内容回顾

- **SQL语言的基本特征**
- 数据定义功能：**Create Table、Create Index**
- 数据查询功能**Select**
- 单表查询、连接查询、嵌套查询、集合查询

SQL语言的基本特征

- 一体化的特点
- 两种使用方式，统一的语法格式
- 高度非过程化
- 语言简洁、易学易用
- SQL语言也支持关系数据库三级模式体系结构
  - 外模式：视图+一些基本表
  - 模式(概念模式）：基本表
  - 内模式：存储文件(逻辑结构)

SQL数据定义功能

- 创建表(模式)、索引(内模式)和视图(外模式)

一般格式如下：

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

（）必须有的内容,[ ] 可有可无,自定义的内容

SQL的查询功能

- SQL查询语句的基本结构

SELECT [ALL|DISTINCT] 目标列

FROM 基本表（视图) 范围变量名,….

[WHERE 条件表达式]

[Group by 列名1 [having 分组表达式]]

[Order By 列名2 asc | desc];

查询条件

结果输出条件

SQL语言（2）

Structured Query Language

- 数据插入
- 数据删除
- 数据修改

数据插入（对表中内容的操作，即表中的数据，而非表的定义）

一般格式：

INSTERT INTO <表名> [(<属性列1> [, <属性列2>…)]] VALUES (<常量1> [, <常量2>…)；

说明：对into子句中没有出现的且可以取Null的属性列，新记录将在这些列上取null。

数据插入（举例）

- 单记录（元组）插入
  - insert into Students values(‘95020’, ‘陈东’, ‘男’, 18,‘IS’)；
  - insert into SC(Sno, Cno) values(’95020', ’1')；
- 多记录插入(插入子查询结果)

假定数据库中有表:

deptage(Sdept char(15), Avgage smallint)

半字长二进制整数

双字，字，字节，位

```
insert into deptage ( sdept, avgage )
        select   Sdept,  avg( Sage )
        from     Students
        group    by   Sdept；
```

数据删除（对表中内容的操作）

一般格式：

DELETE FROM <表名> [WHERE <条件>];

数据删除（举例）

- delete from Students where Sno = '95019’;
- delete from SC;
- 带有子查询的删除：删除计算机系所有学生的选课记录

```
delete from SC
 where 'CS' =
        ( select Sdept
          from   Students
          where  Students.Sno = SC.Sno );
```

数据更新（修改）

使用“更新”更准确一些，指对数据库中相关联的所有表中的数据进行“刷新”操作。

一般格式：

UPDATE <表名> **SET** <字段名> = <表达式> [**WHERE** <条件>];

数据更新（修改）举例

- update Students set Sage = 22
  where Sno= ‘95001’
- update Students set Sage = Sage + 1
- 带有子查询的修改：将计算机系全体学生的成绩置零

```
update SC set G = '0'
where ‘CS' =
      ( select Sdept
        from Students
        where Sno=SC.Sno )
```

更新操作与数据库的一致性

- 更新操作从表面看只对一个表操作, 但实际中可能是对相关联的一系列表进行操作:

例如: 请删除学号为95019的学生, 隐含将其所有选课记录删去

1. delete from students
   where Sno='95019';
2. delete from SC where Sno='95019';

- 如果建表时, 说明SC参照Students存在, 则按以上顺序进行操作是2中的删除操作失败。

更新操作与数据库的一致性

- 数据库提供事务概念处理这类问题
- 如果建表时，说明SC参照Students存在，且说明（勾选）on delete cascade功能，则只要

```
delete  from students
where Sno='95019';
```

便可将95019及其相关的所有信息，包括选课记录全部删去。

视图

- 视图是从一个或几个基本表（视图）导出的表
- 视图是虚表：数据库中只存放视图的定义(存放于数据字典中)，不存放视图对应的数据（数据在相应的基本表中）
- 视图也称动态窗口：
- 视图可以和基本表一样被查询，被删除
- 视图的更新是有一定限制的（最终要转换为表）

