
集函数方法:

```
SELECT sname
FROM Sailor
WHERE rating > (SELECT MAX(rating)
                FROM Sailor
                WHERE sname= 'Horatio')
```

3.3.3 嵌套查询(续)

- 3. 集合之间的比较

- Ex 5: 查询平均成绩最高的学生的编号



```
SELECT sno
FROM SC
GROUP BY sno
HAVING AVG(score) >= ALL (SELECT AVG(score)
                           FROM SC
                           GROUP BY sno)
```



3.3.3 嵌套查询(续)

- 4. 相关嵌套查询

- 概念

- 不相关子查询
 - 相关子查询

- Ex 6: 查询选修了001号课程的学生姓名

```
SELECT sname
FROM S
WHERE sno IN ( SELECT sno
                FROM SC
                WHERE cno='001')
```



3.3.3 嵌套查询(续)

- 4. 相关嵌套查询(EXISTS)

- [NOT] EXISTS (子查询)

- 说明

- 判断子查询的结果集中是否有任何元组存在

–Ex 6: 查询选修了001号课程的学生姓名

```
SELECT sname
FROM S
WHERE EXISTS
  (SELECT *
   FROM SC
   WHERE sno = S.sno
    AND cno= '001')
```



3.3.3 嵌套查询(续)

- 4. 相关嵌套查询(EXISTS)
 - Ex 7: 查询未选修 '001' 课程的学生姓名

```
SELECT sname  
FROM S  
WHERE NOT EXISTS  
      (SELECT *  
       FROM SC  
       WHERE sno =  
S.sno  
       AND cno='001')
```

```
SELECT sname  
FROM S  
WHERE sno <> ALL (SELECT sno  
                  FROM SC  
                  WHERE  
cno='001')
```



3.3.3 嵌套查询(续)

- 4. 相关嵌套查询

Ex 8: 查询选修了至少一门由 's001'选修的课程的学生学号

```
SELECT cno FROM SC SCX
WHERE    cno IN
        (SELECT cno FROM SC SCY
         WHERE SCY.sno = 's001')
```

Ex 9: 查询至少被两个人选修的课程号

```
SELECT Distinct cno FROM sc
WHERE sno <> any (SELECT sno FROM sc newsc
                 WHERE cno = sc.cno)
```



3.3.3 嵌套查询(续)

- 4. 相关嵌套查询(EXISTS)
- IN和EXISTS的比较

*IN*后的子查询与外层查询无关，每个子查询执行一次，而 *EXISTS*后的子查询与外层查询有关，需要执行多次。

课后复习：教材例42,43,44



3.3.4 集合查询

- 格式

subquery ::= SELECT 子句
 FROM 关系名表
 [WHERE 条件表达式]
 [GROUP By 分组属性名表达式
 [HAVING 条件]]

| subquery {UNION [ALL] | INTERSECT [ALL] |
 MINUS[ALL]
 } subquery



3.3.4 集合查询

- Ex 8:求选修了001或002号课程的学生号

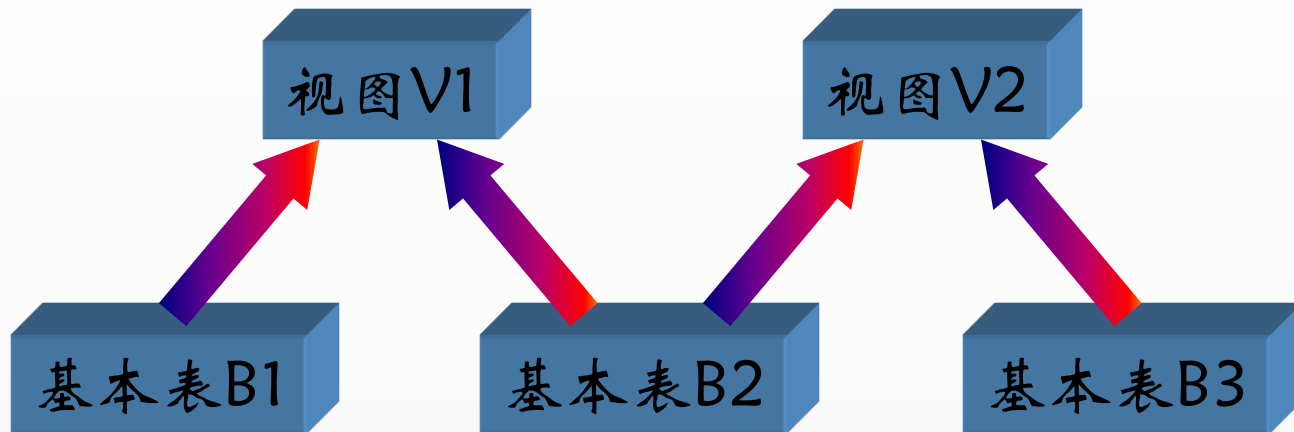
```
(SELECT sno  
FROM SC  
WHERE cno = '001')
```

