数据库MySQL使用方法

- 核心功能词

SQL功能	动词
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

表 3.2　SQL 的数据定义语句

操作对象	操作方式		
	创　建	删　除	修　改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视　图	CREATE VIEW	DROP VIEW	
索　引	CREATE INDEX	DROP INDEX	ALTER INDEX

- 模式操作

定义模式:

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>[<
表定义子句>|<视图定义子句>|<授权定义子句>]

eg:

```
CREATE SCHEMA `S-T` AUTHORIZATION WANG;
CREATE SCHEMA AUTHORIZATION WANG;
```

如果没有指定<模式名>，那么<模式名>隐含为<用户名>，隐含为用户名WANG

> mysql不用authorization，而是用grant！

> 分号是个分隔符，看到分号就标志着本条sql语句结束

删除模式:
DROP SCHEMA <模式名><CASCADE|RESTRICT>
CASCADE(级联):删除模式的同时把该模式中所有的数据库对象全部删除。
RESTRICT(限制):如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。
当该模式中没有任何下属的对象时才能执行。

eg:

```
drop SCHEMA `TEST` ;
```

- 基本表的定义、删除与修改

定义基本表
CREATE TABLE <表名>
（<列名> <数据类型>[<列级完整性约束条件>]
[，<列名> <数据类型>[<列级完整性约束条件>]] …
[，<表级完整性约束条件>]）；
如果完整性约束条件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。

eg:

```
CREATE TABLE Student
  (Sno CHAR(9) PRIMARY KEY,  /*列级完整性约束条件*/
  Sname CHAR(20) UNIQUE,  /* Sname取唯一值*/
  Ssex CHAR(2),
  Sage SMALLINT,
  Sdept CHAR(20)
  FOREIGN KEY (Cpno) REFERENCES Course(Cno)
);
```

```
CREATE TABLE SC
  (Sno CHAR(9),
  Cno CHAR(4),
  Grade SMALLINT,
  PRIMARY KEY (Sno, Cno),
  /*主码由两个属性构成，必须作为表级完整性进行定义*/
  FOREIGN KEY (Sno) REFERENCES Student(Sno),
  /*表级完整性约束条件，Sno是外码，被参照表是Student */
  FOREIGN KEY (Cno) REFERENCES Course(Cno)
  /*表级完整性约束条件， Cno是外码，被参照表是Course*/
);
```

> 选择模式使用 Use `S-T`;

- SQL数据类型

数据类型	含义
CHAR(n)	长度为n的定长字符串
VARCHAR(n)	最大长度为n的变长字符串
INT	长整数（也可以写作INTEGER，4字节）
SMALLINT	短整数（2字节）
NUMERIC(p，d)	定点数，由p位数字（不包括符号、小数点）组成，小数后面有d位数字
REAL	取决于机器精度的浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为n位数字
DATE	日期，包含年、月、日，格式为YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为HH:MM:SS

- 建表
- 方法一：在表名中明显地给出模式名

```
Create table ` S-T `.Student（......）; /*模式名为 S-T*/
Create table ` S-T `.Cource（......）;
Create table ` S-T `.SC（......）;
```

- 方法二：在创建模式语句中同时创建表

```
Create schema XXX create table XXX
```

- 方法三：设置所属的模式

```
Use schema XXX
```

- 特殊:

> 显示当前的搜索路径： SHOW search_path;

> 设置搜索路径并创建搜索表

```
SET search_path TO “S-T”, PUBLIC;
Create table Student（......）;
```

> MYSQL没有搜索路径的概念

- 修改基本表

```
ALTER TABLE <表名>
[ ADD <新列名> <数据类型> [完整性约束 ] ]/*新增列*/
[ DROP <完整性约束名> ]/*删除列*/
[ ALTER COLUMN<列名> <数据类型> ]; /*修改列*/
MYSQL 使用 ALTER TABLE STUDENT MODIFY CLASS INT;/*修改列*/
```

[例8]向Student表增加“入学时间”列，其数据类型为日期型。

```
ALTER TABLE Student ADD S_entrance DATE;
/*不论基本表中原来是否已有数据，新增加的列一律为空值。*/
```