官方称为数据字典或系统目录，它通常是一系列表，表中存储着数据库中的有关信息（数据库三级模式、数据类型、用户名表、用户权限、程序与其用户联系等有关数据库系统的信息）的当前描述，起着系统状态的目录表的作用，它能帮助用户、数据库管理员和数据库管理系统本身使用和管理数据库。其表现形式类似于图书馆的图书分类编目。

视图定义

一般格式：

CREATE VIEW <视图名>[(<列名>[, <列名>]…)] AS <子查询> [WITH CHECK OPTION];

With check option 表示对视图进行 **UPDATE,INSERT,DELETE**操作时要保证更新插入或删除的行满足视图定义中的谓词条件。

视图定义举例

- 例子：建立信息系学生的视图
  - **create view IS_Students as**

    select * from Students

    where Sdept='IS';
  - **create view IS_Students(No, Name, Dept) as select Sno, Sname, Sdept from Students where Sdept='IS';**

视图的属性名（是不是可以全部省略或者全部列出？）

视图属性名的书写规则

- 视图的属性或者全部指定，或者全部不指定
  - 换名
  - 某个目标列是集函数或列表达式
  - 多表连接时有几个同名列需要区分（how？）
- 例子：将学生的学号及它的平均成绩定义为一个视图

```
create view avgGrade(Sno, avgG) as
    select Sno, avg(Grade)
    from SC
    group by Sno
```

虚拟列

视图定义举例

- 例子：建立信息系统选修了1号课程且成绩在90分以上的学生的视图

```
Create view IS_Students(Sno, Sname, Grade)
as  select Students.Sno, Sname, Grade
    from Students, SC
    where Students.Sno=SC.Sno and Cno='1'
    and Grade>90;
```

- with check option子句使用
  - **create view IS_Students as**

```
select * from Students
where Sdept='IS'
with check option;
```

查询视图（系统将此转换为对表的查询：这一过程叫视图消解）

- 视图查询执行过程: 把定义中的子查询和用户查询结合起来, 转化成等价的对基本表的查询
- 例子: 在信息系的学生中找到年龄小于20岁的学生

```
select * from IS_Students
where Sage<20;
```

- 转换后的查询为(系统完成):

```
select * from Students
where Sdept='IS' and Sage<20;
```

- 查询信息系选修了1号课程的学生姓名和成绩

```
select Sname, Grade from IS_Students, SC
where IS_Students.Sno=SC.Sno and Cno='1';
```

查询视图

- 对视图的查询有些是不能直接进行转换的（指在相同位置等价转换）
  - 例子: 在avgGrade视图中, 查询平均成绩在90分以上的学生学号和平均成绩

```
select * from avgGrade
where avgG>=90;
```

  - 转换:

```
select Sno, avg(Grade)
from SC
where avg(Grade)>90
group by Sno; Having avg(Grade)>90;?
```

/* where子句中不能以集函数作为条件表达式 */

视图更新

- 对视图更新最终转换成对基本表的更新
- 将信息系学生学号为95002的学生姓名改为“刘辰”

```
update IS_Students set Sname='刘辰'
where Sno='95002';
```

- 通过视图IS_Students插入一信息系的学生记录

```
insert into IS_Students
values('95029', '赵新', '女', 20);
```

视图更新

- 删除信息系学号为95029的记录

```
delete from IS_Students
where Sno='95029';
```

- 视图的作用: (简化性，多样性，可重构性和保护性)
  - 简化用户的操作
  - 视图使用户能以多种角度看待同一数据
  - 视图可以对机密数据提供一定的安全保护

视图的作用

- 视图对数据库重构提供了一定程度的逻辑独立性
  - 例如: Students(Sno, Sname, Ssex, Sage, Sdept)

    分为: S1(Sno, Sname, Sage)