UNION

```
(SELECT sno  
FROM SC  
WHERE cno = '002')
```

3.3.5 视图

- 1. 视图的基本概念



- 创建视图

CREATE *view_name* **AS** <查询表达式>

[WITH CHECK OPTION]

- 撤销视图

DROP VIEW *view_name*



3.3.5 视图(续)

- 2. 视图定义示例 1

```
CREATE VIEW dbo.st_bjqk
AS
SELECT dbo.bjqk.bjbh, dbo.bjqk.zybh, dbo.bjqk.bjmc,
       dbo.bjqk.bjrs, dbo.zyqk.Zymc,
       dbo.zyqk.Sznf, dbo.zyqk.Tznf, dbo.zyqk.Yxbh,
       dbo.bjqk.sfdezybj
FROM dbo.bjqk INNER JOIN
     dbo.zyqk ON dbo.bjqk.zybh = dbo.zyqk.Zybh
```



3.3.5 视图(续)

- 2. 视图定义示例 2

```
CREATE VIEW dbo.st_ktt
AS
SELECT dbo.ktt.kttbh, dbo.ktt.kttzh, dbo.ktt.kcbh, dbo.ktt.ywcdzbh, dbo.ktt.jszypzzl,
       dbo.kcjb.kcmc, dbo.kttbjdy.bjbh, dbo.bjqk.bjmc, dbo.bjqk.bjrs, dbo.bjqk.zybh,
       dbo.zyqk.Zymc, dbo.zyqk.Yxbh, dbo.xyqk.yxmc, dbo.pkdzzlb.pkdzmc,
       dbo.xyqk.sznf AS yxsznf, dbo.xyqk.tzsj AS yxtzsj, dbo.zyqk.Sznf AS zysznf,
       dbo.zyqk.Tznf AS zytzsj
FROM dbo.pkdzzlb INNER JOIN
      dbo.kcjb INNER JOIN
      dbo.ktt ON dbo.kcjb.kcbh = dbo.ktt.kcbh INNER JOIN
      dbo.kttbjdy ON dbo.ktt.kttbh = dbo.kttbjdy.kttbh ON
      dbo.pkdzzlb.pkdzbh = dbo.ktt.ywcdzbh INNER JOIN
      dbo.bjqk ON dbo.kttbjdy.bjbh = dbo.bjqk.bjbh INNER JOIN
      ...
```



3.3.5 视图(续)

- 2. 视图定义示例 3

```
CREATE VIEW agentorders(ordno, month, cid, aid, pid, qty,charge,  
    aname, acity, percent)
```

```
AS
```

```
SELECT o.ordno, o.month, o.cid, o.aid, o.qty, o.dollars, a.aname,  
    a.city, a.percent
```

```
FROM orders o, agents a WHERE o.aid = a.aid
```

```
SELECT sum(charge) FROM agentorders  
WHERE acity = 'ABC'
```

```
SELECT sum(o.dollars) FROM orders o, agents a  
WHERE o.aid = a.city AND a.city = 'ABC'
```



3.3.5 视图(续)

- 2. 视图定义示例 4

```
CREATE VIEW Deptsal( dno,low,high,avgsal,total )  
AS ( SELECT  dno, MIN(sal), MAX(sal),AVG(sal),SUM(sal)  
      FROM    Prof  
      GROUP BY dno )
```

- 给出计算机系老师的最低、最高、平均工资以及工资总额

```
SELECT  low , high , avgsal , total  
FROM    Deptsal , Dept  
WHERE   Deptsal.dno = Dept.dno  
        AND  Dept.dname =“计算机系”
```



3.3.5 视图(续)

- 2. 视图 **vs** 临时表

- 临时表（派生关系）

（子查询） **AS** 关系名（列名，列名，...）

–Ex 9: 查询平均成绩及格的学生姓名和成绩
先求出每个学生的平均成绩，再从中找出及格的学生

```
SELECT  sname, AVG (score)
FROM    S, SC
WHERE   SC.sno = S.sno
GROUP BY SC.sno
```




3.3.5 视图(续)

- 2. 视图 vs 临时表

```
SELECT sno, avg_score
FROM student, (SELECT SC.sno, AVG(score)
                FROM S, SC
                WHERE SC.sno = S.sno
                GROUP BY SC.sno)
AS RESULT(sno, avg_score)
WHERE student.sno = result.sno and avg_score >= 60
```



