数据库系统概论

An Introduction to Database System

第三章 关系数据库标准语言 SQL(4)

第三章 关系数据库标准语言SQL

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.1 插入数据

- 三种插入数据方式

  1) 插入单个元组

  2) 插入子查询结果

  3) 一次性插入多条记录

1. 插入单个元组

- 语句格式

INSERT

INTO <表名> [(<属性列1>[，<属性列2 >…)]

VALUES (<常量1> [，<常量2>]　…　　)

- 功能

将新元组插入指定表中。

插入单个元组（续）

- **INTO子句**
  - 指定要插入数据的表名及属性列(属性列的顺序可与表定义中的顺序不一致)
  - 没有指定属性列：表示要插入的是一条完整的元组，且属性列属性与表定义中的顺序一致
  - 指定部分属性列：插入的元组在其余属性列上取空值
- **VALUES子句**
  - 提供的值必须与**INTO**子句匹配
    - > 值的个数
    - > 值的类型

插入子查询结果（续）

DBMS在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则

- 实体完整性
- 参照完整性
- 用户定义的完整性
  - 对于有NOT NULL约束的属性列是否提供了非空值
  - 对于有UNIQUE约束的属性列是否提供了非重复值
  - 对于有值域约束的属性列所提供的属性值是否在值域范围内

插入单个元组（续）

[例1] 将一个新学生记录（学号：200215128；姓名：陈冬；性别：男；所在系：IS；年龄：18岁）插入到Student表中。

```
INSERT
INTO Student(sno,sname,ssex,sdept,sage)
VALUES (‘200215128'，'陈冬'，'男'，'IS'，18)
```

插入单个元组（续）

[例2] 将学生张成民的信息插入到Student表中

```
INSERT
INTO Student
VALUES (‘200215126’,‘张成民 ‘,'男 ‘, 18 ,‘CS');
```

插入单个元组（续）

[例3] 插入一条选课记录(‘200215128'，'1 ')。

```
INSERT
INTO SC(Sno，Cno)
VALUES (' 200215128 '，' 1 ');
```

新插入的记录在Grade列上取空值

2. 插入子查询结果

- 语句格式

  INSERT

  INTO <表名> [(<属性列1> [，<属性列2>…)]

  子查询；

- 功能:

  将子查询结果插入指定表中

插入子查询结果（续）

[例4] 对每一个系，求学生的平均年龄，并把结果存入数据库。

第一步：建表

```
CREATE  TABLE  Deptage
   (Sdept  CHAR(15) ,        /* 系名*/
   Avgage SMALLINT);    /*学生平均年龄*/
```

插入子查询结果（续）

第二步：插入数据

```
INSERT
INTO  Deptage(Sdept，Avgage)
SELECT  Sdept，AVG(Sage)
FROM  Student
GROUP BY Sdept；
```

3.一次性插入多条记录

- 语句格式
- **INSERT**
  INTO table_name(col_one,col_two)
  SELECT 'col1','col2'
  UNION
  SELECT 'col11','col22'

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.2 修改数据

- 语句格式

  UPDATE <表名>

  SET <列名>=<表达式>[，<列名>=<表达式>]…

  [WHERE <条件>];

- 功能

  修改指定表中满足**WHERE**子句条件的元组

修改数据（续）

- 三种修改方式

1) 修改某一个元组的值

2) 修改多个元组的值

3) 带子查询的修改语句

修改数据（续）

- **SET子句**

  指定修改方式

  要修改的列

  修改后取值

- **WHERE子句**

  指定要修改的元组

  缺省表示要修改表中的所有元组

修改数据（续）

DBMS在执行修改语句时会检查修改操作是否破坏表上已定义的完整性规则

- 实体完整性
- 主码不允许修改
- 用户定义的完整性
  - NOT NULL约束
  - UNIQUE约束
  - 值域约束

1. 修改某一个元组的值

[例5] 将学生200215121的年龄改为22岁。

```
UPDATE  Student
SET Sage=22
WHERE  Sno=' 200215121 ';
```

