# 数据库原理

# The Theory of Database System

## 第三章 关系数据库标准语言SQL

中国矿业大学计算机学院

# 本讲主要内容

数据更新

数据控制

视图

# 数据更新

1 插入数据

2 修改数据

3 删除数据

# 1 插入数据

- 两种插入数据方式
  - 插入单个元组
  - 插入子查询结果

# 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）

[例]将一个新学生记录（学号：091530；姓名：夏雨；性别：男；籍贯：海南；出生年份：1999；学院：计算机；）插入学生表中。

```
INSERT
INTO 学生
VALUES ('091530', '夏雨', '男','海南', '1999',
 '计算机');
```

# 插入单个元组（续）

[例]插入一条选课记录(学号：091530，课程号：080102)。

```
INSERT
INTO 学习(学号, 课程号)
VALUES ('091530', '080102');
```

新插入的记录在成绩列上取空值

# 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] 统计每门课程的平均分，并把结果存入数据库。

第一步：建表

```
CREATE TABLE 课程平均分
    (课程号 CHAR(8),
     课程名 CHAR(15),
     平均分 DOUBLE );
```

# 插入子查询结果（续）

第二步：插入数据

```
INSERT
INTO  课程平均分(课程号, 课程名, 平均分)
    SELECT  课程号, 课程名, AVG(成绩)
    FROM  课程, 学习
    WHERE 课程.课程号=学习.课程号
    GROUP BY 课程号,课程名;
```

# 数据更新

1 插入数据

2 修改数据

3 删除数据

# 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
  - 修改某一个元组的值
  - 修改多个元组的值
  - 带子查询的修改语句

# 修改某一个元组的值

[例]将091611号学生的籍贯改为江苏。

```
UPDATE  学生
SET 籍贯='江苏'
WHERE 学号='091611';
```

# 修改多个元组的值

[例]将选修了180101号课程的学生成绩增加一分。

```
UPDATE   学习
SET  成绩=成绩+1
WHERE  课程号='180101';
```

# 修改多个元组的值(续)

[例]将计算机学院学生的成绩清零。

```
UPDATE 学习
SET   成绩=0
WHERE  学号 IN
( SELECT 学号
   FROM     学生
   WHERE 学院='计算机' );
```

# 带子查询的修改语句

[例]将计算机学院学生的成绩清零。

```
UPDATE 学习
SET  成绩=0
WHERE  EXISTS
   ( SELECT  *
     FROM   学生
     WHERE 学院='计算机'
         AND 学生.学号=学习.学号);
```

# 数据更新

1 插入数据

2 修改数据

3 删除数据

# 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
  - 删除某些元组
  - 删除全部元组
  - 带子查询的删除

# 删除某些元组

[例] 删除学号为’092010’的学生记录。

```
DELETE
FROM   学生
WHERE   学号='092010';
```

[例]删除130101号课程的所有选课记录。

```
DELETE
FROM   学习
WHERE   课程号='130101';
```

# 删除全部元组

【例】清空学习表中的内容。

```
DELETE
FROM  学习;
```

# 带子查询的删除

[例] 删除计算机学院所有学生的选课记录。

```
DELETE
FROM 学习
WHERE  学号 IN
   ( SELECT 学号
      FROM   学生
      WHERE 学院='计算机' );
```

# 带子查询的删除

[例] 删除计算机学院所有学生的选课记录。

```
DELETE
FROM 学习
WHERE   EXISTS
  ( SELECT 学号
    FROM   学生
    WHERE 学院='计算机'
      AND 学生.学号=学习.学号);
```

