嵌入式SQL

北京理工大学　计算机学院
张文耀
zhwenyao@bit.edu.cn

第4章 关系数据库标准语言SQL

- 4.1SQL简介
- 4.2SQL的系统结构
- 4.3SQL的数据定义
- 4.4SQL的数据操纵
- 4.5SQL中的视图
- 4.6SQL的数据控制
- 4.7嵌入式SQL
- 4.8小结

主要内容

- 嵌入式SQL概述
- 嵌入式SQL与主语言的接口
- 不用游标的嵌入式SQL
- 用游标的嵌入式SQL
- 嵌入式SQL 应用实例
- 动态SQL
- 嵌入式SQL小结

嵌入式SQL概述

- **SQL语言提供了两种不同的使用方式**
  - 交互式
  - 嵌入式

    SQL嵌入到C, C++, Java, COBOL等程序设计语言中，称为嵌入式SQL(Embedded SQL)。
- 宿主语言（主语言）

  被嵌入**SQL**的程序设计语言，称为宿主语言，如**C、C++、Java**等，称为宿主语言，简称主语言。
- 引入嵌入式**SQL**的原因
  - **SQL**语言是非过程性语言
  - 事务处理一般是过程性的，需要应用高级语言

- 将SQL嵌入主语言后
  - SQL语句负责数据库的操纵
  - 宿主语言负责控制程序流程和数据的输入输出
- 嵌入式SQL和交互式SQL
  - 主要命令语句相同
  - 实现细节上有所不同
  - 嵌入式SQL做了某些必要的扩充

嵌入式SQL与主语言的接口

- 将SQL语句嵌入到宿主语言中必须解决的问题
  1、将嵌入主语言的SQL语句编译成为可执行代码
  2、数据库和主语言程序间的通信
  3、数据库和主语言程序间的数据交换
  4、需要协调面向集合和面向记录两种不同的处理方式

1.嵌入式SQL的编译

- 采用预编译方法
  - 在编译之前，先对**SQL**语句进行预处理；
  - 通过执行预处理程序把**SQL**语句变为主语言能够识别的形式；
  - 由主语言编译器统一对预处理后的源程序进行编译。
- 预处理要求：
  - 所有**SQL**语句必须加前缀**EXEC SQL**，以区分**SQL**语句与主语言语句。
  - 有的主语言还在**SQL**语句结尾加上结束标志。
  - 在**C**语言嵌入式**SQL**的一般格式为：

    EXEC SQL <SQL语句>;

- 以COBOL作为主语言的嵌入式SQL语句的一般形式

  EXEC SQL <SQL语句> END-EXEC

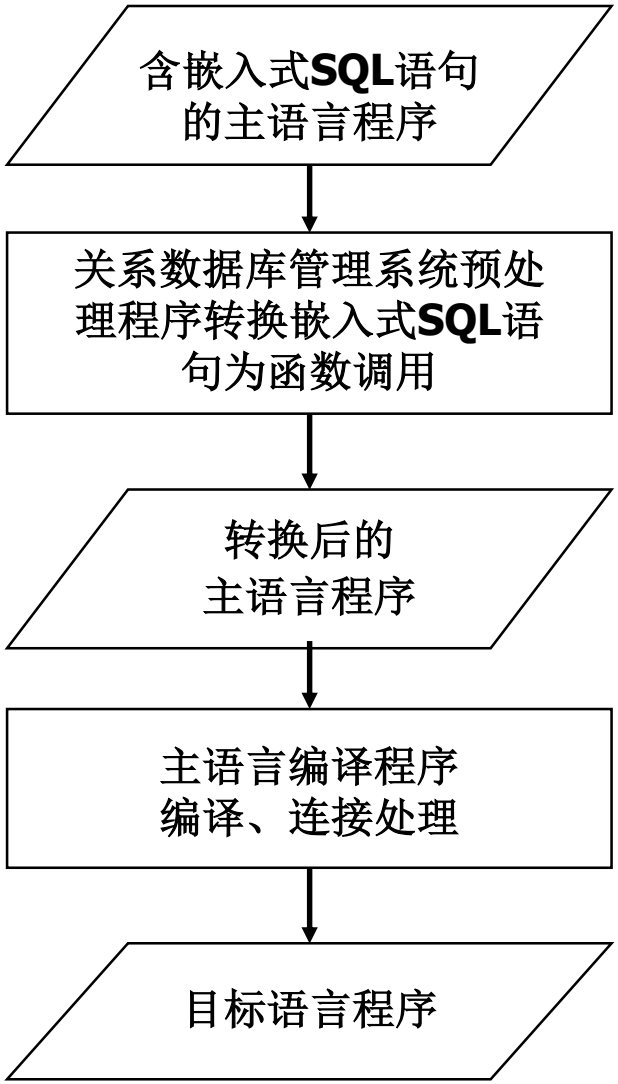
- 嵌入SQL的C语言源程序的基本编译过程

  1）使用预编译器对含有嵌入式SQL语句的C语言源程序进行预处理，把SQL语句变为C语言编译器可识别的命令语句。预处理包括语法检查、存取权限检查、路径选择等。SQL语句处理后生成调用语句。