2. 修改多个元组的值

[例6] 将所有学生的年龄增加1岁。

```
UPDATE Student
SET Sage= Sage+1；
```

修改多个元组的值(续)

[例] 将信息系所有学生的年龄增加1岁。

```
UPDATE Student
SET Sage= Sage+1
WHERE Sdept=' IS ';
```

3. 带子查询的修改语句

[例7] 将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET  Grade=0
WHERE  'CS'=
    (Select  distinct Sdept
     FROM  Student
     WHERE  Student.Sno = SC.Sno);
```

3.5 数 据 更 新

3.5.1 插入数据

3.5.2 修改数据

3.5.3 删除数据

3.5.3 删除数据

DELETE
FROM <表名>
[WHERE <条件>];

- 功能
  - 删除指定表中满足WHERE子句条件的元组
- WHERE子句
  - 指定要删除的元组
  - 缺省表示要修改表中的所有元组

删除数据（续）

- 三种删除方式
  - 删除某一个元组的值
  - 删除多个元组的值
  - 带子查询的删除语句

1. 删除某一个元组的值

[例8] 删除学号为200215128的学生记录。

```
DELETE
FROM S
WHERE Sno='200215125';
```

2. 删除多个元组的值

[例9] 删除所有的学生选课记录。

```
DELETE
FROM SC;
```

[例] 删除2号课程的所有选课记录。

```
DELETE
FROM SC
WHERE Cno='2';
```

3. 带子查询的删除语句

[例10] 删除计算机科学系所有学生的选课记录。

```
DELETE
FROM SC
WHERE  'CS'=
    (SELET Sdept
     FROM Student
     WHERE Student.Sno=SC.Sno)；
```

删除数据(续)

DBMS在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则

– 参照完整性
  - 不允许删除
  - 级联删除

更新数据与数据一致性

DBMS在执行插入、删除、更新语句时必须保证数据库一致性

- 必须有事务的概念和原子性
- 完整性检查和保证

第三章 关系数据库标准语言SQL

3.6 视　图

视图的特点

- 虚表，是从一个或几个基本表（或视图）导出的表
- 只存放视图的定义，不会出现数据冗余
- 基表中的数据发生变化，从视图中查询出的数据也随之改变

3.6 视　图

基于视图的操作

- 查询
- 删除
- 受限更新
- 定义基于该视图的新视图

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

1. 建立视图

- 语句格式

CREATE VIEW

<视图名> [(<列名> [，<列名>]…)]

AS <子查询>

[WITH CHECK OPTION];

几点说明：

1. **DBMS执行CREATE VIEW语句时只是把视图的定义存入数据字典，并不执行其中的SELECT语句。在对视图查询时，按视图的定义从基本表中将数据查出。**

2. WITH CHECK OPTION　透过视图进行UPDATE, INSERT和DELETE操作时，不得破坏视图定义中的谓词条件（即子查询中的条件表达式）

4. 组成视图的属性列名全部省略或全部指定

– 可以省略:

由子查询中SELECT目标列中的诸字段组成

– 必须明确指定视图的所有列名:

(1) 某个目标列是集函数或列表达式(只要指定列名称即“别名”也可）。

(2) 多表连接时选出了几个同名列作为视图的字段

(3) 需要在视图中为某个列启用新的更合适的名字

5. 只能在当前的数据库中创建视图（被引用的表可以存在于其他的数据库中。

- 不能将规则、默认值绑定在视图上
- 定义视图的查询语句中不能包括**ORDER BY**子句或包括**INTO**关键字。

6. **EXEC SP_HELPTEXT** 查看视图定义信息。

[例1] 建立信息系学生的视图。

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

从单个基本表导出只是去掉了基本表的某些行和某些列，保留了主码，称这类视图为行列子集视图。如IS_Student视图就是一个行列子集视图。

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

之后对该视图进行插入、修改和删除操作时，自动加上Sdept= 'IS'条件。

基于多个基表的视图

[例3] 建立计算机系选修了1号课程的学生视图。

