第三章 关系数据库标准语言SQL

- 概述
- SQL数据查询功能
- SQL数据定义功能
- SQL视图操作
- SQL数据更新
- 空值的处理
- SQL数据控制
- 嵌入式SQL

SQL概述

- SQL的产生与发展
  - 1974年，由Chamberlin和Boyce提出，称为SEQUEL（Structured English Query Language）；
  - 1975-1979年，在IBM的San Jose研究室研制的System R上实现;
  - 1981年， IBM在推出SQL/DS关系数据库时，将其命名为SQL (Structured Query Language);
  - 随着SQL语言应用的日益广泛，ANSI和ISO先后制定了SQL-86、 SQL-89、 SQL-92、 SQL-99、SQL-2003、SQL-2008、SQL-2011等多个SQL标准。

SQL特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式
- 以同一种语法结构提供两种使用方式
- 语言简捷，易学易用

SQL功能	操作符
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

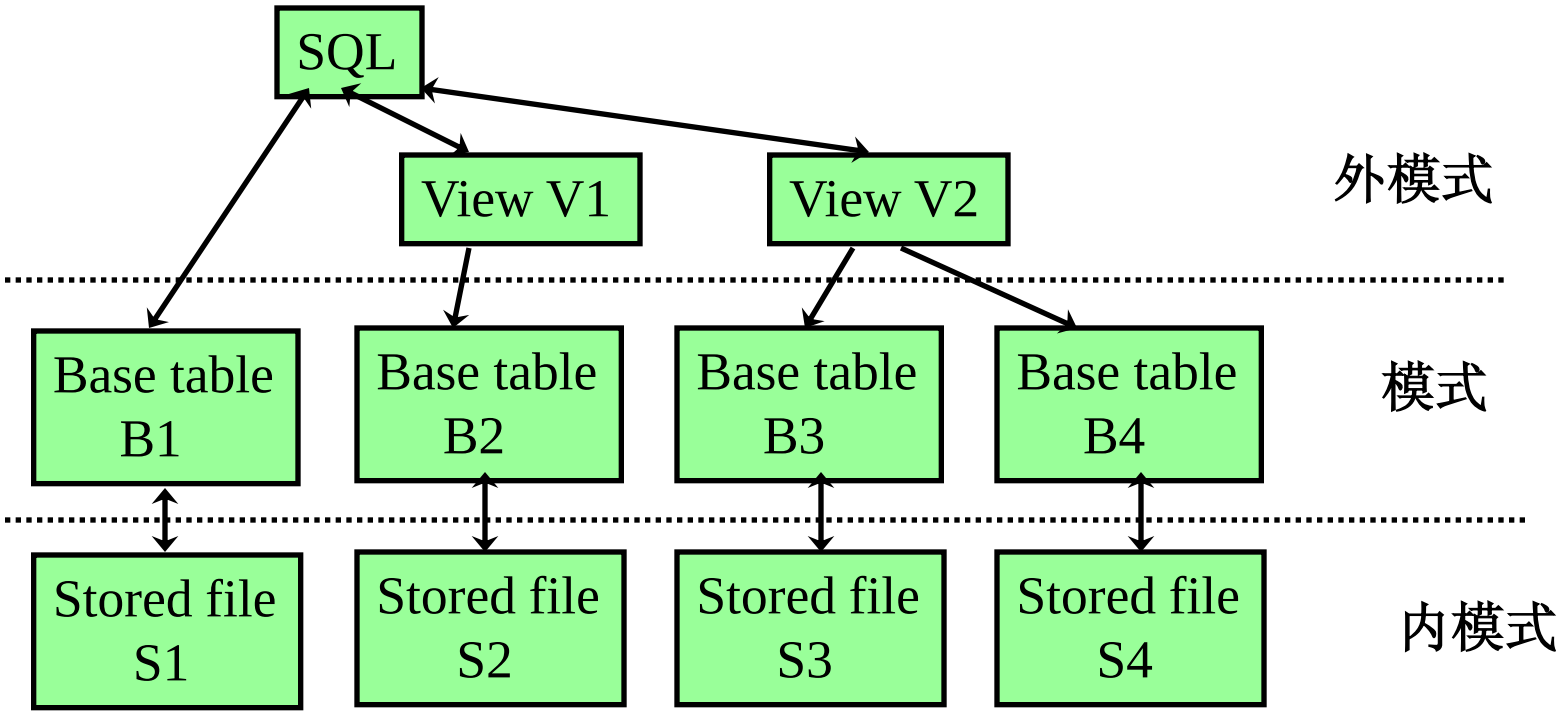
SQL语言的基本概念

- 基本表与导出表
  - 基本表：是实际存在的，每个表在存储中可用一个存储文件来表示；
  - 导出表:是从基本表导出的表，有视图（View）和快照(Snapshot)
    - 视图是一个虚表。即视图所对应的数据不实际存储在数据库中，只在数据库的数据字典中存储视图的定义；
    - 视图一经定义就可以和基本表一样进行查询等操纵，也可以用来定义新的视图
    - 视图定义示例：

```
Create View CS_Student
 As
        SELECT S#, SN, SA FROM S
        WHERE   SD = 'CS'
```

SQL语言的基本概念

- 关系数据库的三级模式结构



SQL数据查询功能

- 查询的基本结构是 SELECT-FROM-WHERE组成的查询块。一般形式：

SELECT 目标列

FROM 基本表（或视图）

WHERE 检索条件

要检索的数据项

要操作的关系名，1个或多个

查询结果应满足的条件表达式

  - 查询块的结果仍是一个表；
  - 查询块执行的过程是在表的水平方向上按“检索条件”选取元组，又在垂直方向上按SELECT指定的列进行投影；
  - 查询块可进行关系代数中投影、选取、连接等操作的组合。

示例用表

学生表 S

S#	SN	SA	SD
s1	李勇	20	CS
s2	刘晨	19	IS
s3	王敏	18	MA
s4	张立	19	IS

学生选课表SC