  2）对预处理后的源程序用C编译器进行编译，生成目标代码。

  3）连接目标代码，包括C语言库函数和SQL库函数，生成可执行程序。

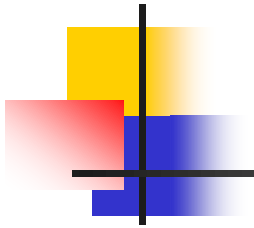
嵌入式SQL的基本编译过程



2.数据库和主语言程序间的通信

- 数据库的**DBMS**负责数据存取，**SQL**语句的执行是否成功，其执行结果需反馈给应用程序
  - 在**SQL**中设有一通信区**SQLCA(SQL Communication Area)**
    - 一个数据结构，其中包含描述**DBMS**当前工作状态的若干信息
  - 应用程序和用户通过**SQLCA**了解数据库的执行情况
    - **SQLCA**中有一状态指示单元**SQLCODE**，用于存放**SQL**语句的执行结果，通过判断**SQLCODE**的值可以知道**SQL**语句的执行情况（**0**表示成功）
  - 应用程序引用**SQLCA**，需加以说明，格式为：

    EXEC SQL INCLUDE SQLCA;



- **SQLCA的具体内容**
  - 与所执行的**SQL**语句有关
  - 与该**SQL**语句的执行情况有关

3.数据库和主语言程序间的数据交换

- 嵌入式SQL语句通过主变量与主语言程序进行数据交换。
- 在SQL语句中使用的主语言程序变量简称为主变量（Host Variable）或宿主变量。按用途分：
  - 输入变量
    - 指定向数据库中插入的数据；
    - 将数据库中的数据修改为指定值；
    - 指定执行的操作；
    - 指定WHERE子句或HAVING子句中的条件；
  - 输出变量
    - 获取SQL语句的结果数据
    - 获取SQL语句的执行状态

- 主变量的使用
  - 输入主变量
    - 由应用程序对其赋值，**SQL**语句引用
  - 输出主变量
    - 由**SQL**语句赋值或设置状态信息，返回给应用程序
  - 一个主变量有可能既是输入主变量又是输出主变量
  - 主变量可以出现在**SQL**语句中任何一个能够使用表达式的地方
  - 先说明后使用
  - 为了区别于数据库对象（表、视图、列等），在**SQL**语句中，主变量前要加冒号“**:**”作为标志。

- 主变量的说明格式

EXEC SQL BEGIN DECLARE SECTION;

<主变量>

……

EXEC SQL END DECLARE SECTION;

4.面向集合和面向记录的协调

- **SQL**语言是面向集合的，一条**SQL**语句可以产生或处理多条记录；主语言是面向记录的，一组主变量一次只能存放一条记录。
- 使用主变量并不能完全满足**SQL**语句向应用程序输出数据的要求。
- 嵌入式**SQL**引入游标（**cursor**）机制来协调面向集合和面向记录的不同处理方式。
  - 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。
  - 游标包含两部分：**1**）由定义该游标的**SELECT**语句返回的结果集；**2**）指向结果集中某一行的指针。

不用游标的嵌入式SQL

- 如果嵌入式SQL语句的执行结果为单记录，或一次处理多个记录而不是对记录逐个处理，可以不用游标。
- 不用游标的SQL语句的种类
  - 说明性语句
  - 数据定义语句
  - 查询结果为单记录的SELECT语句
  - INSERT语句
  - 非CURRENT形式的UPDATE语句
  - 非CURRENT形式的DELETE语句
  - 授权和回收语句

- 不用游标，用主变量；先说明，后使用。

【例】定义字符型主变量**Sno**和**Sname**。

```
EXEC SQL BEGIN DECLARE SECTION;
    CHAR Sno(6);
    CHAR Sname(8);
        … …
EXEC SQL END DECLARE SECTION;
```