# 数据更新

DBMS在执行更新语句时会检查所做操作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

# 数据控制

数据控制亦称为数据保护，包括数据的：

- 安全性控制
- 完整性控制
- 并发控制
- 故障恢复

# 存取控制

- 概念：

通过规定不同用户对于不同数据对象所允许执行的操作，从而限制用户只能存取他有权存取的数据。

- 存取控制是保证数据安全性的主要措施。

# DBMS实现存取控制的过程

- 用户或DBA把授权决定告知系统

  SQL的GRANT和REVOKE

- DBMS把授权的内容存入数据字典
- 当用户提出操作请求时，DBMS根据授权定义进行检查，以决定是否执行操作请求

# 存取控制(续)

1 授权

2 回收权限

# 1、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];

- GRANT功能：将对指定操作对象的指定操作权限授予指定的用户。

# WITH GRANT OPTION子句

- 指定了WITH GRANT OPTION子句：

  获得某种权限的用户还可以把这种权限再授予别的用户。

- 没有指定WITH GRANT OPTION子句：

  获得某种权限的用户只能使用该权限，不能传播该权限

# 操作权限

| 对象 | 对象类型 | 操 作 权 限 |
| --- | --- | --- |
| 属性列 | TABLE | SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES |
| 视图 | TABLE | SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES |
| 基本表 | TABLE | SELECT, INSERT, UPDATE, DELETE ALTER, INDEX, ALL PRIVILEGES |
| 数据库 | DATABASE | CREATETAB |

# 1、授权

【例】把学生表的查询权限授予用户User1。

```
GRANT   SELECT
ON   TABLE   学生
TO   User1;
```

# 1、授权

【例】把查询学习表和修改成绩的权限授给用户User2。

```
GRANT UPDATE(成绩), SELECT
ON TABLE 学习
TO User2;
```

# 1、授权

【例】DBA把在数据库SMD中建立表的权限授予用户User3。

```
GRANT CREATE
ON DATABASE SMD
TO User3;
```

# 1、授权

【例】把学生表和课程表的全部权限授予用户User4和User5。

```
GRANT ALL
ON TABLE 学生, 课程
TO User4, User5;
```

# 1、授权

【例】把学习表的查询权限授予全部用户。

```
GRANT SELECT
ON TABLE 学习
TO PUBLIC;
```

# 1、授权

【例】把学生表的INSERT权限授予User6用户，并允许他再将此权限授予其他用户。

```
GRANT INSERT
ON TABLE 学生
TO User6
WITH GRANT OPTION;
```

# 权限的传播

如果User6用户执行了这句：

GRANT INSERT ON TABLE 学生 TO User7
WITH GRANT OPTION;

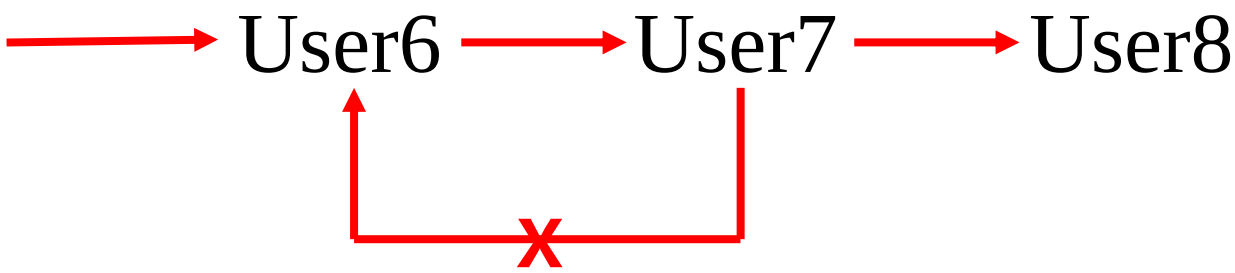
这样，User7不仅获得了相同的权限，还可以将此权限授予User8：

GRANT INSERT ON TABLE 学生 TO User8;

但User8不能再传播此权限。

# 权限的传播

不允许循环授权！



# 存取控制(续)

1 授权

2 回收权限

# 2、回收

- REVOKE语句的一般格式为：

  REVOKE <权限>[,<权限>]...

  [ON <对象类型> <对象名>]

  FROM <用户>[,<用户>]...;

- 功能：从指定用户那里收回对指定对象的指定权限

# 2、回收

【例】把用户User2修改成绩的权限收回。

```
REVOKE UPDATE(成绩)
ON TABLE 学习
 FROM User2;
```

# 2、回收

【例】把用户User6对学生表的INSERT权限收回。

```
REVOKE INSERT
ON TABLE 学生
FROM User6;
```

# 权限的级联回收

- 之前：

  -->User6--> User7--> User8

- 当回收User6对学生表的INSERT权限时，系统将收回直接或间接从User6处获得的对学生表的INSERT权限。

  <--User6<-- User7<-- User8

# 视图

视图的定义

视图的查询

视图的更新

# 视 图

- 概念：

  视图是从一个或几个基本表（或视图）导出的表。

- 视图也称为虚表或者虚关系：只存放视图的定义。

- 视图是一种外模式，从一定程度上提高了数据的安全性。

# 定义视图

- 语句格式

CREATE VIEW

<视图名> [(<列名> [，<列名>]...)]

