数据库基础与应用

- 宋安平
- 上海大学计算机学院1015室
- Apsong@shu.edu.cn

- 第1周—第3周
- 一3-4，二7-8上机，四7-8单

“数据库基础与应用”课程的特点：

- 理论性：关系运算理论、模式设计理论等；
- 实用性：数据库语言、数据库设计；
- 可操作性：较强，有大量问题和应用值得分析和设计；
- 创新性:有许多深层的问题具有发展的余地，有待去挖掘、发现和总结。

课程教学目的和要求：

- 掌握数据库系统的基本概念和原理；
- 理解SQL、关系代数和关系演算等数据库语言；
- 学会关系数据库规范设计的方法和步骤；
- 了解数据库系统的实现技术；
- 具备使用关系数据库软件开发数据库应用系统的能力 。

《数据库基础和应用》课堂教学（30学时)内容及安排：

- 第一章　数据库发展史　（1学时）
- 第二章　数据库系统结构　（3学时）
- 第三章　关系运算　（4学时）
- 第四章　结构化查询语言SQL（8学时）
- 第七章　数据库设计　（2学时）
- 第八章　数据库管理　（8学时）
- 习题分析、研讨　（2学时）
- 总复习　（2学时）

实验内容和安排（20学时）：

- 见WORD附件
- 写实验报告

第4章 结构化查询语言SQL

- SQL概述
- SQL的数据定义
- SQL的数据查询
- SQL的数据更新
- 视图
- 嵌入式SQL

第一节 SQL概述

- SQL发展历程
- SQL数据库的体系结构
- SQL的组成

一、SQL发展历程

- **SQL(Structured Query Language)语言是1974年提出的在IBM公司的System R上实现。**
- **是介于关系代数和关系演算之间的语言。**
- **1986年ANSI批准SQL作为关系数据库语言的美国标准，同年，ISO也同样决定。**
- **以后相继出现SQL89，SQL2（1992），SQL3（1999）。**

二、SQL数据库的体系结构

- **SQL**支持数据库的三级模式结构，如图**4—1**所示。从图中可以看出，模式与基本表相对应，外模式与视图相对应，内模式对应于存储文件。基本表和视图都是关系。
- 1．基本表（**Base Table**）
  - 基本表是模式的基本内容。每个基本表都是一个实际存在的关系。