- 所有主变量在使用前需用**DECLARE**语句说明。

【例】根据学号查询学生姓名和选修C2课的成绩

```
EXEC SQL SELECT Sname, Cno, Grade
        INTO :Hsname, :Hcno, :Hgrade
        FROM Student, SC
        WHERE Student.Sno = :Hsno AND Cno='C2'
                AND  Student.sno=SC.sno;
```

- Hsno为输入主变量，在执行该语句前，应先用主语言命令给Hsno赋值
- INTO子句指定将查询结果放入相应的主变量（按顺序对应）

【例】在学生选课表中插入选课成绩。

```
EXEC SQL INSERT INTO SC
    VALUES (:Sno, :Cno, :GRADE);
```

- 在执行该语句前，应先给主变量赋值，然后执行该语句，将选课成绩插入**SC**表。

【例】给出学生的学号和要修改的课程号，修改学生选修记录。

```
EXEC SQL UPDATE SC
  SET Cno=:Hnewcno
    WHERE Sno =:Hsno AND Cno=:Holdcno;
```

- 主变量**Holdcno**中为修改前的课程号，**Hnewcno**中为修改后的课程号。
- 非**CURRENT**形式的**UPDATE**语句可以操作多条记录。

【例】给出学号和课程号，删除学生选课记录。

```
EXEC SQL DELETE
        FROM SC
        WHERE Sno=:Hsno AND Cno=:Hcno;
```

- 非CURRENT形式的DELETE语句可以操作多个元组

用游标的嵌入式SQL

- 主语言一次只能处理一条记录，当处理的结果为多条记录时，需借助游标机制，把对结果集的操作转化为对单个记录的处理。
- 必须使用游标的SQL语句
  - 查询结果为多条记录的SELECT语句
  - CURRENT形式的UPDATE语句
  - CURRENT形式的DELETE语句

- 使用游标的步骤
  1、定义游标
  2、打开游标
  3、推动游标
  4、关闭游标

1、定义游标

- 使用**DECLARE**语句
- 语句格式

  EXEC SQL DECLARE <游标名> CURSOR
  FOR <SELECT 语句>
  [FOR UPDATE OF <列名>];

- 功能
  - 说明性语句
  - 用于建立游标和**SELECT**语句之间的关系，并不执行**SELECT**语句
  - **FOR UPDATE OF**子句表示更新查询结果中的列

2、打开游标

- 使用**OPEN**语句
- 语句格式

  EXEC SQL OPEN <游标名>;

- 功能
  - 执行与游标关联的**SQL**查询语句，并将查询结果放入数据缓冲区。
  - 此时游标处于打开状态，游标指针指向查询结果集中第一条记录之前的位置。

3、推动游标

- 使用**FETCH**语句
- 语句格式

  EXEC SQL FETCH <游标名>
  INTO <主变量> [,<主变量>]…;
- 功能
  - 把游标向前推进一个位置，然后按照游标的当前位置取一条记录（元组），将元组的值赋给对应的主变量。
  - 每执行一次该语句，游标向前推进一个位置。
  - 如果最后游标指针所指的位置为空，则**SQLCODE**返回一个状态码，表示缓冲区没有可处理的元组。

4、关闭游标

- 使用**CLOSE**语句
- 语句格式

  EXEC SQL CLOSE <游标名>;
- 功能
  - 关闭游标，释放结果集占用的缓冲区及其他资源
- 说明
  - 游标被关闭后，就不再和原来的查询结果集相联系
  - 被关闭的游标可以再次被打开，与新的查询结果相联系

游标使用示例

使用游标查询学生选课记录。

```
EXEC SQL DECLARE C1 CURSOR FOR
SELECT Sno,Cno,Grade FROM SC;
EXEC SQL OPEN C1;
while(1){
  EXEC SQL FETCH C1
       INTO :Hsno,:Hcno,:Hgrade;
  if(sqlca.sqlcode <>0) break;
  printf("sno:%s,cno:%s,grade:%d",
              Hsno, Hcno, Hgrade);
     … …
};
EXEC SQL CLOSE C1;
```

游标的扩充

- 为进一步方便用户处理数据，现在一些关系数据库管理系统对**FETCH**语句做了扩充，允许用户向任意方向以任意步长移动游标指针。
- **SQL2**提供了滚动游标（**Scroll Cursor**）
  - **EXEC SQL DECLARE** <游标名> **SCROLL CURSOR FOR** <**SELECT** 语句>
    [**FOR UPDATE OF** <列名>];
  - **EXEC SQL FETCH [NEXT | PRIOR | FRIST | LAST | ABSOLUTE n | RELATIVE n]** <游标名>
    INTO <主变量> [,<主变量>]…;

