SQL数据更新

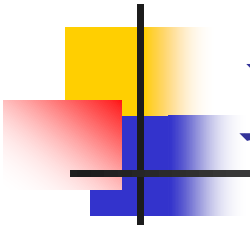
北京理工大学　计算机学院

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
  - 4.4.1数据查询
  - 4.4.2数据更新
- 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

- 数据更新
  - 插入
  - 修改
  - 删除

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。

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

- **INTO**子句
  - 指定要插入数据的表名及属性列
  - 属性列的顺序可以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或者是默认值
- **VALUES**子句
  - 提供的值必须与**INTO**子句匹配：个数、顺序和值的类型

【例】在学生表中插入一个学生元组，其学号为**101215**，姓名为李斌，男，**19**岁，是计算机系的学生。

```
INSERT INTO Student
  VALUES('101215','李斌' , '男' ,19, '计算机');

INSERT INTO Student ( Sno, Sname, Sdept, Sage, Ssex )
  VALUES('101215','李斌' , '计算机' ,19, '男');
```

【例】学号为101253的学生选了C2号课，将其插入选课表中。

```
INSERT INTO SC(Sno, Cno)
  VALUES('101253','C2');

INSERT INTO SC
  VALUES('101253','C2', NULL);
```

如果表的定义说明某列为NOT NULL，
则插入时不能取空值。

空值与默认值

- 将子查询结果插入指定表中

  INSERT INTO <表名>[(<列名1>[,<列名2>, …])]

  <SELECT语句>; /*子查询*/;

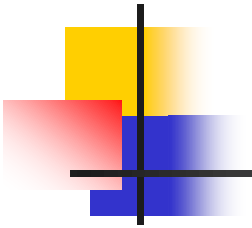
  - **INTO**子句
    - 指定要插入数据的表名及属性列
    - 属性列的顺序可与表定义中的顺序不一致
      - 没有指定属性列，则表示插入一条完整的元组
      - 指定部分属性列，则在其余属性列上取空值
  - 子查询
    - **SELECT**子句目标列必须与**INTO**子句匹配：个数、顺序和值的类型

【例】计算计算机系每个学生的平均成绩，并保存在**CS-AVG**表中。

1. 生成学生的平均成绩表CS-AVG
2. 在CS-AVG中插入计算机系学生的平均成绩

```
CREATE TABLE CS-AVG
      (Sno CHAR(6)NOT NULL，
      Grade NUMBER(4,1));
```

```
INSERT INTO CS-AVG (Sno, Grade)
     SELECT Sno, AVG(Grade)  FROM SC
          WHERE Sno IN (
               SELECT Sno FROM Student
               WHERE Sdept='CS')
          GROUP BY Sno ;
```



- **DBMS**在执行插入语句时会检查所插入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

2.修改数据

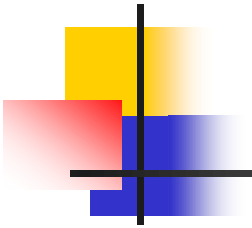
- 语句格式

  UPDATE <表名>

  SET 列名1=<表达式1>[,列名2=<表达式2>]…

  [WHERE <条件表达式>];

- 功能：修改指定表中满足**WHERE**子句条件的元组的指定列的内容
- **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



- **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（主键）不允许修改
  - 用户定义的完整性
    - **NOT NULL**约束
    - **UNIQUE**约束
    - 值域约束

- 修改一个或多个元组的值

【例】将数据库课的学分修改为4。

```
UPDATE Course SET Ccredit =4 WHERE Cname='数据库';
```

【例】将所有学生的年龄增加1岁。

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

【例】将所有选修了数据库课的学生的成绩清空。

```
UPDATE SC SET Grade=NULL
    WHERE Cno IN
        (SELECT Cno
            FROM Course
            WHERE Cname='数据库');
```

【例】将计算机系全体学生的成绩置零。

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

```
Update SC set Grade=0
From SC, Student
where SC.Sno=Student.Sno and Sdept=‘CS’
```

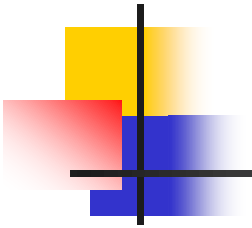
3.删除数据

- 语句格式

  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



- **DBMS在执行删除语句时**
  - 检查所删元组是否破坏表上已定义的完整性规则（参照完整性）；如果破坏，则
    - 不允许删除
    - 级联删除