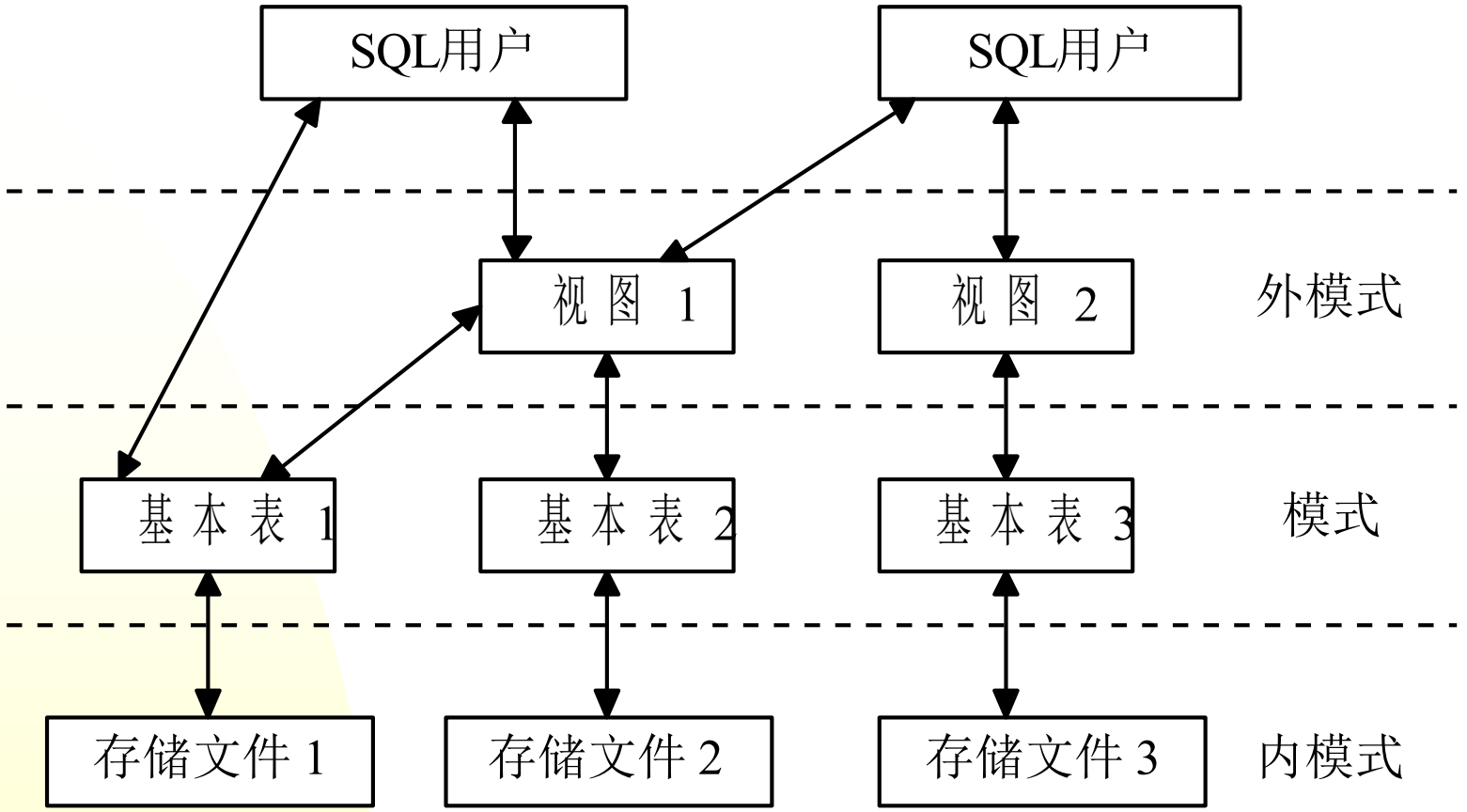

图4—1 SQL支持的数据库模式

二、SQL数据库的体系结构

- **2．视图（View)**
  - 视图是外模式的基本单位，用户通过视图使用数据库中基于基本表的数据（基本表也可作为外模式使用）。
  - 视图是虚表，实际并不存在，只有定义存放在数据字典中。

二、SQL数据库的体系结构

- **3．存储文件**
  - 存储文件是内模式的基本单位。每一个存储文件存储一个或多个基本表的内容。一个基本表可有若干索引，索引也存储在存储文件中。存储文件的存储结构对用户是透明的。
- 下面将介绍**SQL**的基本语句。各厂商的**RDBMS**实际使用的**SQL**语言，与标准**SQL**语言都有所差异及扩充。因此，具体使用时，应参阅实际系统的有关手册

三、SQL的组成

- 数据定义DDL：CREATE、DROP、ALTER
- 数据操纵DML：
  - 数据查询DQL：SELECT
  - 数据操纵DML：INSERT、DELETE、UPDATE
- 数据控制DCL：GRANT、REVOKE
- 嵌入式SQL

第二节　SQL的数据定义

- SQL模式的创建和撤消
- SQL提供的基本数据类型
- 基本表的创建、修改和撤消
- 索引的创建和撤消

一、SQL模式的创建和撤消

- SQL模式的创建
  - CREATE SCHEMA 〈模式名〉 AUTHORIZATION 〈用户名〉
- SQL模式的撤消
  - DROP SCHEMA 〈模式名〉 [CASCADE|RESTRICT]

二、SQL提供的基本数据类型

- 各具体DBMS所提供的数据类型是不同的。但下面的数据类型几乎都是支持的：
  - INT或INTEGER　　全字长二进制整数
  - SMALLINT　　半字长二进制整数
  - DEC(p［,q］)或压缩十进制数，共p位，其中小数点后有q位，
  - FLOAT　　双字长的浮点数
  - CHAR(n)或CHARTER(n)　长度为n的定长字符串
  - VARCHAR(n)　　最大长度为n的变长字符串
  - DATE　　日期型，格式为YYYY—MM—DD
  - TIME　　时间型，格式为HH.MM.SS

三、基本表的创建、修改和撤消

- 基本表的创建—CREATE TABLE
  - **CREATE TABLE <表名>**
    (<列名1> <类型> [<该列的完整性约束>]
    [，<列名2> <类型> [<该列的完整性约束>]]...
    [<,表级完整性约束>])
  - **<该列的完整性约束>**：该列上数据必须符合的条件。最常见的有：
    - **NOT NULL** 该列值不能为空
    - **NULL** 该列值可以为空
    - **UNIQUE** 该列值不能有相同者
    - **DEFAULT** 该列上某值未定义时的默认值
  - **<表级完整性约束>**：对整个表的一些约束条件，常见的有定义主码(外码)，各列上数据必须符合的关联条件等。