CURRENT形式的UPDATE和DELETE语句

- **CURRENT形式的更新语句格式：**

  EXEC SQL UPDATE <表名>

  SET=<表达式>

  WHERE CURRENT OF <游标名>;

  - **CURRENT OF <游标名>表示当前游标所指的记录**
  - 要求游标的定义带有**FOR UPDATE**子句

- **CURRENT形式的删除语句格式为：**

  EXEC SQL DELETE FROM <表名>

  WHERE CURRENT OF <游标名>;

  - 要求使用带**FOR UPDATE**的**DECLARE**语句定义游标

嵌入式SQL应用示例

4.7.4 嵌入式 SQL 应用实例

下面通过一个 C 语言的例子说明嵌入式 SQL 的编程框架和嵌入式 SQL 语句的使用。在这个例子中，使用游标查询并显示学生选修课的成绩，如果成绩在 85 分以上，显示成绩为“优”，在 75～84 之间，成绩为“良”，在 60～74 之间，成绩为“及格”，在 60 分以下，成绩为“不及格”。

下面给出实现如上功能的较完整的 C 语言程序。

```
# include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;   /*定义主变量*/
CHAR uid(20);
CHAR pwd(20);
CHAR hsno(6);
CHAR hcno(6);
INT hgrade;
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;          /*定义 SQL 通讯区*/
main ()
{cahr g (6);
strcpy (uid, 'SA');
strcopy (pwd, 'CRT');
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd     /*建立与 DBMS 的连接*/
Scanf ("%s", hcno);                          /*输入要查询的课程号*/
EXEC SQL DECLARE C1 CURSOR FOR
SELECT Sno, Grade
FROM SC
WHERE Cno=:hcno ;                /* (定义游标)*/
EXEC SQL OPEN C1;                            /* (打开游标)*/
while (1)
{EXEC SQL FETCH C1 INTO :hsno, :hgrade;      /*(推进游标)*/
if (sqlca.sqlcode <>0)
    break;
if (hgrade>=85)
    g = '优';
  else if ((hgrade<85) &&( hgrade>=75))
    g = '良';
  else if ((hgrade<75) &&( hgrade>=60))
    g = '及格';
  else g='不及格';
printf("sno: %s  grade: %s", hsno, g)};
};
EXEC SQL CLOSE C1;                           /*（关闭游标）*/
EXEC SQL COMMIT WORK RELEASE;                /*提交事务，退出 DBMS*/
Exit(0);
}
```

以上程序中用到了事务的概念，事务是 DBMS 为保证数据一致性、

- 【嵌入式SQL示例】
依次检查某个系的学生记录，交互式更新某些学生年龄。

```
EXEC SQL BEGIN DECLARE SECTION;    /*主变量说明开始*/
   char Deptname[20];
   char Hsno[9];
   char Hsname[20];
   char Hssex[2];
   int HSage;
   int NEWAGE;
EXEC SQL END DECLARE SECTION;      /*主变量说明结束*/
EXEC SQL INCLUDE SQLCA;            /*定义SQL通信区*/
```

```
int main(void)                          /*C语言主程序开始*/
{
        int  count = 0;
        char  yn;                                   /*变量yn代表yes或no*/
        printf("Please choose department name(CS/MA/IS): ");
        scanf("%s",Deptname);                   /*为主变量deptname赋值*/
        EXEC SQL CONNECT TO TEST@localhost:54321 USER
                    "SYSTEM"/"MANAGER";         /*连接数据库TEST*/
        EXEC SQL DECLARE SX CURSOR FOR    /*定义游标SX*/
                SELECT Sno,Sname,Ssex,Sage    /*SX对应的语句*/
                   FROM Student
                   WHERE SDept = :Deptname;
        EXEC SQL OPEN SX;     /*打开游标SX，指向查询结果的第一行*/
```