3.3.5 视图(续)

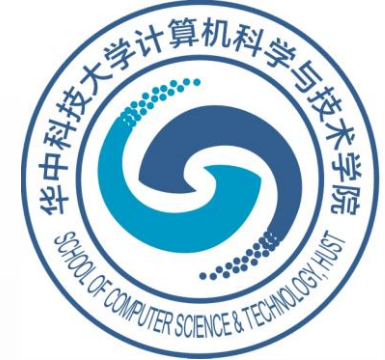
- 3. 相关说明
 - 对视图的删除操作
 - 对视图的更新操作
 - INSERT, UPDATE, DELETE
 - 操作约束
 - 视图的优点
 - 个性化服务
 - 简化应用处理
 - 安全性
 - 逻辑独立性



自我学习

- 关系的连接
 - 基本分类
 - 连接成分
 - 包括两个输入关系、连接条件、连接类型
 - 连接条件
 - 决定两个关系中哪些元组相互匹配，以及连接结果中出现哪些属性
 - 连接类型
 - 决定如何处理与连接条件不匹配的元组

自我学习



连接类型	连接条件
INNER JOIN LEFT OUTER JOIN RIGHT OUTER JOIN FULL OUTER JOIN	ON $[A_1, A_2, \dots, A_n]$ 运算符 $[B_1, B_2, \dots, B_n]$...



练习

- Suppliers(sno, sname, rating, city)
 - Part(pno, pname, color, weight, city)
 - Job(jno, jname, city)
 - SPJ(sno, pno, jno, qty)
-
- 1. 查询使用供应商S1所供应零件的工程号码
 - 2. 查询工程项目J2使用的各种零件的名称及其数量
 - 3. 查询使用上海产的零件的工程名称
 - 4. 查询上海厂商供应的所有的零件号码
 - 5. 查询使用了两种以上不同零件的工程名称
 - 6. 查询使用零件类别大于10的项目的编号
 - 7. 查询使用零件类别最多的项目的编号



3.4 SQL数据修改功能

- 1. 插入
- 2. 删除
- 3. 更新
- 4. 视图更新



3.4 SQL数据修改功能(续)

- 1. 插入

- 格式

- 插入单个元组

- ```
INSERT INTO 表名 [(列名[, 列名]...)
VALUES (值 [, 值]...)
```

- 插入子查询结果

- ```
INSERT INTO 表名 [(列名[, 列名]...)  
(子查询)
```



3.4 SQL数据修改功能(续)

- 1. 插入
 - 示例
 - *INSERT INTO Prof*
VALUES ('P123', '王明', 35, 'D08', '498')
 - *INSERT INTO Prof (PNO, PNAME, DNO)*
VALUES ('P123', '王明', 'D08')



3.4 SQL数据修改功能(续)

- 1. 插入操作

- Ex 10:将平均成绩大于90的学生加入到表Excellent中

Excellent(sno,avg_score)

SC(sno,cno,score)

```
INSERT INTO Excellent ( sno, score)
```

```
SELECT sno , AVG(score)
```

```
FROM SC
```

```
GROUP BY sno HAVING AVG(score) > 90
```



3.4 SQL数据修改功能(续)

```
INSERT INTO Excellent  
SELECT *  
FROM Excellent
```

不支持修改在子查询中出现的表

一般在完成查询后，再执行修改操作



3.4 SQL数据修改功能(续)

- 2. 删除操作

- 格式

- `DELETE FROM <表名>`
`[WHERE <条件表达式>]`

- Ex11: 删除所有选课记录

- `DELETE FROM SC`

- Ex12: 删除王明老师的任课记录

- `DELETE FROM PC`
`WHERE pno IN (SELECT Pno`
`FROM Prof`
`WHERE pname = "王明")`



3.4 SQL数据修改功能(续)

- 3.更新操作

- 格式

UPDATE 表名

SET 列名 = 表达式 | 子查询

列名 = [, 表达式 | 子查询]...