S#	C#	G
s1	C1	92
s1	C2	85
s1	C3	88
s2	C2	90
s2	C3	80

课程表C

C#	CN	PC#
C1	数据库	C5
C2	数学	
C3	信息系统	C1
C4	操作系统	C6
C5	数据结构	C7
C6	数据处理	
C7	Pascal语言	C6

投影检索

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- SELECT-FROM-WHERE查询块中，没有WHERE子句，是单纯的投影操作。

例1：检索学生的姓名，年龄

```
SELECT SN，SA
FROM S ；
```

- 采用DISTINCT消去SELECT结果中的重复行。

例2：检索学生所选修课程的课程号

```
SELECT DISTINCT C#
FROM SC ；
```

选取检索

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 由WHERE子句指出查询条件。
- 检索条件可以包括如下运算符：
  - 比较运算符：=,< >(!=),> , >=, < , <=
  - 布尔运算符：AND，OR，NOT
  - ( )

例1：检索选修C2课程的所有学生的学号和成绩。

```
SELECT  S#，G
FROM SC
WHERE C#='C2'；
```

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例2：检索选修C1或C2且成绩高于70分的学生学号、课程号和成绩。

```
SELECT  S#，C#，G
FROM SC
WHERE （C#='C1' OR C#='C2'）AND G>=70；
```

例3：检索成绩在70至85分之间的学生学号、课程号和成绩。

```
SELECT  S#，C#，G
FROM SC
WHERE G BETWEEN 70 AND 85；
```

排序检索

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 在SELECT-FROM-WHERE查询块后接ORDER BY子句，将结果按指定列排序。

  格式：ORDER BY 列名 ASC 或DESC

  – ASC为升序；DESC为降序，缺省为升序
  – 可以是单列排序或多列排序
  – 该子句在SELECT语句中作为最后一个子句出现

例1：检索全体学生信息，并按系号升序，同一个系按年龄降序排列。

```
SELECT *
FROM S
ORDER BY SD , SA DESC;
```

连表检索（1）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 将多个相互关联的表按照一定条件连接起来，实现多表数据检索。
- SELECT-FROM-WHERE语句块结构：