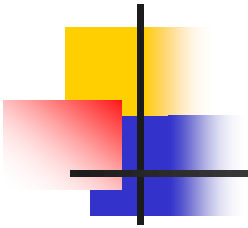
```
for ( ; ; )      /*用循环结构逐条处理结果集中的记录*/
{
    EXEC SQL FETCH SX INTO :HSno,:Hsname,:HSsex,:HSage;
        /*推进游标，将当前数据放入主变量*/
    if (SQLCA.SQLCODE!= 0)  /*SQLCODE != 0，表示操作不成功*/
       break;          /*利用SQLCA中的状态信息决定何时退出循环*/
    if(count++ == 0)           /*如果是第一行的话，先打出行头*/
       printf("\n%-10s %-20s %-10s %-10s\n",
                    "Sno","Sname","Ssex", "Sage");
    printf("%-10s %-20s %-10s %-10d\n",
                HSno,Hsname,Hssex,HSage);       /*打印查询结果*/
    printf("UPDATE AGE(y/n)?");   /*询问是否更新该学生的年龄*/
    do{
       scanf("%c",&yn);
    }while(yn != 'N' && yn != 'n' && yn != 'Y' && yn != 'y');
```

```
    if (yn == 'y' || yn == 'Y')              /*如果选择更新操作*/
    {
       printf("INPUT NEW AGE:");
       scanf("%d",&NEWAGE);      /*用户输入新年龄到主变量中*/
       EXEC SQL UPDATE Student          /*嵌入式SQL更新语句*/
                        SET Sage = :NEWAGE
                        WHERE CURRENT OF SX;
    }                          /*对当前游标指向的学生年龄进行更新*/
  }
  EXEC SQL CLOSE SX;              /*关闭游标SX，不再和查询结果对应*/
  EXEC SQL COMMIT WORK;                                  /*提交更新*/
  EXEC SQL DISCONNECT TEST;                             /*断开数据库连接*/
} /*C语言主程序结束*/
```

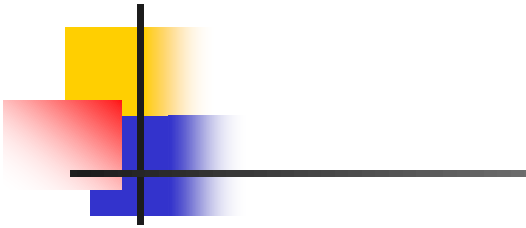
动态SQL

- 静态SQL
  - 数据库对象、查询条件以及所做的操作，在预编译时都是确定的。
  - 语句中主变量的个数与数据类型在预编译时也是确定的。
  - 用户可以在程序运行过程中根据实际需要输入**WHERE**子句或**HAVING**子句中某些变量的值。
  - 只是主变量的值可以在程序运行过程中动态输入。
  - 静态**SQL**语句提供的编程灵活性在许多情况下仍显得不足，不能编写更为通用的程序。

- 动态SQL
  - 在程序运行期间临时组装SQL语句
- 动态SQL的应用范围
  - 在预编译时下列信息不能确定时
    - SQL语句正文
    - 主变量个数
    - 主变量的数据类型
    - SQL语句中引用的数据库对象（列、索引、基本表、视图等）



- 动态SQL的形式
  - 语句可变
    - 临时构造完整的SQL语句
  - 条件可变
    - WHERE子句中的条件
    - HAVING短语中的条件
  - 数据库对象、查询条件均可变
    - SELECT子句中的列名
    - FROM子句中的表名或视图名
    - WHERE子句中的条件
    - HAVING短语中的条件



- 动态SQL语句的参数化
  - 使用主变量
  - 动态参数
    - SQL语句中的可变元素；
    - 使用参数符号（?）表示该位置的数据在运行时设定
    - 输入不是在编译时完成绑定，而是通过 PREPARE语句准备主变量和执行语句EXECUTE绑定数据或主变量来完成 。

- 动态SQL的执行方式
  - 即席输入后直接执行
    EXEC SQL EXECUTE IMMEDIATE <主变量1>;
  - 经过预处理后多次执行
    预处理语句格式:
    EXEC SQL PREPARE <语句名> FROM <主变量1>;
    执行语句格式:
    EXEC SQL EXECUTE <语句名>
    [USING <主变量2>];
  - 主变量1中存放可执行的SQL语句
  - 主变量2给出与语句名关联的SQL语句的参数

【直接执行】

```
String dstr="DELETE FROM SC WHERE Sno='S01';";
const char *stmt="CREATE TABLE test(a int);";
EXEC SQL EXECUTE IMMEDIATE :dstr;
EXEC SQL EXECUTE IMMEDIATE :stmt;
```

【预处理执行】

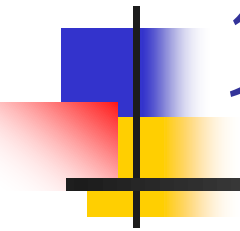
```
strcpy (dstr, "UPDATE SC SET Grade=:g WHERE
   Cno='C005';");
EXEC SQL PREPARE S1 FROM :dstr;
EXEC SQL EXECUTE S1 USING :HGrade;
```