```
CREATE VIEW IS_S1(Sno,Sname,Grade)
        AS
        SELECT S.Sno,Sname,Grade
        FROM  S,SC
        WHERE  Sdept= 'cS' AND
               S.Sno=SC.Sno AND
               SC.Cno= '1'
```


```
CREATE VIEW IS_S2 (a,b,c,d)
AS
SELECT s.sno,sc.sno,Sname,Grade
   FROM  S,SC
WHERE  Sdept= 'cS' AND
            S.Sno=SC.Sno AND
            SC.Cno= '1'
```

基于视图的视图

[例4] 建立信息系选修了1号课程且成绩在90分以上的学生的视图。

```
CREATE VIEW IS_S2
    AS
    SELECT Sno，Sname，Grade
    FROM  IS_S1
    WHERE  Grade>=90；
```

带表达式的视图

[例5] 定义一个反映学生出生年份的视图。

```
CREATE  VIEW BT_S(Sno，Sname，Sbirth)
AS
SELECT Sno，Sname，2019-Sage
FROM  Student
```

设置一些派生属性列，也称为虚拟列--Sbirth 带表达式的视图必须明确定义组成视图的各个属性列名

建立分组视图

[例6] 将学生的学号及他的平均成绩定义为一个视图

假设SC表中“成绩”列Grade为数值型

```
CREAT  VIEW S_G(Sno，Gavg)
AS
  SELECT Sno，AVG(Grade)
   FROM  SC
   GROUP BY Sno；
```

常见的视图形式

- 行列子集视图
- **WITH CHECK OPTION**的视图
- 基于多个基表的视图
- 基于视图的视图
- 带表达式的视图
- 分组视图

2. 删除视图

- **DROP VIEW <视图名>;**
  - 该语句从数据字典中删除指定的视图定义
  - 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除
  - 删除基表时，由该基表导出的所有视图定义都必须显式删除

删除视图(续）

[例8] 删除视图IS_S1

```
DROP VIEW IS_S1；
```

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

3.5.2 查询视图

- 从用户角度：查询视图与查询基本表相同
- DBMS实现视图查询的方法
  - 视图实体化法（View Materialization）
    - 有效性检查：检查所查询的视图是否存在
    - 执行视图定义，将视图临时实体化，生成临时表
    - 查询视图转换为查询临时表
    - 查询完毕删除被实体化的视图(临时表)

查询视图（续）

– 视图消解法（View Resolution）

- 进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义
- 把视图定义中的子查询与用户的查询结合起来，转换成等价的对基本表的查询
- 执行修正后的查询

查询视图（续）

[例9] 在信息系学生的视图中找出年龄小于20岁的学生。

```
SELECT   Sno，Sage
FROM     IS_Student
WHERE    Sage<20；
```

IS_Student视图的定义 (视图定义例1)：

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

查询视图（续）

– 视图实体化法

– 视图消解法

转换后的查询语句为：

```
SELECT  Sno，Sage
FROM  Student
WHERE  Sdept= 'IS'  AND  Sage<20；
```

查询视图（续）

[例10] 查询信息系选修了1号课程的学生

```
SELECT  Sno，Sname
FROM    IS_Student，SC
WHERE  IS_Student.Sno =SC.Sno AND
          SC.Cno= '1'；
```

查询视图（续）

- 视图消解法的局限
  - 有些情况下，视图消解法不能生成正确查询。采用视图消解法的DBMS会限制这类查询。

查询视图（续）

[例11]在S_G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   S_G
WHERE  Gavg>=90;
```

S_G视图定义：

```
CREATE VIEW S_G (Sno，Gavg)
AS
   SELECT  Sno，AVG(Grade)
   FROM  SC
   GROUP BY Sno;
```

查询转换

错误：

```
SELECT Sno，AVG(Grade)
FROM    SC
WHERE  AVG(Grade)>=90
GROUP BY Sno；
```