AS <子查询>

[WITH CHECK OPTION];

- DBMS执行CREATE VIEW语句时只是把视图的定义存入数据字典，并不执行其中的子查询语句。

# 常见的视图形式

- 行列子集视图
- WITH CHECK OPTION的视图
- 基于多个基本表的视图
- 基于视图的视图
- 带有表达式的视图
- 带有分组的视图

# 行列子集视图

- 从单个基本表导出
- 只是去掉了基本表的某些行和某些列
- 保留了码

【例】建立计算机学院学生的视图。

```
CREATE VIEW CS_Stu
    AS
    SELECT 学号，姓名，籍贯
    FROM   学生
    WHERE  学院= '计算机';
```

# WITH CHECK OPTION的视图

- WITH CHECK OPTION

透过视图进行数据更新时，不得破坏视图定义中子查询的条件。

# WITH CHECK OPTION的视图

【例】建立计算机学院学生的视图，并要求透过该视图进行的更新操作只涉及计算机学院的学生。

```
CREATE VIEW CS_Stu
    AS
    SELECT 学号，姓名，籍贯
    FROM    学生
    WHERE  学院= '计算机'
WITH CHECK OPTION;
```

# WITH CHECK OPTION的视图

对CS_Stu视图进行更新时：

- 修改操作：DBMS自动加上学院= '计算机'的条件
- 删除操作：DBMS自动加上学院= '计算机'的条件
- 插入操作：DBMS自动检查学院属性值是否为'计算机'
  - 如果不是，则拒绝该插入操作
  - 如果没有提供学院值，则学院列自动填充'计算机'

# 基于多个基表的视图

【例】建立计算机学院选修了《数据库原理》课程的学生成绩视图。

```
CREATE VIEW CS_DB(学号，姓名，成绩)
  AS
  SELECT 学生.学号，姓名，成绩
  FROM  学生，学习，课程
  WHERE  学生.学号=学习.学号
      AND  学习.课程号=课程.课程号
      AND  学院= '计算机'
      AND  课程名= '数据库原理';
```

# 基于视图的视图

【例】建立计算机学院选修《数据库原理》课程且成绩在90分以上的学生的视图。。

```
CREATE VIEW CS_DB_Good
    AS
    SELECT 学号，姓名，成绩
    FROM  CS_DB
    WHERE  成绩>=90；
```

# 带有表达式的视图

【例】建立一个反映学生年龄的视图。

```
CREATE  VIEW Stu_Age(学号，姓名，年龄)
  AS
SELECT 学号，姓名，year(now())-出生年份
FROM  学生;
```

# 带有分组的视图

【例】建立学生平均成绩视图，显示学号、姓名以及平均成绩。

```
CREAT VIEW Stu_Avg(学号，姓名，平均成绩)
    AS
    SELECT 学号，姓名，AVG(成绩)
    FROM  学生，学习
    WHERE 学生.学号=学习.学号
    GROUP BY 学号，姓名；
```

# 说明

- 计算后的值或者统计值，被称为虚拟列；
- 带有虚拟列的视图，定义时要给出组成视图的各个属性列名；
- 为了减少数据冗余，可以将计算后的值或统计值放在视图中。

# 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。

【例】删除视图CS_Stu。

```
DROP VIEW CS_Stu;
```

# 查询视图

- 从用户角度：查询视图与查询基本表相同。
- DBMS实现视图查询的思想：
  - 转成对基本表的查询

# 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】查询计算机学院学生的视图，找出籍贯为江苏的学生。

```
SELECT  学号，姓名
FROM    CS_Stu
WHERE   籍贯='江苏';
```

CS_Stu视图的定义：

```
CREATE VIEW CS_Stu
    AS
    SELECT 学号，姓名，籍贯
    FROM   学生
    WHERE  学院= '计算机';
```