[*WHERE* 条件表达式]

Ex 13: 老师工资上调5%

UPDATE Prof

*SET sal = sal * 1.05*



3.4 SQL数据修改功能(续)

- 3.更新操作
 - Ex 14:将D01系系主任的工资改为该系的平均工资

```
UPDATE Prof
SET sal = (SELECT AVG(sal)
           FROM Prof
           WHERE dno = 'D01')
WHERE pno = (SELECT dean
             FROM Dept
             WHERE dno = 'D01')
```



3.4 SQL数据修改功能(续)

- 4. 视图更新
 - 操作分类
 - INSERT, DELETE, UPDATE
 - 特点
 - 对视图的更新，最终要转换为对基本表的更新
- View Ex 1:

```
CREATE VIEW IS_Student
AS
SELECT Sno, Sname, Sage
FROM Student
WHERE Sdept='IS'
```



3.4 SQL数据修改功能(续)

- 4. 视图更新

- Ex 1: 向信息系学生视图IS_S中插入一个新的学生记录, 其中学号为95029, 姓名为赵新, 年龄为20岁

```
INSERT  
INTO IS_Student  
VALUES('95029', '赵新', 20)
```



```
INSERT  
INTO Student(sno,sname,sage,sdept)  
VALUES('95029', '赵新', 20, 'IS')
```



3.4 SQL数据修改功能(续)

- 4. 视图更新
- Ex 2:删除计算机系学生视图CS_S中学号为95029的记录

```
DELETE  
FROM IS_Student  
WHERE sno='95029'
```



```
DELETE  
FROM Student  
WHERE Sno='95029' AND Sdept='IS'
```




3.4 SQL数据修改功能(续)

- 4. 视图更新
 - 视图更新的限制
 - SELECT子句中的目标列不能包含聚集函数
 - SELECT子句中不能使用UNIQUE或DISTINCT关键字
 - 不能包括GROUP BY子句
 - 不能包括经算术表达式计算出来的列
 - 对于行列子集视图可以更新（视图是从单个基本表使用选择、投影操作导出的，并且包含了基本表的主码）

视图更新的限制

- 示例1:

```
CREATE VIEW P_SAL  
AS (SELECT pno ,pname , sal  
FROM Prof)
```

```
INSERT INTO P_SAL  
VALUES ( P08 , “张立” , 750 )
```

转换为

```
INSERT INTO PROF  
VALUES ( P08 , “张立” , null , null , 750 )
```



视图更新的限制

- 示例2：

```
create view S_C_G  
as (select SNAME, CNAME, GRADE  
    from S, C, SC  
    where S.Sno = SC.Sno  
    and C.Cno = SC.Cno)
```

```
insert into S_C_G  
values ( "张立", "物理", 97 )
```