三、基本表的创建、修改和撤消

- 例如：有一个学生数据库中，有三个关系
  - **S（Sno，SName，Age，Sex ，Sdept）**
  - **C（Cno，CName，Tname）**
  - **SC（Sno，Cno，Grade）**

```
Create table s
(sno char(4) not null,
sname char(8),
age int,
sex  char(2),
primary key sno);

Create table c
(cno char(4) not null,
cname char(10),
tname  char(8),
primary key cno);

Create table sc
(sno char(4) not null,
cno char(4) not null,
grade int,
primary key (sno,cno),
foreign key sno reference s(sno),
foreign key cno reference c(cno),
check (grade between 0 and 100));
```

三、基本表的创建、修改和撤消

- 基本表结构的修改—ALTER TABLE
  - 基本表的结构是可以随环境的变化而修改的，即根据需要增加、修改或删除其中一列(或完整性约束条件等)。
  - ALTER TABLE<表名>

    ［ADD <列名> <数据类型>［完整性约束］］

    ［DROP <列名>］

    ［MODIFY <列名> <数据类型>［完整性约束］］

三、基本表的创建、修改和撤消

- 基本表的撤消—**DROP TABLE**
  - **DROP TABLE <表名> [CASCADE|RESTRICT]**
  - 此语句一执行，指定的表即从数据库中删除（表被删除，表在数据字典中的定义也被删除），此表上建立的索引和视图也被自动删除(有些系统对建立在此表上的视图的定义并不删除，但也无法使用了)。

四、索引的创建和撤消

- 索引的建立—CREATE INDEX
  - 在一个基本表上，可建立若干索引。有了索引，可以加快查询速度。索引的建立和删除工作由DBA或表的属主(建表人)负责。用户在查询时并不能选择索引，选择索引的工作由DBMS自动进行。
  - CREATE［UNIQUE］INDEX <索引名>
    ON <表名> （ <列名> [ASC|DESC]）...
  - 本语句为规定<表名>建立一索引，索引名为<索引名>。

四、索引的创建和撤消

- 删除索引—**DROP INDEX**
  - 索引太多，索引的维护开销也将增大。因此，不必要的索引应及时删除。
  - **DROP INDEX <索引名>**
  - 本语句将删除规定的索引。该索引在数据字典中的描述也将被删除。

Back

第三节 SQL的数据查询

- SELECT语句的基本格式
- 举例说明

一、SELECT语句的基本格式

- **SELECT［DISTINCT］*|<目标列表达式［别名］清单>**
  FROM <关系名［别名］或视图名清单>
  ［WHERE <查询条件表达式>］
  ［GROUP BY 列名清单
  ［HAVING <组条件表达式>］］
  ［ORDER BY 列名[ASC|DESC]，...]

- 整个语句的执行过程如下：
- 1、读取FROM子句中基本表、视图的数据，执行笛卡尔积操作。
- 2、选取满足WHERE子句中给出的条件表达式的元组。
- 3、按GROUP子句中指定列的值分组，同时提取满足HAVING子句中组条件表达式的那些组。
- 4、按SELECT子句中给出的列名或列表达式求值输出。
- 5、ORDER子句对输出的目标表进行排序,按附加说明ASC升序排列，或按DESC降序排列。

- **SELECT语句中:**
- **WHERE子句称为“行条件子句”，**
- **GROUP子句称为“分组子句”，**
- **HAVING子句称为“组条件子句”，**
- **ORDER子句称为“排序子句”。**

- 在WHERE子句的行条件表达式中可使用下列运算符：

- 算术比较运算符：＜，＜＝，＞，＞＝，＝，＜＞或！＝；
- 逻辑运算符：AND，OR，NOT；
- 集合成员资格运算符：IN，NOT IN；
- 谓词：EXISTS，ALL，SOME，UNIQUE；
- 聚合函数：AVG，MIN，MAX，SUM，COUNT；
- 集合运算符：UNION，INTERSECT，EXCEPT。