- 删除一个或多个元组

【例】删除学号为201225的学生记录。

```
DELETE FROM Student WHERE Sno='201225';
```

【例】删除所有的学生选课记录。

```
DELETE  FROM SC;
```

- 带子查询的删除

【例】删除所有选修数据库课学生的选课信息

```
DELETE FROM SC
    WHERE Cno IN
        (SELECT Cno FROM Course
            WHERE Cname='数据库');
```

截断表TRUNCATE TABLE

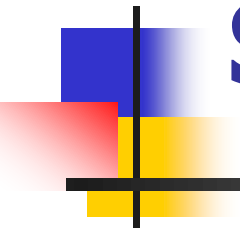
- 语句格式

  TRUNCATE TABLE *table_name*

- 功能
  - 删除表中所有行（与不带 **WHERE** 子句的 **DELETE** 语句相同），但不记录单个行删除操作
  - 比 **DELETE** 速度快，使用的系统和事务日志资源少
    - **DELETE** 语句每次删除一行，并在事务日志中为所删除的每行记录一项。
    - **TRUNCATE TABLE** 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。
  - 操作不能回滚， 但**DELETE** 可以回滚

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的一致性
  - 数据库不一致性的形成
    - 当某同学退学时，首先删除成绩表（子表）中数据，然后删除学籍表（主表）中的数据，但如果执行完第一步后，计算机发生故障，则第二步永远不会执行，就会形成数据库的不一致性。
    - 问题的解决：必须有事务的概念和原子性保障
  - 完整性检查和保证
    - 系统自动在删除主表元组时删除子表对应的元组
    - 系统检查子表中是否有相应的元组，如果存在，禁止删除动作



SQL中的视图

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

- 视图的定义
- 创建视图
- 删除视图
- 视图查询
- 视图更新
- 视图更新的限制
- 视图的优点

视图（View）

- 视图
  - 相对于基本表而言；对应于外模式
  - 是从一个或几个基本表（或视图）导出的虚表
  - 视图的定义是递归的，可以定义基于该视图的新视图
  - **DBMS**只存放视图的定义，不存放视图的数据，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改变
  - 视图实际上提供了一种观察数据的逻辑窗口，用户可从不同的角度观察数据库
  - 对视图的操作意味着对基表进行相对应的操作；但对视图的更新(插入数据、删除、修改)有一些限制

创建视图

- 语句格式

  CREATE VIEW <视图名> [(<列名1> [,<列名2>]...)]

  AS < SELECT语句>

  [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SELECT语句。在对视图进行操作时才按照视图定义生成数据，供用户使用。**
- **SELECT语句表示子查询，视图的属性列和数据都是由该子查询决定的。**

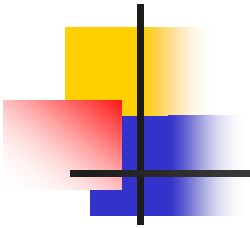
- 选项[(<列名1>[, <列名2>]…)]用来定义视图的列名。
- 组成视图的属性列名可以全部省略或全部指定
  - 省略:

    由SELECT查询结果的目标列名组成
  - 以下情况必须明确指定视图的所有列名:

    (1) 目标列中包含聚集函数或表达式

    (2) 视图中包含出现在多个表中的相同列名

    (3) 需要在视图中为某个列启用新的更合适的名字



- **WITH CHECK OPTION**选项的作用
  - 通过视图插入、删除或修改元组时，检查元组是否满足视图定义中的条件（即子查询中的条件表达式），如果不满足将拒绝执行这些操作。
  - 如果视图定义中含有条件，建议选择**WITH CHECK OPTION**选项，以约束更新的数据。

- 建立与基本表结构相同的视图

【例】建立年龄小于23岁的学生视图，并要求数据更新时进行检查。

```
CREATE VIEW Sage_23
    AS SELECT * FROM Student
        WHERE Sage < 23
        WITH CHECK OPTION;
```

  - 当通过视图更新学生元组时，系统将检查所更新的学生年龄是否小于23岁，不满足条件时系统将拒绝执行更新操作

- 建立带表达式的视图

【例】按系建立学生平均年龄的视图。

```
CREATE VIEW D-Sage (Sdept, Avgage)
    AS   SELECT Sdept, AVG(Sage)
           FROM Student
           GROUP BY Sdept;
```

  - 因在**SELECT**目标表中有聚集函数**AVG**，视图定义中必须含有列名选项。
  - 视图的列名与**SELECT**后的列名相对应，即使有与基本表相同的列名也不能省略。

- 建立基于多个基表的视图

【例】建立计算机系选修了C2课的学生姓名和成绩的视图。

```
CREATE VIEW CS_SC(Sno, Sname, Grade)
  AS SELECT Student.Sno, Sname, Grade
        FROM Student, SC
        WHERE Sdept='计算机' AND
            Student.Sno=SC.Sno AND SC.Cno='C2';
```

- 建立基于视图的视图

【例】建立计算机系选修了C2课且成绩在90分以上的学生视图。

```
CREATE VIEW CS_90
        AS SELECT Sno, Sname, Grade
                FROM CS_SC
                WHERE Grade>=90 ;
```

- 一类不易扩充的视图

  以 **SELECT *** 方式创建的视图可扩充性差，应尽可能避免.