往SC中插入对应姓名为张立的学号 and 对应课程名为物理的课程号，以及成绩为97的元组

如果有多个张立怎么办？



3.5 SQL数据控制功能

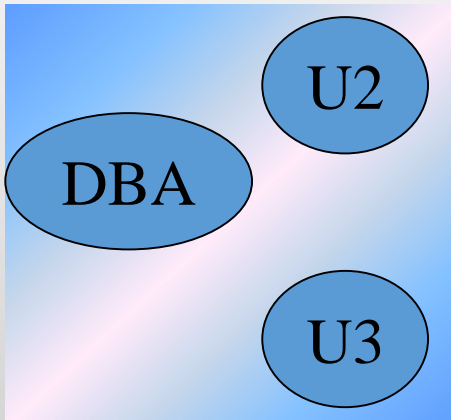
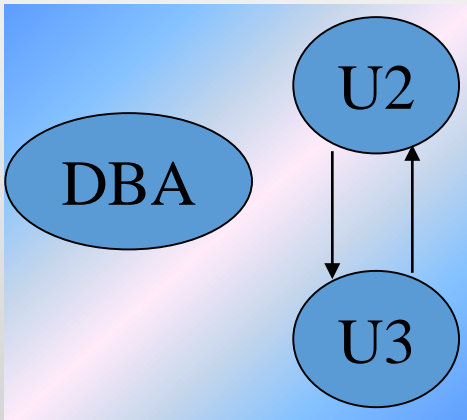
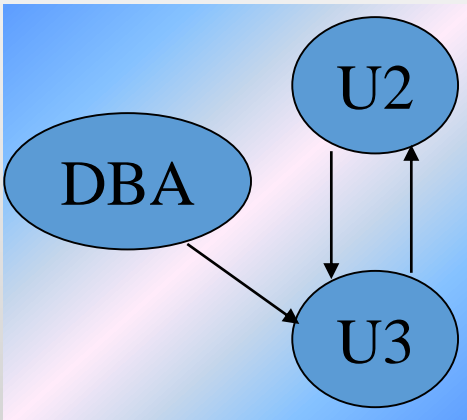
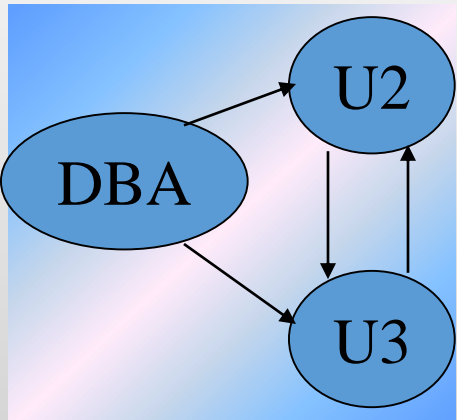
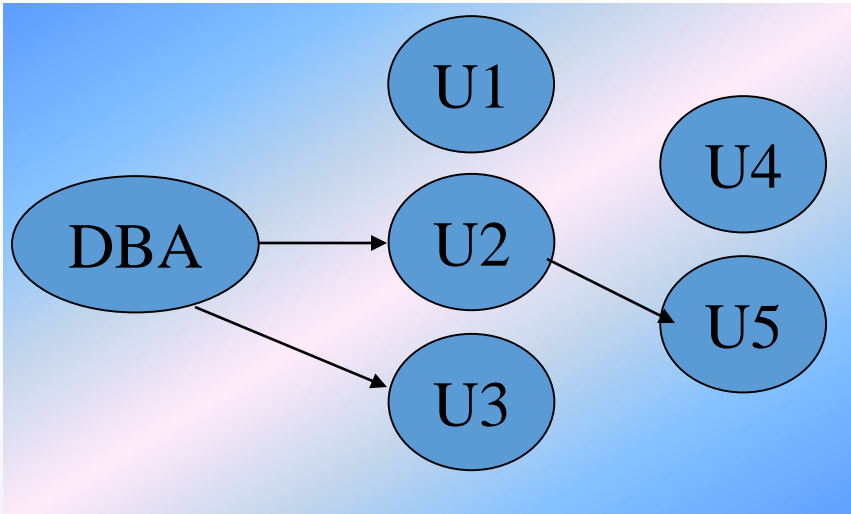
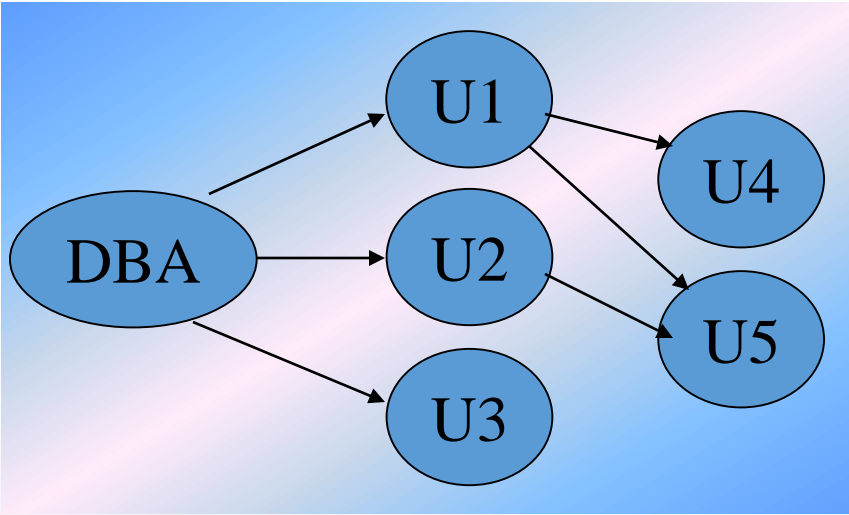
- 3.5.1 数据控制简介
- 3.5.2 安全性控制简介
- 3.5.3 授权
- 3.5.4 收回权限



3.5.2 安全性控制简介

- 安全性控制定义
- 安全性控制的基本措施
- 安全控制的级别
 - 物理级
 - 人际级
 - 操作系统级
 - 网络级
 - 数据库系统级
- 权限的转授和回收
 - 允许用户把已获得的权限转授给其他用户，也可以把已授给其他用户的权限再回收上来

权限的转授和回收





3.5.3 授权

- 1. 命令格式

```
GRANT <权限>[,<权限>]...  
    [ON <对象类型> <对象名>]  
    TO <用户>[,<用户>]...  
    [WITH GRANT OPTION];
```

Ex 1: 把查询Student表权限授给用户A

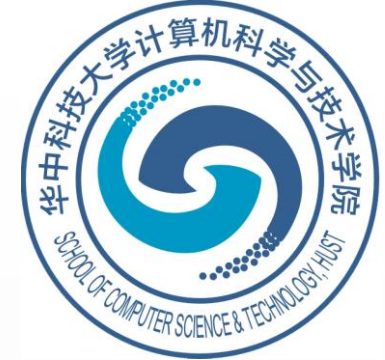
```
GRANT SELECT ON TABLE Student TO A;
```