  SELECT—— 指明选取的列名（来自多个表）

  FROM—— 指明要进行连接的表名

  WHERE——指明连接条件（连接谓词）与选取条件。

  - 连接条件一般格式为：

[<表名>.]<列名><比较运算符>[<表名>.]<列名>

其中，<列名>称为连接字段；<比较运算符>主要有：=，>，<，>=，<=，!=

连表检索（2）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索学生张华所学课程的成绩。

```
SELECT SN，C#，G
FROM  S，SC
WHERE S.S#=SC.S# AND SN='张华'；
```

连表条件　　选取条件

注：如果连接的表中有属性名相同，要用表名作前缀加以区分。

连表检索（3）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 表自身的连接
  - 通过定义别名，将一个表看成两个表，进行连接。

例2：检索所有比李勇年龄大的学生姓名、年龄。

```
SELECT X.SN，X.SA
FROM S X，S Y
WHERE X.SA>Y.SA AND Y.SN='李勇'；
```

连表检索（4）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 外连接
  - 在连接谓词某一边加（*或+），则逻辑上为*所在边的表增加了一个空行。它可以与另一个表中所有不满足连接条件的元组进行连接，使这些元组能够输出。

例3：检索所有学生的全部信息。

```
SELECT *
FROM S，SC
WHERE S.S#=SC.S#(*);
```

学生表S

S#	SN	SA	SD
s1	李勇	20	CS
s2	刘晨	19	IS
s3	王敏	18	MA
s4	张立	19	IS

学生选课表SC

S#	C#	G
s1	C1	92
s1	C2	85
s1	C3	88
s2	C2	90
s2	C3	80

子查询嵌套检索（1）

- WHERE子句中可以包含另一个查询块，该查询块称为子查询或嵌套查询，包含子查询的语句称为外部查询
- 外部查询利用子查询来获取检索条件的条件值，检索条件根据子查询的结果来确定外部查询的结果数据
- 子查询按照与外部查询的联系不同，分为普通子查询和相关子查询
  - 普通子查询：与外部查询无关，可单独执行得一组值。
  - 相关子查询：把外查询的列值作为检索条件的条件值。

子查询嵌套检索（2）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 涉及同一个表的子查询

例1：检索与李勇同岁的学生姓名。

```
SELECT  SN
FROM  S
WHERE S.SA=
        (SELECT SA FROM S
          WHERE SN='李勇' );
```

- 如果子查询返回单值，可以直接用比较运算符=,<>,>,>=,<,<=等连接子查询。
- 如果子查询返回一组值，则必须在比较运算符和子查询之间插入ANY、ALL等操作符。

子查询嵌套检索（3）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例2：检索选修C2课程的学生姓名。

```
SELECT  SN
FROM  S
WHERE S#=ANY
        (SELECT S# FROM SC
          WHERE C#='C2');
```

例3：检索选修C2课程的成绩最高的学生学号。

```
SELECT  S#
FROM  SC
WHERE  C#='C2' AND
       G>=ALL
         (SELECT G  FROM SC
          WHERE C#='C2');
```

子查询嵌套检索（4）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用IN检索
  - IN在嵌套子查询中最常使用。可代替“=ANY”，是集合运算中的“∈”运算。

例4：检索选修C2课程的学生姓名。

```
SELECT  SN
FROM  S
WHERE S# IN
        (SELECT S# FROM SC
         WHERE C#='C2');
```

- 用NOT IN检索
  - NOT IN表示不在集合中，与！=ALL相同。

子查询嵌套检索（5）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用EXISTS检索
  - EXISTS表示存在量词“∃”。
  - 表达式 EXISTS（子查询）
    当且仅当子查询结果为非空时为真。