【例】将**Student**中所有女生定义为一个视图。

```
CREATE VIEW F_Std (num, name, age, sex, dept)
      AS  SELECT *
          FROM  Student
          WHERE Ssex='女'
```

```
CREATE VIEW F_Std (num, name, age, sex, dept)
      AS  SELECT Sno, Sname, Sage, Ssex, Sdept
          FROM  Student
          WHERE Ssex='女'
```

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
  - 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。
- 【例】删除学生视图**CS_90**。

```
DROP VIEW CS_90;
```

视图查询

- 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
  - 实体化视图（**View Materialization**）
    1. 有效性检查：检查所查询的视图是否存在
    2. 执行视图定义，将视图临时实体化，生成临时表
    3. 查询视图转换为查询临时表
    4. 查询完毕删除被实体化的视图(临时表)

- 视图消解法（**View Resolution**）
  1. 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；
  2. 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；
  3. 执行修正后的查询。

- 【例】查询计算机系年龄小于23岁的学生。

```
SELECT *
    FROM  Sage_23
    WHERE Sdept='计算机';
```

  - 视图消解法：系统首先从数据字典中取出视图定义，把查询语句与视图定义中的子查询合并在一起，然后在相关基本表上执行查询 。转换后的查询语句为：

```
SELECT *
    FROM Student
    WHERE Sdept='计算机' AND Sage < 23;
```

- 【例】查询专业系，要求学生平均年龄小于**21**岁。

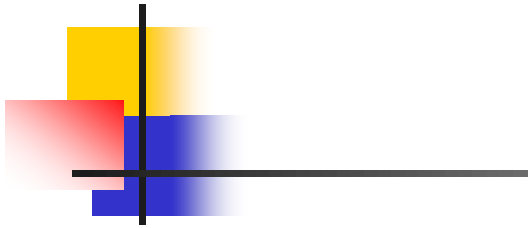
```
CREATE VIEW D-Sage (Sdept, Avgage)
      AS   SELECT Sdept, AVG(Sage)
           FROM Student
           GROUP BY Sdept;
```

```
SELECT Sdept
    FROM D-Sage
    WHERE Avgage<21;
```

```
SELECT Sdept
    FROM Student
    GROUP BY Sdept
    HAVING AVG(Sage)<21;
```

转化为

对基本表的查询



- 视图消解法的局限性
  - 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。
  - 对于简单视图，视图消解是总能进行的。
- 实体化视图的方法
  - 无限制

视图更新

- 对视图的数据插入、删除、修改最终转换为对基表的操作来进行
- 用户角度：更新视图与更新基本表相同
- 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
- 有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】通过视图Sage_23插入学生刘敏的信息（'20041' ,'刘敏' ,21,'女','数学'）。

```
INSERT INTO Sage_23
  VALUES ('20041' ,'刘敏' ,21,'女','数学');
```

以上插入将转换成如下语句执行：

```
INSERT INTO Student
  VALUES ('20041' ,'刘敏' ,21,'女','数学');
```

```
CREATE VIEW Sage_23
      AS SELECT * FROM Student
           WHERE Sage < 23
     WITH CHECK OPTION;
```

- 【例】通过视图Sage_23删除学生王茵的记录。

```
DELETE FROM Sage_23
    WHERE Sname='王茵';
```

该删除语句将转换为对基本表的操作：

```
DELETE FROM Student
    WHERE Sname='王茵' AND Sage < 23;
```

```
CREATE VIEW Sage_23
    AS SELECT * FROM Student
        WHERE Sage < 23
    WITH CHECK OPTION;
```

- 【例】通过视图**Sage_23**修改学生王茵的年龄为**22**岁。

```
UPDATE Sage_23
    SET Sage=22
    WHERE Sname='王茵';
```

该修改转换为对学生表的修改：

```
UPDATE Student
    SET Sage=22
    WHERE Sname='王茵';
```

  - 因修改后学生年龄小于**23**岁，该操作可直接对表**Student**修改。

视图更新的限制

- 【例】通过视图**D-Sage**插入计算机系学生的平均年龄（'计算机'，**21**）。

```
INSERT INTO D-Sage
  VALUES ('计算机',21);
```