二、举例说明

- 单表查询--选择列
- **选择表中的若干列**

1. 查询指定列

例：查询学生的学号和姓名

2. 查询全部列：使用 * 表示所有属性

例：查询全体学生的详细情况

3. 查询经过计算的值: 表达式可用别名代替

例: 查询学生的姓名和出生年份

注意: 字段名称 AS 别名

字段名称 别名

别名=字段名称

4. 消除取值重复的行

例: 查询选课的学生学号

- 单表查询—选择行
- 1. 比较大小：比较运算符 > >= < <= = <>
  - 例：查询1995年以后出生的学生的基本情况
- 2. 确定范围：列名 [NOT] BETWEEN <下限> AND <上限>
  - 例: 查询年龄在20到25之间的学生姓名
- 3. 确定集合：列名 [NOT] IN(<值序列>
  - 例: 查询选择了c1、c4、c6课号的学生学号和成绩
- 4. 涉及空值的查询：列名 IS [NOT] NULL
  - 例：列出所有成绩为空的学生学号
- 5. 字符匹配：列名　[NOT] LIKE '<匹配串>'

通配符：% 表示任意多个字符 王% %卫% %卫

_ 表示任意一个字符 _卫_ __千 __平% 王__

例：查询姓名中包含“敏”的学生姓名

查询所有刘姓、王姓且为双名的学生姓名

单表查询—聚合函数

- 1. COUNT([DISTINCT]*)　　统计元组的个数
- 2. COUNT([DISTINCT]<列名>) 统计一列中值的个数
- 3. SUM([DISTINCT]<列名>)　计算一列值的总和
- 4. AVG([DISTINCT]<列名>)　计算一列值的平均值
- 5. MAX([DISTINCT]<列名>)　求一列值中的最大值
- 6. MIN([DISTINCT]<列名>)　求一列值中的最小值
  - 例：查询选过课的学生人数
  - 查询选课次数
  - 查询最大和最小年龄
  - 统计有多少名年龄超过19岁的女同学
  - 查询学号s1的学生考试成绩的平均分和总分

单表查询—分组排序

- 查询结果分组
  - 查询每门课的学生人数
  - 查询选课一门以上的学生学号
  - 统计男女同学的人数和平均年龄
  - 查询每门课程都超过70分的学生学号
- 查询结果排序
  - 查询每门课的学生人数和总成绩，查询结果按总成绩升序,人数降序排列
  - 统计每个学生选修课程的门数（超过5门的学生才统计）。要求输出学生学号和选修门数，查询结果按门数降序排列，若门数相同，按学号升序排列

联接操作

联接条件可在WHERE中指定也可以在 FROM子句中指定。

在 FROM 子句中指定联接条件时，SQL2将

联接操作符分成：联接类型、联接条件。

联接类型：决定了如何处理联接条件中不匹配的元组。

联接条件：决定了两个关系中哪些元组应该匹配。

联接类型中的OUTER字样可不写。

联接类型	联接类型说明
INNER JOIN	内联接：结果为两个联接表中的匹配行的联接。
LEFT OUTER JOIN	左外联接：结果包括“左”表（出现在 JOIN 子句的最左边）中的所有行。不包括右表中的不匹配行。
RIGHT OUTER JOIN	右外联接：结果包括“右”表（出现在JOIN 子句的最右边）中的所有行。不包括左表中的不匹配行。
FULL OUTER JOIN	完全外联接：结果包括所有联接表中的所有行，不论它们是否匹配
CROSS JOIN	交叉联接：结果包括两个联接表中所有可能的行组合。交叉联接返回的是两个表的笛卡儿积

多表连接查询

- 自身连接
  - 查询选课二门或二门以上的学生学号
  - 检索选修c1和c2课的学生学号，姓名
- 复合条件连接
  - 查找选修了《数据结构》课程且成绩在70分以上的姓名及成绩
  - 查询每门课程都超过70分的学生学号、姓名
  - 统计每个学生选修课程的门数（超过5门的学生才统计）。要求输出学生学号、姓名和选修门数，查询结果按门数降序排列，若门数相同，按学号升序排列

- 相关子查询
- 带有谓词IN的子查询
  - 例：查询选修了课程号为c2的课程的学生姓名
  - 查询选修了数据库原理课程的学生姓名
- 带有谓词ANY和ALL的比较子查询
  - 例：查询平均成绩超过所有女学生成绩的男学生的学生学号和姓名
- 带有谓词EXISTS的子查询：返回子查询是逻辑值
  - 例：查询没有选修c2课程的学生姓名
  - 查询所有学生都选修的课程号和课程名
  - 求选修学号为S3的学生所修全部课程的学生号码