[例9]将年龄的数据类型由字符型（假设原来的数据类型是字符型）改为整数。

```
ALTER TABLE Student ALTER COLUMN Sage INT;  #Mysql 会报错
```

[例10]增加课程名称必须取唯一值的约束条件。

```
ALTER TABLE Course ADD UNIQUE(Cname);
```

- alter table表名称 change字段名称 字段名称 字段类型[是否允许非空];
- alter table表名称 modify字段名称 字段类型 [是否允许非空];
- ```
  ALTER TABLE `2020exp1`.`course_ave`
  CHANGE COLUMN `AVE_SCORE` `AVE_SCORE` FLOAT NULL DEFAULT NULL ;
  ```

eg:
修改表expert_info中的字段birth,允许其为空

```
alter table expert_info change birth birth varchar(20) null;
```

- 删除基本表

```
DROP TABLE <表名>[RESTRICT | CASCADE];
```

索引的建立与删除

建立索引（Index）的目的：加快查询速度

谁可以建立索引:

- DBA或 表的属主（即建立表的人）
- DBMS一般会自动建立以下列上的索引:
  PRIMARY KEY, UNIQUE

谁维护索引:DBMS自动完成
使用索引:DBMS自动选择是否使用索引以及使用
哪些索引.

建立索引

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名>(<列名>[<次序>][,<列名>[<次序>] ]…);
```

> mysql中没有cluster index
> 聚簇索引：元组按照索引键值的顺序存储

- 聚簇索引是顺序结构与数据存储物理结构一致的一种索引
  - 通常物理顺序结构只有一种，那么一个表的聚簇索引也只能有一个，设置了主码，系统默认就为表加上了聚簇索引。若需要用其他字段作为索引，可在设置主码之前自己手动的先添加上唯一的聚簇索引，然后再设置主码。
```

    - B+Tree的叶子节点上的data就是数据本身（主码或其它字段）
- 非聚簇索引记录的物理顺序与逻辑顺序没有必然的联系，与
  数据的存储物理结构没有关系
    - 一个表对应的非聚簇索引可以有多条，根据不同列的约束可以建立不同要求的非聚簇索引
    - B+Tree的叶子节点上的data，并不是数据本身，而是数据存放的地址

[例13]在Student表的Sname（姓名）列上建立一个聚簇索引

```
CREATE CLUSTER INDEX Stusname
ON Student(Sname);
```

> 1. 在最经常查询的列上建立聚簇索引以提高查询效率
> 2. 一个基本表上最多只能建立一个聚簇索引
> 3. 经常更新的列不宜建立聚簇索引

[例14]为学生-课程数据库中的Student，Course，SC三个表建立索引。
Student表按学号升序建唯一索引
Course表按课程号升序建唯一索引
SC表按学号升序和课程号降序建唯一索引

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);
CREATE UNIQUE INDEX Coucno ON Course(Cno);
CREATE UNIQUE INDEX SCno ON SC(Sno ASC，Cno DESC);
```

删除索引

```
DROP INDEX <索引名>;
```

> 删除索引时，系统会从数据字典中删去有关该索引的描述

[**例15**]删除Student表的Stusname索引

```
DROP INDEX Stusname;
或者
DROP INDEX Stusname on student;
```

数据查询

语句格式:

```
SELECT [ALL|DISTINCT] <目标列表达式>
[，<目标列表达式>] …
FROM <表名或视图名>[，  <表名或视图名> ] …
[ WHERE <条件表达式> ]
[ GROUP BY <列名1> [ HAVING <条件表达式> ] ]
[ ORDER BY <列名2> [ ASC|DESC ] ];
```

单表查询

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname，2004-Sage /*假定当年的年份为2004年*/
FROM Student；
```

[例5]查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名,并使用别名

```
SELECT Sname NAME，'Year of Birth: '  BIRTH，2004-Sage BIRTHDAY，
LOWER(Sdept)  DEPARTMENT
FROM Student；
```

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	is
王敏	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

指定DISTINCT关键词，去掉表中重复的行 (只出现一个)

```
SELECT DISTINCT Sno
FROM SC；
```

表3.4 常用的查询条件

查询条件	谓 词
比 较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空 值	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