正确：

```
SELECT  Sno，AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90；
```

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

3.6.3 更新视图

- 用户角度：更新视图与更新基本表相同
- **DBMS实现视图更新的方法**
  - 视图实体化法（View Materialization）
  - 视图消解法（View Resolution）
- **指定WITH CHECK OPTION子句后**

  DBMS在更新视图时会进行检查，防止用户通过视图对不属于视图范围内的基本表数据进行更新

[例12] 将信息系学生视图IS_Student中学号200215122的学生姓名改为“刘辰”。

UPDATE IS_Student

SET Sname= '刘辰'

WHERE Sno= ‘200215122';

转换后的语句：

UPDATE Student

SET Sname= '刘辰'

WHERE Sno= ‘200215122' AND Sdept= 'IS';

[例13] 向信息系学生视图IS_Student中插入一个新的学生记录：（200215129，赵新，20岁）

```
INSERT
INTO IS_Student
VALUES('200215129', '赵新', 20);
```

转换为对基本表的更新：

```
INSERT
INTO Student(Sno，Sname，Sage，Sdept)
VALUES('200215129'，'赵新'，20，'IS' );
```

更新视图（续）

[例14] 删除视图IS_Student中学号为200215129的记录

```
DELETE
FROM IS_Student
WHERE Sno= ‘200215129';
```

转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno= ‘200215129' AND Sdept= 'IS';
```

更新视图的限制

- 在关系数据库中，并不是所有的视图都是可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

例：视图S_G为不可更新视图。

```
CREATE VIEW S_G (Sno，Gavg)
 AS
SELECT  Sno，AVG(Grade)
FROM    SC
GROUP BY Sno；
```

更新视图（续）

如果想把学号为200215121的学生的平均成绩改成90分，语句如下：

```
UPDATE  S_G
SET     Gavg=90
WHERE  Sno= '200215121';
```

无论实体化法还是消解法都无法将其转换成对基本表SC的更新（系统无法修改各科成绩，以使平均成绩为90）

3.6 视　图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用

1. 视图能够简化用户的操作

当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作

- 基于多张表连接形成的视图
- 基于复杂嵌套查询的视图
- 含导出属性的视图

2. 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应数据库共享的需要

3.视图对重构数据库提供了一定程度的逻辑独立性

例：数据库逻辑结构发生改变

学生关系Student(Sno，Sname，Ssex，Sage，Sdept)

“垂直”地分成两个基本表：

SX(Sno，Sname，Sage)

SY(Sno，Ssex，Sdept)

3.视图对重构数据库提供了一定程度的逻辑独立性

通过建立一个视图**Student**：

```
CREATE VIEW  Student(Sno，Sname，Ssex，Sage，Sdept)
AS
SELECT  SX.Sno，SX.Sname，SY.Ssex，SX.Sage，
   SY.Sdept
FROM  SX，SY
WHERE  SX.Sno=SY.Sno；
```

使用户的外模式保持不变，从而对原**Student**表的查询程序不必修改

3. 视图对重构数据库提供了一定程度的逻辑独立性

- 物理独立性与逻辑独立性的概念
- 视图在一定程度上保证了数据的逻辑独立性
- 视图只能在一定程度上提供数据的逻辑独立性
  - 由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

4. 视图能够对机密数据提供安全保护

- 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据
- 通过**WITH CHECK OPTION**对关键数据定义操作限制

5.适当地利用视图可以更清晰地表达查询

- 例：对每个同学找出他获得最高成绩的课程号
- **Create view VMGrade**
- **As**
- **Select Sno, Max(Grade) as 'Mgrade'**
- **From SC**
- **Group by Sno;**
- 再用如下的查询语句完成查询：
- **Select SC.Sno, Cno**
- **From SC, VMGrade**
- **Where SC.Sno = VMGrade.Sno and SC.Grade = VMGrade.Mgrade;**
- *实验思考题：如果不使用视图怎样进行查询？*