Ex2: 把对Student表和Course表的全部权限授予用户U2和U3

```
GRANT ALL PRIVILIGES ON TABLE Student, Course TO U2, U3;
```

Ex3: 把对表SC的查询权限授予所有用户

```
GRANT SELECT ON TABLE SC TO PUBLIC;
```



授权操作权限表

对象	对象类型	操作权限
属性列	TABLE	SELECT, INSERT, UPDATE, DELETE ALL PRIVIEGES
视图	TABLE	SELECT, INSERT, UPDATE, DELETE ALL PRIVIEGES
基本表	TABLE	SELECT, INSERT, UPDATE, ALTER, INDEX,DELETE ALL PRIVIEGES
数据库	DATA BASE	CREATETAB



3.5.4 收回权限

- 1. 命令格式

```
REVOKE <权限>[,<权限>]...  
[ON <对象类型> <对象名>]  
FROM <用户>[,<用户>| public]...
```

- Ex 4:把用户U4修改学生学号的权限收回

```
REVOKE UPDATE(sno) ON TABLE Student FROM U4
```

参见教材P132例7, 8, 9以及相关说明



练习

1.设有关系R如下， 写一条SQL语句将其数据变为右表所示。

序号	总金额	金额	部门
1		10	1
2		10	1
3		10	1
4		20	2
5		20	2

序号	总金额	金额	部门
1	30	10	1
2	30	10	1
3	30	10	1
4	40	20	2
5	40	20	2

2.查询 ‘1’ 号课程成绩比 ‘2’ 号课程成绩高的学生学号

练习



1. Update test1 Set zje = (Select Sum(je)
From test1 new Where bm = test1.bm)
2. Select sc1.sno From sc sc1,sc sc2
Where sc1.cno = '1'
And sc2.cno = '2' And sc1.sno = sc2.sno
And sc1.score > sc2.score

练习



- 1.列出有二门以上(含两门)不及格课程的学生学号、姓名及其平均成绩
- 2.列出仅学过课程“DS”和课程“OS”的所有学生姓名

练习

```
select s.sno,s.sname,avg(sc.score)
from student s,sc
where s.sno in
(select sno from sc
where score < 60
group by sno
having count(distinct cno) > 1)
group by s.sno,s.sname
```





练习

select sname from student
where sno in

(select distinct sno from sc where cno in
(select cno from course
where cname = 'DS'))

And sno in (select distinct sno from sc where cno in
(select cno from course
where cname = 'OS'))

And sno in

(select sno from sc
group by sno
having count(distinct cno) = 2)

练习

```
select sno,sname from student
where sno not in
(select distinct sno from sc where cno not in
(select cno from course
where cname in ('DS','OS') ))
And sno in
(select sno from sc sc
group by sno
having count(distinct cno) = 2 )
```





练习

select sno, sname from student
where not exists

(select distinct sno from sc
where sno = student.sno and cno not in
(select cno from course
where cname in ('DS','OS')))

And sno in

(select sno from sc sc
group by sno
having count(distinct cno) = 2)



练习-排名

查询选修1号课程的学号、成绩和排名，排名有两种：连续和不连续

- ① 同分数同名次，总排名连续。如95、90、90、85、85、80的排名结果为1,2,2,3,3,4
- ② 同分数同名次，总排名不连续。如95、90、90、85、85、80的排名结果为1,2,2,4,4,6

-- 连续排名

```
select sno,grade,(select count(distinct grade) from sc
where cno = 1 and grade >=s.grade) as `rank`
from sc as s
where cno = 1
order by grade desc;
```

sno	grade	rank
201215121	92	1
201215122	92	1
201215888	90	2
201215123	80	3
201215125	80	3
201215889	78	4



练习-排名

-- 不连续排名

```
select sno, grade,  
(select count(*) from sc where cno = 1 and grade>s1.grade)+1  
as `rank`  
from sc as s1  
where cno = 1  
order by `rank`;
```

sno	grade	rank
201215121	92	1
201215122	92	1
201215888	90	3
201215123	80	4
201215125	80	4
201215889	78	6

练习

1.设有关系R如下，写一条SQL语句将其数据变为右表所示。

id1	id2	bz
1	2	a
2	1	a
1	3	b
3	1	b
2	3	c
3	2	c

id1	id2	bz
1	2	a
1	3	b
2	3	c

id1	id2	bz
2	1	a
3	1	b
3	2	c



Exists

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 cno = course.cno
     and sno = student.sno))
```