    S2(Sno, Ssex, Sdept)
  - 但在Students上已有开发了多个应用, 为了达到不改变应用程序的目的, 可以建视图Students:

```
create view Students(Sno, Sname, Ssex, Sage,
                     Sdept) as
        select S1.Sno, S1.Sname, S2.Ssex,
               S1.Sage, S2.Sdept
        from Students S1, Students S2
        where S1.Sno=S2.Sno;
```

数据控制

- 数据库中数据的操作权限(增、删、改、查)。
- SQL语言的安全控制功能：设计者通过SQL语言的grant 和revoke语句告诉系统某个用户对某类数据所具有的操作权限。
- SQL定义的这些信息存于数据字典中。
- 当用户提出操作请求时，系统查看数据字典，根据该用户的权限决定是否执行该请求。

系统

授权

- Grant语句的一般格式

  grant <权限>[,<权限>]…

  [on <对象类型> <对象名>]

  to <用户>[,<用户> …]

  [with grant option];

- 把指定对象的一些操作权限授给一些用户
- 例子: 把查询Students表的权限授给用户wang

  grant select on table Students to wang;
- 例子: 把查询Students和SC表的权限授给全部用户

  grant select on table Students, SC to public;

授权

- with grant option子句

  例子: 把对表SC的查询权限、修改成绩权限授给wang和zhang, 并允许wang和zhang将该权限授予他人

```
grant select, update(Grade) on table SC
    to wang, zhang
    with grant option;
```

- DBA、对象的建立者和经过with grant option授权的用户可以把他们对该对象具有的操作权限授予其它的合法用户
- DBA把在数据库S-C中建立表的权限授予zhao

```
grant createtab on database S-C to zhao;
```

授权

- 不同对象允许的操作权限

对象	对象类型	操作权限
属性列	table	select, update, all privileges
视图	table	select, update, insert, delete, all privileges
基本表	table	select, update, insert, delete alter, index, all privileges
数据库	database	createtab

DBA具有在数据库中建立表的权限

- 用户仅能根据自己所拥有的权限来操作数据库中的数据

收回授权

- 授出的权限可以由DBA或其他的授权者收回

  revoke <权限>[,<权限>]…

  [on <对象类型> <对象名>]

  from <用户>[,<用户> …]

- 例子: 把用户wang和zhang修改成绩的权限收回

  revoke update(Grade) on table SC

  from wang, zhang;