例5：检索选修C2课程的学生姓名。

```
SELECT  SN
FROM  S
WHERE EXISTS
        (SELECT * FROM SC
         WHERE  S#=S.S#  AND C#='C2');
```

- 用NOT EXISTS 检索
  - 表示“不存在”。
  - 表达式NOT EXISTS（子查询）在子查询结果为空时为真。

子查询嵌套检索（6）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用NOT EXISTS表达全称量词（∀）
  - 任何一个带有全称量词的谓词总可以转换为等价的带存在量词的谓词：
  - (∀x)P ≡¬ (∃ x (¬ P))

例6：检索选修所有课程的学生姓名。

本题等价于“检索这样的学生的姓名，不存在他不选修的课程”。

```
SELECT  SN
FROM  S
WHERE NOT EXISTS
      (SELECT * FROM C
       WHERE  NOT EXISTS
             (SELECT *  FROM SC
              WHERE  S#=S.S#  AND C#=C.C#));
```

s.s# 不选修的课程 (不存在s.s#选修记录的课程）

S.S#选修C.C#的记录

子查询嵌套检索（7）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 用NOT EXISTS 表达蕴涵
  - $p \rightarrow q \equiv \neg p \vee q$

$(\forall y)\ p \rightarrow q \equiv \neg (\exists y(\neg (p \rightarrow q))) \equiv \neg (\exists y (\neg (\neg p \vee q))) \equiv \neg \exists y (p \wedge \neg q)$

例7：检索至少选修了学生S2选修的全部课程的学生学号。

本题用蕴含表达：

“检索这样学生的学号，对于任意课程，只要S2选修，他就选修。”

本题等价于“检索这样学生的学号，不存在S2选修而他没有选修的课程”。

```
SELECT  DISTINCT S#
FROM  SC  SCX
WHERE NOT EXISTS
    (SELECT * FROM SC   SCY
     WHERE  SCY.S#='S2' AND NOT EXISTS
          (SELECT *  FROM SC SCZ
           WHERE  S#=SCX.S#  AND C#=SCY.C#));
```

S2选修而SCX.S#没有选修的课程（S2选修的课程并且不存在SCX.S#选修该课程的记录）

SCX.S#选修SCY.C#的记录

并、差、交检索（1）

- 并、差、交的SQL运算符：
  - 并：UNION
  - 差：MINUS
  - 交：INTERSECT
- 并、差、交检索的操作对象必须是相容的，是同类关系，即必须有相同数量的属性列，且相应属性列的域也必须相同。

并、差、交检索（2）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索选修了C1或C2课程的学生学号。

```
SELECT  S#     FROM  SC   WHERE C#='C1'
UNION
SELECT S#   FROM SC   WHERE  C#='C2';
```

例2：检索无人选修的课程号和名称。

```
SELECT  C#，CN   FROM  C   WHERE C# IN
        (SELECT C#   FROM C
          MINUS
          SELECT  DISTINCT C#   FROM  SC)；
```

库函数检索（1）

- 库（集）函数
  - COUNT( ) 按列值计个数，COUNT(*)对行记数。
  - SUM() 对数值列求总和
  - AVG() 求数值列的平均值
  - MAX() 在列中找出最大值
  - MIN() 在列中找出最小值
- 只能在SELECT子句以及HAVING子句中出现

库函数检索（2）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索学生总数。

```
SELECT  COUNT(*)    FROM  S ;
```

例2：检索选修了课程的学生人数。

```
SELECT  COUNT(DISTINCT S#)  FROM  SC  ;
```

例3：求学号为S4的学生的总分和平均分。

```
SELECT  SUM(G)，AVG(G)
FROM  SC
WHERE  S#='S4' ;
```

例4：检索选修了C1课程的学生最高分。

```
SELECT  MAX(G)  FROM  SC  WHERE C#='C1';
```

分组检索（1）

- 按属性列（列组）将关系的元组分组，每组在这些分组属性列（列组）上具有相同值，对每一组执行SELECT操作。
- 分组子句：

  GROUP BY 列名

  [HAVING 条件表达式] ——分组条件

- WHERE子句与HAVING子句
  - WHERE 子句是针对“行”进行，用于去掉不符合条件的若干行；HAVING子句针对“分组”进行，必须和GROUP BY 连用，用于去掉不符合条件的若干分组。
  - 在查询块中出现的顺序：WHERE — GROUP BY — HAVING