第四节 SQL的数据更新

- 数据插入
- 数据删除
- 数据修改

一、数据插入

- **INSERT INTO <表名> [（<属性名清单>）]**
  VALUES （元组值）；
- **INSERT INTO <表名> （<属性名清单>）**
  VALUES （元组值），（元组值），…；
- **INSERT INTO <表名>[（<属性名清单>）]**
  （子查询）；
- 把子查询的结果插入指定的<表名>中。这样的一条INSERT语句，可以一次插入多条元组。
  - 例：创建一个含有学号、姓名和课程名的新表，然后将男同学的相关数据插入到新表中

二、数据删除

- DELETE　FROM <表名>
  ［WHERE <带有子查询的条件表达式>］
- 本语句将删除使<带有子查询的条件表达式>为真的所有元组。
  - 例：删除没选修任何一门课的学生信息
  - 把课程名为《数据库原理》的成绩从表SC中删除

三、数据修改

- **UPDATE <表名>**

  SET<列名>=<表达式>

  [，<列名>=<表达式>……]

  [WHERE <带有子查询的条件表达式>]

- 本语句执行时，将修改使<带有子查询的条件表达式>为真的所有元组。
  - 例：对低于70分的增加5%，高于等于70分的增加4%
  - 把课程名为《数据库原理》的成绩提高10%

第五节 视图

- 视图的定义
- 视图的查询
- 视图的更新
- 视图的优点

一、视图的定义

- 一个视图是从一个或多个关系（基本表或已有的视图）导出的关系。
- 视图是虚表，导出后，数据库中只存有此视图的定义（在数据字典中），但并没有实际生成此关系。
- 视图一经定义就可以象基本表一样进行查询和更新。

一、视图的定义

- CREATE VIEW <视图名>［<列名清单>］
  AS <子查询>
  ［WITH CHECK OPTION］
- 若有<列名清单>，则此清单给出了此视图的全部属性的属性名；否则，此视图的所有属性名即为子查询中SELECT语句中的全部目标列。
- 有［WITH CHECK OPTION］时，则今后对此视图进行INSERT、UPDATE和DELETE操作时，系统会自动检查视图是否符合原定义视图时子查询中的<条件表达式>。

例：在教学数据库中基本表SC上，建立一个学生学习情况视图：

```
CREATE VIEW S_GRADE(SNO,C_NUM,AVG_GRADE)
    AS (SELECT SNO,COUNT(CNO),AVG(GRADE)
        FROM SC
        GROUP BY SNO);
```

一、视图的定义

- **DROP VIEW <视图名>**
- 此语句将把指定视图的定义从数据字典中删除。
- 一个关系（基本表或视图）被删除后，所有由该关系导出的视图并不自动删除，它们仍在数据字典中，但已无法使用。
- 例：撤消 **S_GRADE** 视图，可用下列语句实现：
- **DROP VIEW S_GRADE;**

二、视图的查询

- DBMS对某SELECT语句进行处理时，若发现被查询对象是视图，则DBMS将进行下述操作：
- (1)从数据字典中取出视图的定义。
- (2)把视图定义的子查询和本SELECT的查询相结合，生成等价的对基本表的查询（此过程称为视图的消解）。
- (3)执行对基本表的查询，把查询结果（作为本次对视图的查询结果）向用户显示。

三、 视图的查询

系统在实现对视图的查询时，根据数据字典的定义将对视图的查询转换为对基本表的查询。

例：对学生学习情况视图执行如下操作：

① SELECT * FROM S_GRADE;

相应的查询转换操作如下：

```
SELECT SNO,COUNT(CNO) AS C_NUM,AVG(GRADE) AS AVG_GRADE
 FROM SC
 GROUP BY SNO;
```

②　SELECT SNO,C_NUM

FROM S_GRADE

WHERE AVG_GRADE>80;

相应的查询转换操作如下：

```
SELECT SNO,COUNT(CNO) AS C_NUM
        FROM SC
        GROUP BY SNO
              HAVING AVG(GRADE)> 80;
```

③ SELECT SNO,AVG_GRADE
FROM S_GRADE
WHERE C_NUM >(SELECT C_NUM
FROM S_GRADE
WHERE SNO='S4');