以上插入语句无法执行，因为视图**D-Sage**为不可更新视图。

```
CREATE VIEW D-Sage (Sdept, Avgage)
      AS   SELECT Sdept, AVG(Sage)
             FROM Student
             GROUP BY Sdept;
```

- 【例】通过视图CS_SC 删除学生刘明亮的信息。

```
DELETE FROM CS_SC
    WHERE Sname='刘明亮';
```

以上语句无法执行，因为不能转化为对基本表的操作，删除操作的语义不明确。

```
CREATE VIEW CS_SC(Sno, Sname, Grade)
    AS SELECT Student.Sno, Sname, Grade
        FROM Student, SC
        WHERE Sdept='计算机' AND
            Student.Sno=SC.Sno AND SC.Cno='C2';
```

- 视图更新是一个比较复杂的问题，在实际商品化系统中对视图的更新都有限制。
- 有的视图是不可更新的，但也有一些视图是可更新的而实际系统没有实现
  - 不可更新的视图：理论上证明不可更新的视图
  - 不允许更新的视图：实际的数据库系统不支持更新，但其本身从理论上可以更新的视图
- 仅在一个表上取其行列值且其列中包含了候选键，这样所形成的视图都是可更新的，这类视图称为“行列子集视图”。
- 除行列子集视图外的视图的更新都会受到限制。

- **DB2对视图更新的限制：**
  - 由两个以上基本表导出的视图不允许更新。
  - 若视图的字段来自字段表达式或常数，则不允许对此视图执行**INSERT**和**UPDATE**操作，但允许执行**DELETE**操作。
  - 若视图的字段来自聚集函数或含有**GROUP BY**子句、含有**DISTINCT**短语，则此视图不允许更新
  - 若视图定义中有嵌套查询，并且内层查询的**FROM**子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。
  - 一个不允许更新的视图上定义的视图也不允许更新。

视图的作用

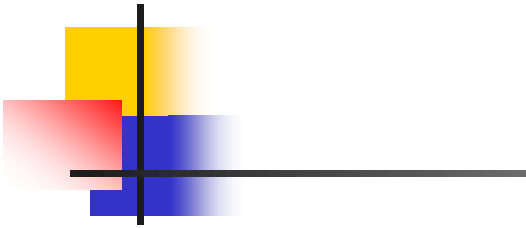
- 视图提供了数据的逻辑独立性
  - 数据库的逻辑结构发生改变时，由于数据库的数据不变，可以通过对视图的重新定义，使用户的外模式保持不变，从而使查询视图数据的应用程序不必修改。
  - 视图在一定程度上保证了数据的逻辑独立性
  - 视图只能在一定程度上提供数据的逻辑独立性
    - 视图更新是有条件的，视图的重新定义可能会影响到数据的更新，此时修改数据的应用程序可能也需要做相应的修改。

【例】学生关系Student(Sno,Sname,Ssex,Sage,Sdept)， 垂直分成两个基本表：SX(Sno,Sname,Sage)，SY(Sno,Ssex,Sdept)

```
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;
```

使用户外模式保持不变，

用户应用程序通过视图仍然能够查找数据。



- 简化了用户视图
  - 视图使用户把注意力集中在自己所关系的数据上，简化了用户的数据结构；
  - 定义视图能够简化用户的操作；适当的利用视图可以更清晰的表达查询
    - 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

- 视图使用户以不同角度看待相同的数
  - 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。
- 视图提供了安全保护功能
  - 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，实现对机密数据的保护。
  - 可以通过**WITH CHECK OPTION**对关键数据定义操作限制，比如操作时间的限制。

【例】通过视图限定1号课程的选课记录的任何操作只能在工作时间进行。

```
CREATE VIEW V_SC AS
    SELECT Sno，Cno，Grade
        FROM  SC
        WHERE  Cno= '1'
           AND  TO_CHAR(SYSDATE,'HH24')
                  BETWEEN  9  AND  17
          AND  TO_CHAR(SYSDATE,'D')
                  BETWEEN  2  AND  6
WITH CHECK OPTION;
```