```
select sname
from student
where sno in
  (select sno
   from sc
   group by sno
   having count(*)=
    (select count(*)
     from course))
```



3.6 嵌入式SQL

- 3.6.1 概述

- 交互式SQL的局限性

- 3.6.2 嵌入式SQL的执行过程

- 3.6.3 主要问题及解决方案

1. 主要问题
2. 嵌入式SQL的一般形式
3. 嵌入式SQL语句与主语言之间的通信
4. 游标

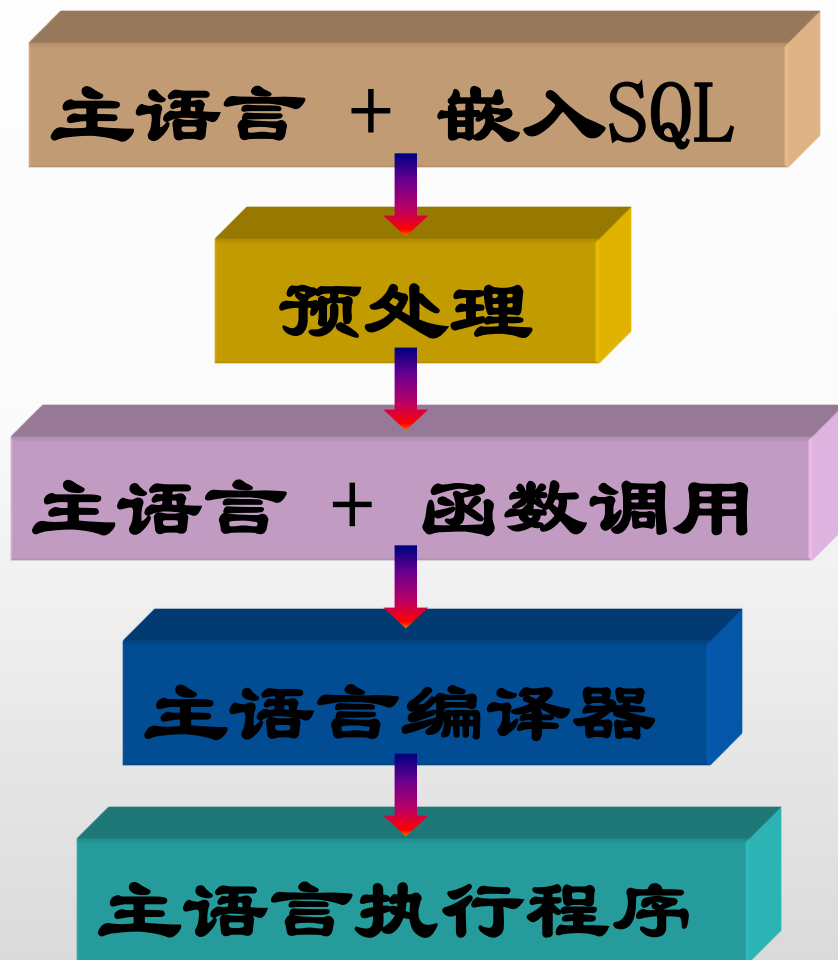
- 3.6.4 数据库访问接口标准



3.6.1 概述

- 交互式SQL的局限性
- 嵌入式SQL与宿主语言的基本分工
 - 主语言
定义工作变量，算术运算，逻辑运算，程序流程控制。
 - SQL
定义表，定义视图，定义索引，表和视图操作（I，D，U，M，……），控制。

3.6.2 嵌入式SQL的执行过程





3.6.3 主要问题及解决方案(续)

- 1. 主要问题
 - 主语言与SQL语句的识别
 - 主语言程序变量与DB工作变量间通讯
 - 向主语言传递SQL执行状态信息，以便主语言控制程序流程 (SQLCA)
 - 主语言向SQL提供参数（插入值，改变值，条件）
 - 将SQL查询结果交给主语言处理，（主变量+游标）
 - 一与多矛盾（游标）



3.6.3 主要问题及解决方案(续)

- 2. 嵌入式SQL的一般形式

- 宿主语言 C, Pascal, PL/1
- 嵌入的SQL语句以**EXEC SQL**开始, 以分号(;) 或 **END_EXEC**结束

```
EXEC SQL    DELETE FROM Prof  
              WHERE dno = '10';
```



3.6.3 主要问题及解决方案(续)

- 3. 嵌入式SQL语句与主语言之间的通信
 - SQL通信区
 - SQLCA数据结构
 - SQLCA功能
 - 存储将向主语言传递SQL执行状态信息，以便主语言控制程序流程。



3.6.3 主要问题及解决方案(续)

- SQLCA数据结构