相应的查询转换操作如下：

SELECT SNO,AVG(GRADE) AS AVG_GRADE
FROM SC
GROUP BY SNO
HAVING COUNT(CNO)>(SELECT COUNT(CNO)
FROM SC
GROUP BY SNO
HAVING SNO='S4');

三、视图的更新

- 视图是虚表，是没有数据的。所谓视图的更新，表面上是对视图执行INSERT、UPDATE和DELETE来更新视图的数据，其实质是由DBMS自动转化成对导出视图的基本表的更新，转化成对基本表的INSERT、UPDATE和DELETE语句（用户在感觉上确实是在对视图更新）。
- 不是所有的视图都是可更新的，因为有些视图的更新不能有意义的转化成相应基本表的更新。

例：如果定义“计算机应用”学生视图：

```
CREATE VIEW STUDENT_COMPUTER(SNO,SNAME,SEX,AGE)
    AS SELECT SNO,SNAME,SEX,AGE
       FROM S
       WHERE SDEPT=‘计算机应用’;
```

该视图是从单个关系仅使用了选择和投影导出的，而且包括键SNO，因此是可以修改的。

如执行插入操作：

```
INSERT INTO STUDENT_COMPUTER
 VALUES(‘S99’, ‘王敏’, ‘男’,22);
```

系统自动会把它转换变成下列语句：

```
INSERT  INTO  S
VALUES（’S99’,‘王敏’,‘男’,22,‘计算机应用’ ）;
```

对于学生学习情况视图：

```
CREATE VIEW S_GRADE(SNO, C_NUM, AVG_GRADE)
        AS SELECT SNO, COUNT(CNO), AVG (GRADE)
            FROM SC
            GROUP BY SNO;
```

执行：UPDATE S_GRADE

```
        SET SNO='S3'
        WHERE SNO='S4';
```

不允许。C_NUM是对SC中的学生选修门数进行统计，在未更改SC表时，要在视图S_GRADE中更改门数，是不可能的。

执行： DELETE FROM S_GRADE

WHERE C_NUM>4；

也不允许的。在视图S_GRADE中删除选修门数在4门以上的学生元组，势必造成SC中这些学生学习元组的删除，这不一定是用户的原意，因此使用分组和聚合操作的视图，不允许用户执行更新操作。

三、视图的更新

- 一般的DBMS只允许对单个基本表导出的视图进行更新。并有下列限制：
- ①若视图的列由表达式或常数组成，则不允许执行INSERT和UPDATE，但可执行DELETE。
- ②若视图的列由集函数组成，则不允许更新。
- ③若视图定义中有GROUPBY子句，则不允许更新。
- ④若视图定义中有DISTINCT选项，则不允许更新。
- ⑤若视图定义中有嵌套查询，且内外层FROM子句中的表是同一个表，则不允许更新。
- ⑥从不允许更新的视图导出的视图是不允许更新的。

四、视图的优点

- 1．视图能方便用户操作，若用户所需数据来自多个基本表，则通过视图可使用户感到数据是来自一个关系的；若用户所需数据是对基本表中的数据通过某种运算才能得到的，
- 2．视图可对数据提供安全保护
- 3．视图能使不同用户都能用自己喜欢的方式看待同一数据同一数据，在不同用户的各个视图中，可以以不同的名称出现，可以以不同的角色出现（平均值，最大值……）。这给数据共享带来了很大的方便。

第六节 嵌入式SQL

- SQL语言的运行环境
- 嵌入式SQL的使用规定
- 嵌入式SQL的使用技术
- 动态SQL语句

一、SQL语言的运行环境

- 常用的方式是用某种传统的编程语言（例如：C、PASCAL等）编写程序，但程序中的某些函数或某些语句是SQL语句。这种方式下使用的SQL语言称为嵌入式SQL（EmbeddedSQL），其中传统的编程语言称为宿主语言（或主语言）。
- DBMS有两种方法处理嵌入式SQL语言：预编译和扩充编译程序法。预编译是指由DBMS的预编译器对源程序进行扫描，识别出其中的SQL语句，把它们转换为宿主语言调用语句，使宿主语言编译器能够识别，最后由编译器将整个源程序编译为目标码。目前使用较多的是预编译方法，其处理过程如图4—2所示。