SQL的数据控制

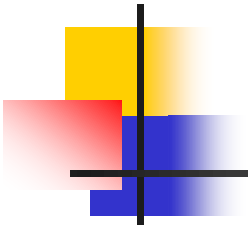
北京理工大学　计算机学院
张文耀
zhwenyao@bit.edu.cn

第4章 关系数据库标准语言SQL

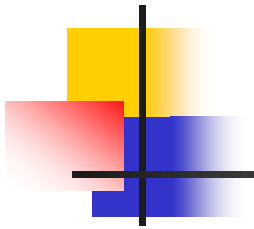
- 4.1SQL简介
- 4.2SQL的系统结构
- 4.3SQL的数据定义
- 4.4SQL的数据操纵
- 4.5SQL中的视图
- 4.6SQL的数据控制
  - 4.6.1 授权
  - 4.6.2 权限回收
- 4.7嵌入式SQL
- 4.8小结

SQL的数据控制

- SQL的数据控制功能包括
  - 数据的安全性 ——第6章
  - 数据的完整性 ——第7章
  - 并发控制 ——第9章
  - 数据恢复 ——第8章
- SQL对数据的安全保护采取了许多措施
  - 视图机制
  - 权限控制
    通过权限控制用户对数据的操作，只有拥有相应的权限，才能实现相应的操作
  - …



- **SQL中的权限**
  使用SQL语句存取数据的能力
- 不同的数据对象有不同的操作权限
  - 数据库
    - **CREATE TABLE**
  - 表
    - **SELECT, INSERT, DELETE, UPDATE**
    - **ALTER, INDEX**
  - 属性列
    - **SELECT, INSERT, DELETE, UPDATE**
  - 视图
    - **SELECT, INSERT, DELETE, UPDATE**



- **SQL**权限由谁定义？
  DBA和表的建立者（即表的主人）
- **SQL**权限如何定义？
  SQL语句：
  - 授权语句**GRANT**
  - 回收语句**REVOKE**
- 表或视图的建立者（**Owner**）拥有所创建表或视图上的所有权限，包括授予权。

授权

- 授权：是指有授予权的用户将自己所拥有的权限授予其他用户。
- 语句格式：

```
GRANT <权限1>[,<权限2>]...
    [ON <对象类型> <对象名>]
    TO <用户1>[,<用户2>... |PUBLIC]
    [WITH GRANT OPTION];
```

- 功能：把指定对象的某些权限授予指定的用户。
- **WITH GRANT OPTION**短语表示被授权的用户还可以把获得的权限再授予其它用户。
- **PUBLIC**短语表示把权限授予数据库的所有用户。

【例】授予用户**User2**在表**Student**上的查询权和删除权，同时使**User2**拥有将所得权限授予其他用户的权力。

```
GRANT SELECT, DELETE
  ON TABLE Student
  TO User2
  WITH GRANT OPTION;
```

【例】授予用户**User4**对表**SC**中的列**Grade**的修改权。

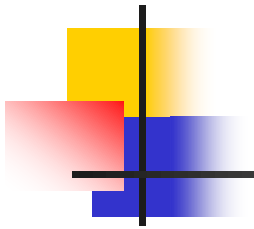
```
GRANT UPDATE(Grade)
  ON TABLE SC
  TO User4;
```

【例】把对表SC的查询权限授予所有用户

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

【例】把对Student表和Course表的全部权限授予用户U2和U3

```
GRANT ALL PRIVILIGES
    ON TABLE Student, Course
    TO U2, U3;
```



【例】 DBA把在数据库dbsc中建立表的权限授予用户U4

```
GRANT CREATE TABLE
    ON DATABASE dbsc
    TO U4;
```

权限回收

- 具有授予权的用户可以通过回收语句将所授予的权限回收
- 回收语句格式为：

REVOKE <权限1>[,<权限2>]...

[ON <对象类型> <对象名>]

FROM <用户1>[,<用户2> ...|PUBLIC]

[RESTRICT|CASCADE];

  - **CASCADE**选项表示回收权限时要引起级联操作，要把转授出去的权限一起回收。
  - **RESTRICT**选项表示，只有用户没有将拥有的权限转授给其他用户时才能回收该用户的权限，否则系统将拒绝执行。

【例】回收用户**User2**对学生表**Student**的查询权和删除权。

```
REVOKE SELECT, DELETE
    ON TABLE Student
    FROM User2
    CASCADE;
```

【例】把用户**U4**修改学生学号的权限收回。

```
REVOKE UPDATE(Sno)
    ON TABLE Student
    FROM U4;
```

SQL授权机制小结

- DBA拥有对数据库中所有对象的所有权限，并可以根据应用的需要将不同的权限授予不同的用户。
- 用户对自己建立的基本表和视图拥有全部的操作权限，并且可以用GRANT语句把其中某些权限授予其他用户。
- 被授权的用户如果有“继续授权”的许可，还可以把获得的权限再授予其他用户。
- 所有授予出去的权力在必要时又都可以用REVOKE语句收回。