分组检索（2）

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

例1：检索至少选修三门课程的学生学号和选课门数。

```
SELECT  S#，COUNT（*）
FROM  SC
GROUP  BY S#
HAVING COUNT（*）>=3；
```

例2：求选修四门以上课程的学生学号和总成绩（不统计不及格的课程）。最后按降序列出总成绩排序名单。

```
SELECT  S#，SUM（G）
FROM  SC
WHERE G>=60
GROUP  BY S#
HAVING COUNT(*) >=4
ORDER BY  SUM(G)  DESC;
```

算术表达式值的检索

- SELECT子句中，可包括由属性列、常数、库函数、算术运算符+-*/等组成的算术表达式。
- 检索结果数据项名可用表达式表示或用“别名”来表示。

例1：有职工表EMP(EMP#，EMPN，JOB，SALARY，BONUS，DEPT)，要求检索所有PROGRAMMER的奖金大于工资25%的职工姓名和一年的总收入，并按奖金与工资之比的降序排列。

```
SELECT  EMPN，BONUS/SALARY   BS ，
                12*（SALARY+BONUS）  TOTAL
FROM EMP
WHERE JOB=‘PROGRAMMER’
          AND BONUS>0.25*SALARY
ORDER   BY  BONUS/SALARY   DESC ;
```

部分匹配查询

- 使用谓词LIKE 或NOT LIKE，一般形式：

  <列名> LIKE/NOT LIKE <字符串常量>

  – “列名”必须为字符型或变长字符型。

  – “字符串常量”可包含两个特殊符号 % 与_

    - %：代表任意序列的0个或多个字符；
    - _ ：代表任意单个字符

例1：检索所有姓刘的学生的学号、姓名。

```
SELECT S#，SN
FROM S
WHERE SN LIKE ‘刘%’;
```

基于派生表的查询

- 当子查询出现在FROM子句中，则子查询生成的表称为临时派生表，该表也可作为主查询的操作对象

例1：检索每个学生超出自己平均成绩的课程号。

```
SELECT  S#，C#
FROM  SC, (SELECT S# , AVG(G)
                FROM SC
                GROUP BY S#)  AS AVG_SC(AVG_S#,AVG_G)
WHERE SC.S# = AVG_SC. AVG_S# AND SC.G >= AVG_SC. AVG_G)
```

- AS 关键字可以省略，但必须为派生表指定别名
- 如果子查询中没有库函数，则派生表可不指定列

SQL数据定义功能

- 定义、删除、修改基本表
- 定义、删除视图
- 定义、删除索引

操作对象	操作方式		
	创建	删除	修改
表	Create Table	Drop Table	Alter Table
视图	Create View	Drop View	
索引	Create Index	Drop Index	

定义基本表

- 定义基本表

  Create Table <表名>

  (<列名><数据类型>[<列级完整性约束>]

  [{,<列名><数据类型>[<列级完整性约束>]}]

  [{, [<表级完整性约束>]}]);

  – 完整性约束
    - **NULL/NOT NULL**
    - **UNIQUE**
    - **PRIMARY KEY**
    - **FOREIGN KEY**
    - **CHECK**

- SQL92的数据类型
  - 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，q）：定点数共p位，小数点右边q位。
  - Real, double precision：浮点数与双精度浮点数,精度与机器有关。
  - Float(n)：n位的精度浮点数。
  - 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

示例

- Create table S
  (S# char(5) not null unique,
  SN char(20) not null,
  SA int,
  SD char(15),
  Primary key(S#),
  Check (SA >=18 and SA <=45))；
- Create table SC
  (S# char(5) not null,
  C# char(5) not null,
  G number(4,2),
  Primary key (S#, C#),
  Foreign key (S#) references S (S#),
  Foreign key (C#) references C (C#));