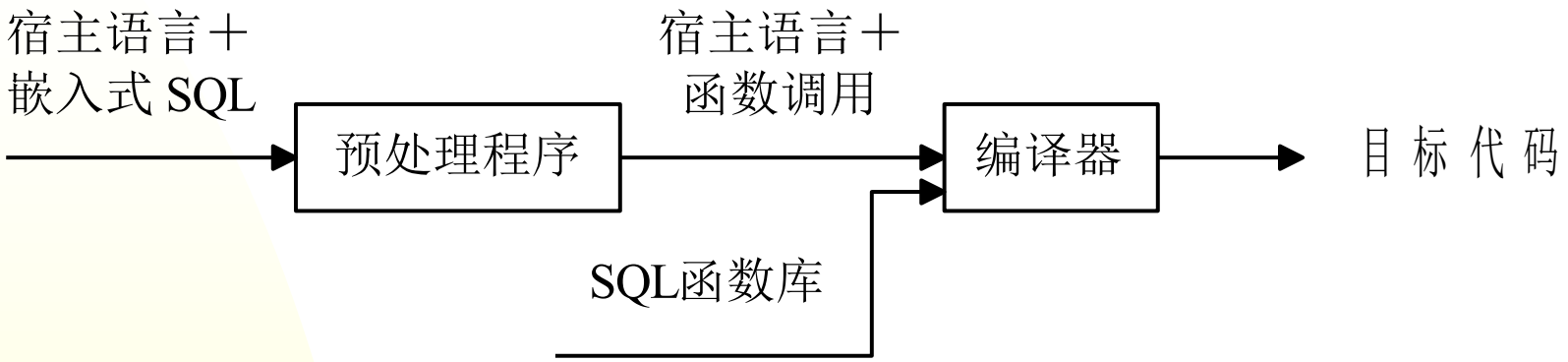

图4—2 嵌入式SQL语句的处理过程

一、SQL语言的运行环境

- 使用嵌入式SQL必须解决以下几个问题：
  - (1)预编译器不能识别和接受SQL语句，因此，嵌入式程序中，应有区分SQL语句与宿主语言语句的标记。
  - (2)DBMS和宿主语言程序（程序工作单元）如何进行信息传递。
  - (3)一条SQL语句原则上可产生或处理一组记录，而宿主语言一次只能处理一个记录，必须协调这两种处理方式。

二、嵌入式SQL的使用规定

- 对嵌入的SQL语句加前缀EXEC SQL，而结束标志则随宿主语言的不同而不同。在C语言中嵌入的SQL语句以EXECSQL开始，以分号“；”结尾：EXEC SQL＜SQL语句＞；
- 在DBMS和宿主语言程序之间的数据传递，是通过宿主语言程序变量，简称主变量（Host variable）来实现的。当SQL语句引用主变量时，变量前应加冒号“：”。
- SQL语言和宿主语言的不同数据处理方式，是通过游标（Cursor）来协调的。游标是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果。每个游标都有一个名字。

二、嵌入式SQL的使用规定

- 用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，由宿主语言作进一步的处理。游标的操作包括四个步骤：
  - 定义游标：**EXEC SQL DECLARE** 游标名 **CURSOR FOR**＜**SELECT**语句＞;
  - 打开游标：**EXEC SQL OPEN** ＜游标名＞;
  - 推进游标：**EXEC SQL FETCH**＜游标名＞ **INTO** ＜主变量名列表＞;
  - 关闭游标：**EXEC SQL CLOSE**＜游标名＞;

三、嵌入式SQL的使用技术

- 不涉及游标的嵌入式**SQL DML**语句
- 涉及游标的嵌入式**SQL DML**语句
- 滚动游标的定义和推进

三、嵌入式SQL的使用技术

(1) 在嵌入式SQL中， SQL的数据定义DDL与控制语句DCL都不需要使用游标。

它们是嵌入式SQL中最简单的一类语句，不需要返回结果数据，也不需要使用主变量。在主语言中嵌入SQL说明性语句（DECLARE）及控制语句（GRANT），只要给语句加上前缀EXEC SQL和语句结束符END_EXEC即可。在C语言中，用分号；代替END_EXEC

例：在C语言中说明共享变量：

```
EXEC SQL BEGIN DECLARE SECTION
 int grade, raise;
 char givencno[5],cname[13], tname[9] ;
 char givensno[5],sname[9],sdept[11];
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
```