- 权利回收操作是级联的

  例如: DBA → user1 → user2

  user2 → user3

授予SC表的select权限

收回授权

```
revoke select
 on table SC
 from user1
```

从user1、user2和user3收回对SC表的select权限

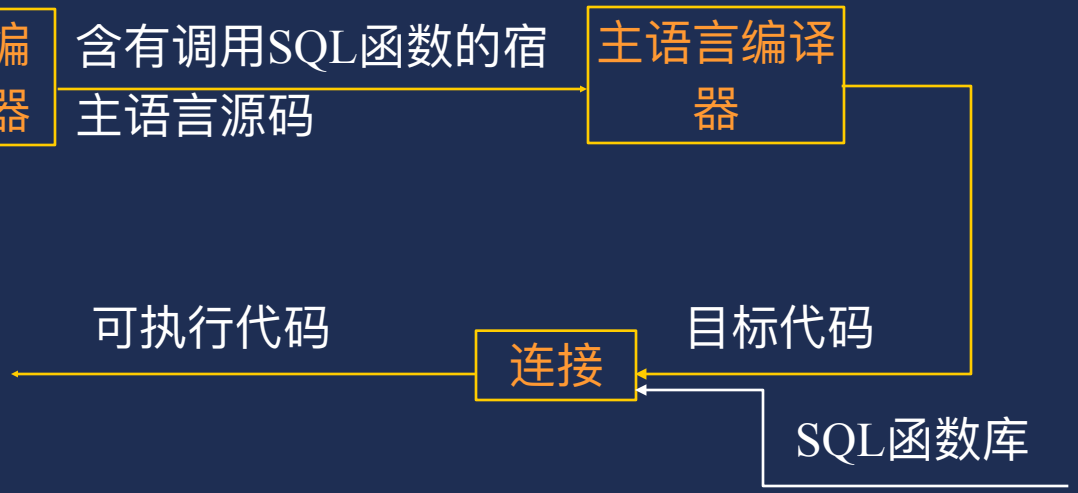
嵌入式SQL

- 实现复杂应用：高级语言计算完备性+SQL语言的高效处理
- 主语言(宿主语言), 嵌入式SQL
- 区分SQL语句和主语言的语句
- SQL语句可以应用主变量
- SQL语句的查询结果是一个集合,嵌入式SQL提供游标(Cursor) 机制来处理查询结果
- 嵌入式SQL的处理过程：数据库厂商提供
  - 预编译器
  - 修改和扩充主语言编译器

嵌入式SQL

- 例如：

Ø 嵌有SQL的宿主语言源码 → 预编译器 → 含有调用SQL函数的宿主语言源码 → 主语言编译器 → 目标代码 → 连接（SQL函数库）→ 可执行代码



- 关系DBMS一般提供一批用宿主语言编写的SQL函数，组成SQL函数库，供应用程序调用DBMS的各种功能

嵌入式SQL

- SQL函数库实际上是DBMS向应用程序提供的一种接口, 称为调用级接口(CLI)
- 预编译器将嵌入SQL语句编译成宿主语言对SQL函数的调用
- 95年ISO公布了一个CLI标准, 作为SQL-92的一个附件,称为SQL-92/CLI或CLI95, 新的RDBMS都支持该标准, 标准中只规定了函数的定义

嵌入式SQL

- 嵌入式SQL 的一般格式
  - 所有SQL语句必须加前缀EXEC SQL和语句结束标志(C语言中一般是“;” 号)
  - EXEC SQL select Sno from SC
    where Cno=‘1’;
  - 嵌入式SQL语句可分为: 可执行语句和说明性语句
- 嵌入式SQL语句与主语言语句之间的通信
  - SQL通信区(SQLCA): 每个SQL语句执行后, 系统要反馈给应用程序若干信息, 这些信息存于SQLCA中

嵌入式SQL

- 应用程序从SQLCA中取出这些状态信息, 并据此决定应用程序的进一步执行
- SQLCA是一个数据结构, 由系统定义, 程序员只需在程序中加入

  “EXEC SQL INCLUDE SQLCA;” 便可以引用其中的分量
- 分量SQLCODE是一整型变量, 系统将每个SQL语句执行成功与否的结果赋给该变量
  - 0: 成功
  - 正数: 已执行但有异常(其中100表示没取到数)
  - 负数:表示SQL语句因某些错误而未被执行, 负数的值表示错误的类别

嵌入式SQL

- 主变量(宿主变量)

嵌入式SQL使用主语言的变量输入/输出数据,

- 主变量的说明(可以和属性同名)

例子: EXEC SQL BEGIN DECLARE SECTION;

```
char Sno[7],
char givenSno[7],
char Cno char[2],
float Grade,
short GradeIn;
EXEC SQL END DECLARE SECTION;
```

- 主变量的使用

在SQL中使用主变量时, 主变量前以定加“:”

嵌入式SQL

- 主变量的使用
  - 在SQL中使用主变量时, 主变量前以定加“:”
  - 例子: EXEC SQL select Sno, Cno, Grade
    into :Sno, :Cno, :Grade
    from SC
    where Sno=:givenSno;
- 指示变量(主变量的一种)
  - 一个主变量可以附带一个指示变量, 指示变量也必须说明, 使用时放在要主变量之后
  - 例子:

嵌入式SQL

```
EXEC SQL select Sno, Cno, Grade
                into :Sno, :Cno, :Grade :GradeIn
                from SC
                where Sno=:givenSno;
```

- 指示主变量的取值情况(是否为空值), 当GradeIn<0时, 表明成绩为空值
- 指示变量不允许用在where子句中

游标

- SQL与主语言具有不同的处理方式, 用游标协调
- 游标区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, 每个游标区有一个名字
- 游标的说明和使用(举例说明)