# 查询视图（续）

转换后的查询语句为：

```
SELECT  学号，姓名
FROM    学生
WHERE   籍贯='江苏'
        AND  学院= '计算机';
```

# 查询视图（续）

【例】查询计算机学院选修了数据库原理课程的学生的学号、姓名和成绩。

```
SELECT  学号，姓名，成绩
 FROM     CS_Stu，学习，课程
 WHERE  CS_Stu.学号=学习.学号
     AND  学习.课程号=课程.课程号
     AND  课程名='数据库原理';
```

# 查询视图（续）

【例】查询计算机学院选修了数据库原理课程的学生的学号、姓名和成绩。

```
SELECT  学号，姓名，成绩
FROM    学生，学习，课程
WHERE   学生.学号=学习.学号
    AND  学习.课程号=课程.课程号
    AND  课程名='数据库原理'
    AND  学院='计算机';
```

# 更新视图

- 用户角度：更新视图与更新基本表相同
- DBMS实现视图更新的思想
  - 转化成对基本表的更新
- 指定WITH CHECK OPTION短语后

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

# 更新视图（续）

【例】将计算机学院的学生视图中学号为091503的学生姓名改为“刘辰”。

```
UPDATE  CS_Stu
SET   姓名= '刘辰'
WHERE   学号= '091503';
```

转换后的语句：

```
UPDATE  学生
SET 姓名= '刘辰'
WHERE 学号= '091503'  AND   学院= '计算机';
```

# 更新视图（续）

【例】向计算机学院的学生视图中插入一个新的学生记录（学号：091622，姓名：赵新，籍贯：浙江）

```
INSERT
INTO CS_Stu
VALUES('091622', '赵新', '浙江');
```

转换为对基本表的更新：

```
INSERT
INTO   学生 (学号，姓名，籍贯，学院)
VALUES('091622'，'赵新'，'浙江'，'计算机' );
```

# 更新视图（续）

【例】删除计算机学院的学生视图中学号为091422的学生记录。

```
DELETE
FROM CS_Stu
WHERE 学号= '091422';
```

转换为对基本表的更新：

```
DELETE
FROM 学生
WHERE 学号= '091422' AND 学院= '计算机';
```

# 更新视图（续）

- 一些视图是不可更新的，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】更新学生平均成绩视图，把学号为091527的学生的平均成绩改为90。

```
UPDATE  Stu_Avg
SET     平均成绩=90
WHERE   学号= '091527';
```

# 更新视图（续）

Stu_Avg视图的定义：

```
CREAT VIEW Stu_Avg(学号, 姓名, 平均成绩)
    AS
    SELECT 学号, 姓名, AVG(成绩)
    FROM  学生, 学习
    WHERE 学生.学号=学习.学号
    GROUP BY 学号, 姓名;
```

无法将其转化成对学习表的更新

# 实际系统对视图更新的限制

- 允许对行列子集视图进行更新。
- 对其他类型视图的更新不同系统有不同限制：

（1）若视图中的属性来自属性表达式或常数，则不允许对视图执行INSERT和UPDATE操作，但允许执行DELETE操作。

（2）若视图中的属性来自集函数，则不允许对此视图更新。

# 更新视图（续）

（3）若视图定义中含有GROUP BY子句，则不允许对此视图更新。

（4）若视图定义中含有DISTINCT短语，则不允许对此视图更新。

（5）若视图定义中有嵌套查询，并且嵌套查询的FROM子句涉及导出该视图的基本表，则不允许对此视图更新。

# 更新视图（续）

【例】建立考试成绩在总平均分之上的学生的视图。

```
CREATE VIEW GOOD_Stu
    AS
    SELECT  学号，课程号，成绩
    FROM    学习
    WHERE   成绩>
            (SELECT AVG(成绩)
             FROM   学习);
```

# 更新视图（续）

（6）若视图由两个以上的基本表导出，则不允许对此视图更新。

（7）如果在一个不允许更新的视图上再定义一个视图，这种二次视图也是不允许更新的。

# 本章小结

- 掌握数据定义语句
- 掌握数据查询语句
- 掌握数据更新语句
- 掌握数据控制语句
- 掌握视图的定义，理解视图的查询与更新