(2) 不涉及游标的嵌入式SQL DML语句

a. 对于INSERT、DELETE和UPDATE语句， 只要加上前缀标识“EXEC SQL”和结束标志“END_EXEC”，就能嵌入在宿主语言程序中使用。例：

① 在关系C中插入一门新的课程，各属性值已在相应的共享变量中：

```
EXEC SQL INSERT  INTO C（CNO,CNAME,TNAME）
                VALUES(:givencno,:cname,:tname);
```

② 从关系SC中删除一个学生的所有选课，

该学生的姓名由共享变量sname提供。

```
EXEC SQL DELETE    FROM SC
                WHERE SNO=(SELECT SNO
                                FROM S
                                WHERE SNAME=:sname);
```

③ 把“数据库”课程的全部成绩增加某个值（该值由共享变量raise 提供）。

```
EXEC SQL UPDATE SC
        SET GRADE=GRADE +:raise
        WHERE CNO IN
           (SELECT CNO
             FROM C
             WHERE CNAME=‘数据库’);
```

b. 对于SELECT语句，如果已知查询结果肯定是单元组时，可直接嵌入在主程序中使用，此时在SELECI语句中增加一个INTO子句，指出找到的值应送到相应的共享变量中去。

例：在关系S中根据共享变量givensno的值检索学生的姓名和所在系。

例：在关系S中根据共享变量givensno的值检索该学生的姓名和所在系：

```
EXEC SQL  SELECT SNAME,SDEPT
                INTO :sname,:sdept
                FROM S
                WHERE SNO=:givensno;
```

此处sname,sdept,givensno 都是共享变量，已在主程序中定义，并用SQL的DECLARE语句加以说明，在引用是加上“：”作为前缀标识，以示与数据库中变量区别。

(3) 涉及游标的嵌入式SQL DML语句

a、当SELECT语句查询结果是多个元组时，此时要用游标机制把多个元组一次一个地传送给宿主语言程序处理。

例:在关系SC表中检索某学生(学生名由共享变量givensname给出)选课信息（SNO,CNO,GRADE），

该查询的C语言程序段：

```
EXEC SQL BEGIN DECLARE SECTION;
  Int grade, rise;
  Char sno[5],cno[5], givensname[9], SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE scx CURSOR FOR
    SELECT SNO,CNO,GRADE
    FROM SC
    WHERE SNO= (SELECT SNO
                FROM S
                WHERE SNAME=:givensname)
    FOR UPDATE  OF  GRADE;
EXEC SQL OPEN scx;
```

```
While(1)
  { EXEC SQL FETCH FROM scx
                INTO :sno,:cno,:grade;
  If (SQLCA.SQLSTATE = ='02000') /*已取完查询结果中的所有元组*/
      Break;
  If (SQLCA.SQLSTATE != '0')    /*取数据出错*/
     Break;
      ...                          /*对游标所取的数据进行处理*/
      printf("%s,%s,%d",sno,cno,grade);
}
EXEC SQL CLOSE scx;
```

b. 对游标指向的元组进行修改或删除操作

当游标处于活动状态时，可以修改或删除游标指向的元组。

例：在上面的例子中，对找到的元组做如下处理：

删除不及格的选课,将60~69分的成绩增加由共享变量rise提供的值,再显示该学生的成绩信息(SNO,CNO,GRADE).

在上例中的”While(1){...}语句改为如下形式：

```
While(1)
{ EXEC SQL FECCH FROM scx
            INTO   :sno,:cno,:grade;
If (SQLCA.SQLSTATE = =’02000’) /*已经取完查询结果中的所有元组*/
   Break;
If (SQLCA.SQLSTATE ! = ’0’)    /*  取数据出错*/
    Break;
    If (grade<60)
        EXEC SQL DELETE FROM SC
                    WHERE CURRENT OF scx;
    Else
       {If (grade<70)
```

```
    {EXEC SQL UPDATE SC
                SET GRADE=GRADE+:rise
                WHERE CURRENT OF scx;
                grade=grade+rise;}
        printf("%s,%s,%d",sno,cno,grade);
    }
}
```

精读和上机、习题要求

精 读： 教材 P.67~P.97 P.260~P.290 P.305~P.372

上 机：

1. 实验一 要求见教材实验题 P.300
2. 实验二 P.301
3. 实验三 P.371
4. 实验四 P.372

习 题： P.100 4.2 4.6 4.7 4.9