【动态参数】

```
EXEC SQL BEGIN DECLARE SECTION;
   const char *stmt = "INSERT INTO test VALUES(?);";
EXEC SQL END DECLARE SECTION;
  ...
EXEC SQL PREPARE mystmt FROM :stmt;
     /*准备语句*/
EXEC SQL EXECUTE mystmt USING 100;
     /*执行语句，设定INSERT语句插入值100 */
EXEC SQL EXECUTE mystmt USING 200;
     /* 执行语句，设定INSERT语句插入值200 */
```

嵌入式SQL小结

- 在嵌入式SQL中，SQL语句与主语言语句分工非常明确
  - SQL语句直接与数据库打交道；
  - 主语言语句，控制程序流程，对SQL语句的执行结果做进一步加工处理。
- SQL语句用主变量从主语言中接收执行参数，操纵数据库。
- SQL语句的执行状态由DBMS送至SQLCA中，主语言程序从SQLCA中取出状态信息，据此决定下一步操作。
- 如果SQL语句从数据库中成功地检索出数据，则通过主变量传给主语言做进一步处理。
- SQL语言和主语言的不同数据处理方式通过游标来协调。



第4章小结

第4章小结

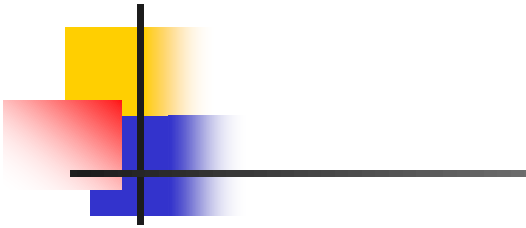
- SQL的特点
  - 综合统一
  - 高度非过程化
  - 面向集合的操作方式
  - 同一种语法结构提供两种使用方式
  - 语言简捷，易学易用
- 交互式SQL的功能
  - 数据定义（CREATE, DROP, ALTER）
  - 数据查询（SELECT）
  - 数据更新（INSERT, UPDATE, DELETE）
  - 数据控制（GRANT，REVOKE）

- 数据查询
  - 对表的选取（**FROM**）
  - 对列的选取（**SELECT**）
    - 目标表达式、更改列标题
  - 对行的选取（**WHERE**）
    - 消除重复行、**WHERE**子句条件表达式
  - 排序（**ORDER BY**）
  - 聚集函数
    - **MIN, MAX, AVG, SUM,COUNT …**
  - 分组
    - **GROUP BY(GROUP BY…HAVING)**

- **WHERE子句的条件表达式**
  - 比较操作符
    = > < >= <= <>
  - 确定范围
    BETWEEN … AND
  - 集合查找
    IN
  - 字符串匹配(匹配符：_ %)
  - 空值运算
    IS NULL,
  - 多重条件组合
    逻辑表达式，逻辑运算符

- 单表查询
- 多表连接查询
  - 条件连接
  - 等值连接
  - 自然连接
  - 外连接
  - 复合条件连接
  - 嵌套查询（子查询）
    - 不相关子查询
    - 相关子查询
- 集合查询

- 数据更新
  - 插入数据
    - 插入单条记录
    - 插入多条记录
  - 修改数据
    - 更新单表数据
    - 带子查询的数据修改
  - 删除数据
    - 删除单表数据
    - 带子查询的数据删除



- 视图
  - 视图的作用
  - 创建视图
  - 使用视图
    - 查询
    - 更新（有条件的）
  - 视图查询的实现
    - 实体化
    - 视图消解

- 嵌入式SQL
  - 与主语言的通信方式
    - SQL通信区（SQLCA）
    - 主变量
    - 游标
  - 静态SQL
    - 不用游标
    - 使用游标
  - 动态SQL

第4章作业

- P92(e97)
  习题6和8
- 作为实验报告的部分内容
- 融入实验报告提交

  - 上机实验内容
    - 实验1：关系数据库系统环境和数据库的建立
    - 实验2：标准SQL语言的基本操作
    - 实验3：复杂查询、触发器和存储过程
    - 实验4：数据备份与恢复，数据库权限管理

第4章思考题

- SQL如何实现基本的关系代数操作
- 教材P92(e97)
  习题1、2、4、7、9
- 调研SQL注入问题
- 开始思考数据库应用系统开发（含嵌入式SQL）
- 注意：不同版本SQL和不同产品SQL的差异