嵌入式SQL

```
EXEC SQL DECLARE cursor1 CURSOR FOR
        select Sno, Cno, Grade
        into :Sno, :Cno, :Grade
        from SC;
EXEC SQL OPEN cursor1;
while(1) {
    EXEC SQL FETCH cursor1;
    if(sqlca.sqlcode)<>0
        break;
    printf("Sno:%s, Cno:%s, Grade:%d", Sno, Cno, Grade);
}
EXEC SQL CLOSE cursor1;
```

不使用游标的SQL语句

- 不使用游标的sql语句
  - Insert语句
  - update(不带current子句)语句
  - delete(不带current子句)语句
  - 查询结果为单记录的语句

特点

- 例子:
  - 例子1: 查询某个学生选修某门课程的成绩, 要求输出学号, 课号和成绩

```
exec sql include sqlca;
exec sql begin declare section;
        char Sno[7],
        char givenSno[7],
```

不使用游标的SQL语句

```
   char Cno char[3],
   givenCno char[3],
   float Grade,
   short GradeIn,
exec sql end declare section;
main() {
    printf(“\n请输入学号:”)
    scanf(“%s”, Sno);
    printf(“\n请输入课号:”)
    scanf(“%s”, Cno);
```

不使用游标的SQL语句

```
 exec sql select Sno, Cno, Grade
              into :Sno, :Cno, :Grade: GradeIn
              where Sno=:givenSno and Cno=:givenCno;
 if(sqlca.sqlcode=0)
   if (GradeIn<0)
        printf("\n%s, %s, %d", Sno, Cno);
   else
   printf("\n%s, %s, %d", Sno, Cno, Grade);
 else
   printf("\n没有查到所需数据\n");
}
```

不使用游标的SQL语句

- 例子2: 将学生的年龄增加1岁

```
exec sql update Students
        set Sage=Sage+1;
```

- 例子3: 将学生的1号课成绩增加指定的分数, 要求从键盘上输入增加的值

```
exec sql update SC
        set Grade=Grade+:Raise
        where Cno='1';
```

- 例子4:将学生的1号课成绩置空

```
exec sql update SC
        set Grade=Grade+:Raise : GradeIn
        where Cno='1';
```

不使用游标的SQL语句

- 例子5: 某学生退学了, 将其所有选课记录删除, 要求输入学生的学号

```
exec sql  delete from SC
          where Sno=:givenSno;
```

- 例子6: 某个学生选修了某门课程, 将有关记录插入SC表中, 要求从键盘输入学生的学号、课号, 成绩为空

分析: 说明主变量, 将指示变量赋初值-1

```
exec sql insert into SC
          values(:Sno, :Cno, :Grade :GradeIn)
```

使用游标的SQL语句

- 使用游标的SQL语句

  查询结果为多条记录的select语句

  带有current的update语句

  带有current的的delete语句

- 例子:

  - 例子1: 查询某个系全体学生的信息, 要求系名由键盘输入

```
exec sql include sqlca;
exec sql begin declare section;
        char Sno[7], Sname[11], Ssex[3];
        char givenSdept[20];
        short Sage;
```

使用游标的SQL语句

```
exec sql end declare section;
main() {
    EXEC SQL DECLARE cursor2 CURSOR FOR
            select Sno, Sname, Ssex, Sage
            into :Sno, :Sname, :Ssex, :Sage
            from Students
            where Sdept=:givenSdept;
    printf("\n请输入要查询的系:");
    scanf("%s", gevenSdept);
    EXEC SQL OPEN cursor2;
    while(1) {
        EXEC SQL FETCH cursor2;
```

使用游标的SQL语句

```
        if(sqlca.sqlcode)<>0
            break;
        printf("\n%s, %s, %s, %d", Sno, Sname, Ssex, Sage);
    }
    EXEC SQL CLOSE cursor2;
}
```