```
typedef struct sqlca{  
    unsigned char  
    sqlcaid [EYECATCH_LEN];           // Eyecatcher = 'SQLCA '  
    long sqlcabc;                      // SQLCA size in bytes = 136  
    long sqlcode;                     // SQL return code  
    short sqlerrml;                   // Length for SQLERRMC  
    unsigned char  
    sqlerrmc[SQLERRMC_SIZE];          // Error message tokens  
    unsigned char sqlerrp[8];         // Diagnostic information  
    long sqlerrd[6];                 // Diagnostic information  
    unsigned char sqlwarn[8];        // Warning flags  
    unsigned char sqlext[3];         // Reserved  
    unsigned char sqlstate[5];       // new member  
} SQLCA;
```



3.6.3 主要问题及解决方案(续)

- 3. 嵌入式SQL语句与主语言(C语言)之间的通信
 - 宿主变量
 - C变量, 既可以用在C语句中, 也可用在SQL语句中, 用来在两者之间传递数据。
 - 声明为通常的C变量, 并将其放在下列标识语句之间

EXEC SQL BEGIN DECLARE SECTION
EXEC SQL END DECLARE SECTION

```
EXEC SQL BEGIN DECLARE SECTION  
        int  prof_no;  
        char prof_name[30];  
        int  salary;  
  
EXEC SQL END DECLARE SECTION
```



3.6.3 主要问题及解决方案(续)

- 3. 嵌入式SQL语句与主语言(C语言)之间的通信

- 宿主变量

- 宿主变量出现于SQL语句中时，前面加 (:) 以区别列名

```
EXEC SQL SELECT pname , sal  
          INTO      :prof_name , :salary  
          FROM      Prof  
          WHERE     pno = : prof_no ;
```

- 宿主变量可出现的地方：SQL的数据操纵语句中可出现 变量的任何地方，SELECT，FETCH等语句的INTO子句中



3.6.3 主要问题及解决方案(续)

- 3. SQL与主语言之间操作方式的协调
 - 执行方式的差别
 - SQL: 一次一集合
 - C语言: 一次一记录
 - 游标
 - 在查询结果的记录集中移动的指针
 - 若一个SQL语句返回单个元组, 则不用游标
 - 若一个SQL语句返回多个元组, 则使用游标



3.6.3 主要问题及解决方案(续)

- 4. 游标

- 定义

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR  
FOR select_statement  
[ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
```

- 类型

- 服务器端, 客户端

```
EXEC SQL DECLARE authors_cursor CURSOR FOR  
SELECT *  
FROM authors;
```



3.6.3 主要问题及解决方案(续)

- 4. 游标
 - 打开（游标）
 - 激活游标，与游标相关的select 执行
 - 游标指向该语句查询结果的第一个元组之前

```
EXEC SQL DECLARE authors_cursor CURSOR FOR  
    SELECT *  
    FROM authors;  
EXEC SQL OPEN authors_cursor;
```




3.6.3 主要问题及解决方案(续)

- 4. 游标
 - 推进（游标）
 - 游标向前推进一个元组，并将游标所指元组的属性值取出，送入into后原来定义的主变量中。
 - Fetch 常用于循环

*EXEC SQL **FETCH INTO** :ename, :eddept ;*

- 关闭（游标）

EXEC SQL CLOSE authors_cursor

程序实例参见教材P243



3.6.3 主要问题及解决方案(续)

- 4. 游标
 - 不需要游标的数据操作
 - (1) 结果是一个元组的select语句

EXEC SQL

SELECT pname , sal

INTO :prof_name : name_id, :salary : sal_id

FROM Prof

WHERE pno = prof_no ;



3.6.3 主要问题及解决方案(续)

- 4. 游标

- 不需要游标的数据操作

- (2) INSERT 语句

EXEC SQL

INSERT INTO Prof VALUES (:prof_no, :prof_name, :salary, :dept_no, :salary);

- (3) 非CURRENT形式UPDATE语句
- (4) 非CURRENT形式DELETE语句



3.6.3 主要问题及解决方案(续)

- 4. 游标
 - 需要游标的数据操作
 - 查询结果为多条记录的SELECT语句
 - 当SELECT语句的结果中包含多个元组时，使用游标可以逐个存取这些元组
 - 活动集
 - SELECT语句返回的元组的集合
 - 当前行
 - 活动集中当前处理的那一行。游标即是指向当前行的指针



3.6.4 数据库访问接口标准

- 1. 专用数据库接口
- 2. 标准接口：ODBC和JDBC



数据库访问标准化接口：ODBC