S(S#,SN,SA,SD);

C(C#,CN,PC#);

SC(S#,C#,G)

修改、删除基本表定义

- 修改基本表
  - 语法：Alter Table <表名>
    [Add <新列名><数据类型>[<完整性约束>]]
    [Drop <完整性约束名>]
    [Modify <列名><数据类型>]
  - 示例
    - 例1：S表增加“入学时间”属性
      Alter Table S Add Scome Date ;
    - 例2：将SA的数据类型改为半字长整数
      Alter Table S Modify SA Smallint
- 删除基本表
  - 语法: Drop Table <表名>
  - 示例: Drop Table S;

定义、删除索引

- 定义索引
  - 语法：Create [Unique][Cluster] Index <索引名>
    On <表名> (<列名>[次序][, <列名>[次序]] …);
  - 示例： Create Unique Index Scno
    On SC(S# ASC, C# DESC)
- 删除索引
  - 语法：Drop Index <索引名>
  - 示例：Drop Index Stusno;

定义视图

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 定义视图
  - 语法：
    Create View <视图名>
    [(<列名>[,<列名>] …)]
    As <子查询>
    [With Check Option]
  - 示例：建立计算机系的学生视图

```
Create View CS_Student
As
SELECT S#, SN, SA FROM S
WHERE   SD = 'CS' ;
```

- 删除视图

  Drop View <视图名>

查询视图

- 视图消解（View Resolution）

  DBMS执行对视图的查询时，从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正的查询。这一转换过程称为视图消解。

视图查询示例

- 在计算机系学生的视图中找出年龄小于20岁的学生。

```
Create View CS_Student
                As
                SELECT S#, SN, SA FROM S
                WHERE   SD = 'CS'

  Select S#, SA From CS_Student
    Where SA < 20;
```

- 转换后为：

```
  Select S#, SA From S
    Where SD='CS' And SA <20;
```

视图的作用

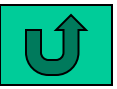
- 能够简化用户操作
  - 例：

```
Create View CS_Student
        As
        SELECT S#, SN, SA FROM S
        WHERE  SD = 'CS'
```

```
Select S#, SA From CS_Student
 Where SA < 20;

Select S#, SA From S
 Where SD='CS' And SA <20;
```

- 使用户能够以多种角度看待同一数据
- 提供了一定程度的逻辑独立性
- 能够对数据提供安全保护



SQL数据更新

- 插入数据——Insert语句
- 修改数据——Update语句
- 删除数据——Delete语句

插入数据

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 插入单个元组

  Insert Into <表名>[(<属性列>[{,<属性列>}])]
  Values(<值>[{,<值>}])

  示例：Insert Into S Values ('S10', '陈冬', 'IS', 18);

- 插入子查询结果

  Insert Into <表名>[(<属性列>[{,<属性列>}])]
  <子查询>

  示例：Insert Into Dept_Age (Sdept, Avgage)
  Select SD, AVG(SA) From S
  Group By SD;

修改数据

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 语法：Update <表名>

  Set <列名>=<表达式>[{, <列名>=<表达式>}]

  [Where <条件>]

- 示例
  - 例1：将学生S1的年龄改为22岁

```
Update S
Set  SA = 22
Where S#='S1';
```

删除数据

S(S#,SN,SA,SD);
C(C#,CN,PC#);
SC(S#,C#,G)

- 语法： Delete From <表名> [Where <条件>]
- 示例：
  - 例1：删除学号为S19的学生的记录

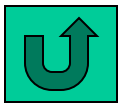
```
Delete From S Where S#='S19'
```

  - 例2：删除所有学生的选课记录

```
Delete From SC
```

  - 例3：删除计算机系所有学生的选课记录

```
Delete From SC
Where 'CS' = (Select SD  From S
                  Where S .S# = SC.S#)
```