- 例子2: 查询某个系全体学生的信息, 要求系名由键盘输入, 并根据用户需要修改某些记录的年龄字段

```
exec sql include sqlca;
exec sql begin declare section;
        char Sno[7], Sname[11], Ssex[3];
        char givenSdept[20];
        short Sage;
```

使用游标的SQL语句

```
exec sql end declare section;
main() {
    EXEC SQL DECLARE cursor3 CURSOR FOR
            select Sno, Sname, Ssex, Sage
            into :Sno, :Sname, :Ssex, :Sage
            from Students
            where Sdept=:givenSdept
            for update of Sage;
    printf("\n请输入要查询的系:");
    scanf("%s", gevenSdept);
    EXEC SQL OPEN cursor3;
    while(1) {
```

使用游标的SQL语句

```
EXEC SQL FETCH cursor3;
if(sqlca.sqlcode)<>0
        break;
printf("\n%s, %s, %s, %d", Sno, Sname, Ssex, Sage);
printf("\n需要修改吗?");
scanf("%c" &yn);
if(yn='y' or yn='Y'){
   printf("\n输入要修改的值:");
   Scanf("%d", givenSage);
   exec sql update Students
        set Sage=:givenSage
        where current of cursor3; }
```

使用游标的SQL语句

```
    }
    EXEC SQL CLOSE cursor3;
}
```

- 带有current的删除语句处理同上
- 当游标中的select带有union或order by 子句时, 不能使用带有current的update和delete
- 当游标中的select相当定义了一个不可更新的视图时, 不能使用带有current的update和delete

动态SQL

- 静态SQL: 编写程序时, SQL语句是已知的, 只有一 些主变量的值需要输入的
- 动态SQL: 如果想查询的列不固定, 查询的条件不固定, 这样通用的查询程序必须通过动态SQL实现
- 动态SQL允许在程序运行过程中, 临时组装SQL语句, 主要有三种形式:
  - 语句可变
  - 条件可变
  - 数据库对象和查询条件都可变
- 例子:
- 例子1: 删除表Students中的某些学生记录（条件不定）

动态SQL

```
exec sql begin declare section;
 char sqlString[200];
 exec sql end declare section;
 char conditons[150];
 main() {
      strcpy(sqlString, “delete from students where ”);
      printf(“enter delete condition:”);
      scanf(“%s”, conditions);
      strcat(sqlString, conditons);
      exec sql execute immediate :sqlString; …
 }
```

以分号结尾

动态SQL

- 例子2: 删除某年以前出生的某些学生记录, 要求由键盘输入年份(仅给出程序段)

```
exec sql begin declare section;
char sqlString[200];
int birthYear;
exec sql end declare section;
….
strcpy(sqlString, “delete from Students where 2001-Sage<=:y;”);
printf(“Enter birthYear for deleting:”);
Scanf(“%d”, &birthYear);
exec sql prepare purge from :sqlString;
```

SQL中带有参数

不在任何地方说明, 仅占位

动态SQL

exec sql execute purge using :birthYear;

….

- 例子3: 求选修某门课程号的学生和成绩, 按什么排序由查询者指定(仅给出程序段)

…. //主变量说明

查询需要返回结果,但是否是多个元组,在编程时不能确定

```
char orderBy[150];
strcpy(sqlString, “select Sno, Grade from SC where
  Cno=:C”);
printf(“输入orderby子句:”);
scanf(“%s”, orderBy);
strcat(sqlString, orderBy);
printf(“输入课程号:”);
```

动态SQL

```
scanf("%s", givenCno);
exec sql prepare query from :sqlString;
exec sql declare cursor4 cursor for query;
exec sql open cursor4 using :givenCno;
while(1) {
    exec sql fetch cursor4 into :Sno, :Grade :GradeIn;
    if(sqlca.sqlcode==100)
        break;
    if(sqlca.sqlcode<0)
        break;
        …
  }
```

结 束