空值的运算

- 空值是“不知道”或“不存在”或“无意义”的值，用NULL表示，例：UPDATE S SET SD = NULL
- 属性定义有NOT NULL或UNIQUE约束以及主属性，都不可取空值
- 空值与另一个值的算术运算结果为NULL，与另一个值的比较运算结果为UNKNOWN
- (TRUE, FALSE, UNKNOWN) 的三值逻辑
  - 空值的AND、OR、NOT运算的真值表

逻辑表达式	值
T AND U, U AND U	U
F AND U	F
F OR U, U OR U	U
T OR U	T
NOT U	U

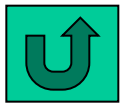
空值的检索

- 空值的判断
  - 判断属性是否为空值，用IS NULL或IS NOT NULL来表示

例：
```
SELECT *
FROM S
WHERE SD IS NULL;
```

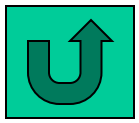
- 在检索操作中，只有使WHERE和HAVING子句中的条件为TRUE的元组才被输出

例：检索C01课程成绩不及格的学生学号
```
SELECT S#
FROM SC
WHERE C#='C01' AND G <60;
```
上述查询中，缺考的学生将不被输出。



SQL数据控制功能

- 定义完整性约束条件
- 支持事务操作
- 提供安全控制功能
  - 授权
    - GRANT 〈权限〉[ON 〈对象类型〉 〈对象名〉]
      TO 〈用户〉
  - 收回权限
    - REVOKE 〈权限〉[ON 〈对象类型〉 〈对象名〉]
      FROM 〈用户〉



嵌入式SQL

- 嵌入式SQL的意义
  - 嵌入式SQL把SQL语句嵌入到高级语言中
  - 嵌入式SQL把SQL的最佳特性与程序设计语言的最佳特性(如过程处理能力）结合起来，使SQL功能更强，灵活性更强

嵌入式SQL程序示例

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
int var_c1;
char var_c2[21] = { 0x00 };
EXEC SQL END DECLARE SECTION;

int main(int argc, char* argv[])
{
    /* 步骤1: 建立连接 */
    EXEC SQL WHENEVER SQLERROR SQLPRINT;    /* 声明异常发生时的处理动作统一为打印消息 */
    EXEC SQL CONNECT TO postgres@localhost:5432 USER postgres/xxxx;    /* 指定连接的目标数据库，用户名，密码 */

    /* 步骤2: 利用游标执行查询 */
    EXEC SQL DECLARE foo_bar CURSOR FOR SELECT c1, c2 FROM tb1;    /* 声明一个游标使之用于执行SELECT文 */
    EXEC SQL OPEN foo_bar;                                        /* 打开游标从而使SELECT文被执行 */
    EXEC SQL FETCH foo_bar INTO :var_c1, :var_c2;                 /* 获取一行数据 */

    printf("C1: %d, C2: %s\n", var_c1, var_c2);

    EXEC SQL CLOSE foo_bar;                                       /* 关闭所打开的游标 */

    /* 步骤3: 关闭连接 */
    EXEC SQL DISCONNECT CURRENT;

    return 0;
}
```

在SQL语句前加前缀 EXEC SQL，区分SQL语句与主语言语句

通过游标协调SQL（一次一集合）与主语言（一次一记录）之间操作方式

通过主变量（主语言程序变量）交换数据

连接PostgreSQL数据库，宿主语言是C

嵌入式SQL的处理

- 对于嵌入式SQL，DBMS多采用预编译方法进行处理。
  - 把嵌入在程序中的SQL语句翻译为高级语言（主语言）源码，然后按主语言的通常方式进行编译、连接形成可执行代码。
- 动态SQL允许在程序运行过程中“临时”组装SQL语句

小结

- 概念
  - 基本表与视图；关系数据库的三级模式结构
  - SQL的特点
- SQL的数据定义、查询、更新、控制功能
- 视图的作用及SQL的视图操作