[例9] 查询考试成绩有不及格的学生的学号。 比较

```
SELECT DISTINCT Sno
FROM SC
WHERE Grade<60；
```

[例11] 查询年龄不在20~23岁之间的学生姓名、系别和年龄(0-19，24-Infty) 范围

```
SELECT Sname，Sdept，Sage
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23；
```

[例13] 查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。集合

```
SELECT Sname, Ssex
FROM Student
WHERE Sdept NOT IN ( 'IS', 'MA', 'CS' );
```

字符匹配

匹配串为含通配符的字符串
！ %：任意多个字符； _：单个字符（汉字是2个字符，要用两个）

[例15] 查询所有姓刘学生的姓名、学号和性别。

```
SELECT Sname, Sno, Ssex
FROM Student
WHERE Sname LIKE '刘%';
```

[例18] 查询所有不姓刘的学生姓名。

```
SELECT Sname, Sno, Ssex
FROM Student
WHERE Sname NOT LIKE '刘%';
```

！使用换码字符ESCAPE '\'将通配符转义为普通字符

[例19] 查询DB_Design课程的课程号和学分。

```
SELECT Cno, Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
    or  WHERE Cname LIKE "DB/_Design" ESCAPE "/";
```

谓词： IS NULL或 IS NOT NULL
！“IS”不能用“=”代替

[例22] 查所有有成绩的学生学号和课程号。

```
SELECT Sno, Cno
FROM SC
WHERE Grade IS NOT NULL;
```

逻辑运算： AND 和 OR

[例23] 查询计算机系年龄在20岁以下的学生姓名。

```
SELECT Sname
FROM Student
WHERE Sdept= 'CS' AND Sage<20;
```

ORDER BY子句

[例25] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT *
FROM Student
ORDER BY Sdept，Sage DESC;
```

聚集函数

计数

```
COUNT（[DISTINCT|ALL] *）
COUNT（[DISTINCT|ALL] <列名>）
```

计算总和

```
SUM（[DISTINCT|ALL] <列名>）
```

计算平均值

```
AVG（[DISTINCT|ALL] <列名>）
```

最大最小值

```
MAX（[DISTINCT|ALL] <列名>）
MIN（[DISTINCT|ALL] <列名>）
```

[例26]查询学生总人数。

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

[例27]查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

[例28]计算1号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno= ' 1 ';
```

[例30] 查询学生200215012选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM SC， Course
WHERE Sno='200215012' AND
SC.Cno=Course.Cno;
```

GROUP BY 子句

作用 对查询结果分组后，聚集函数将分别作用于每个组

[例32] 查询选修了3门以上课程的学生学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING COUNT(*) >3;
```

！**HAVING短语作用于组，从中选择满足条件的组。**

连接查询

等值连接:

连接运算符为=

```
SELECT Student.*, SC.*  FROM Student, SCWHERE Student.Sno = SC.Sno;
```

自然连接:

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
FROM Student, SCWHERE Student.Sno = SC.Sno;
```

自身连接:

一个表与其自己进行连接

[例35] 查询每一门课的间接先修课（即先修课的先修课）

```
SELECT FIRST.Cno, SECOND.Cpno
FROM Course FIRST, Course SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

外连接 ：

- **LEFT JOIN（左连接）**：获取左表所有记录，即使右表没有对应匹配的记录。
- **RIGHT JOIN（右连接）**： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。

[例 36]查询每个学生及其选修课程的情况

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
FROM Student LEFT OUT JOIN SC ON (Student.Sno=SC.Sno);
```

执行结果：

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
200215121	李勇	男	20	CS	1	92
200215121	李勇	男	20	CS	2	85
200215121	李勇	男	20	CS	3	88
200215122	刘晨	女	19	CS	2	90
200215122	刘晨	女	19	CS	3	80
200215123	王敏	女	18	MA	NULL	NULL
200215125	张立	男	19	IS	NULL	NULL

复合条件连接：

WHERE子句中含多个连接条件

[例37] 查询选修2号课程且成绩在90分以上的所有学生

```
SELECT Student.Sno, Sname
FROM Student, SC
WHERE Student.Sno = SC.Sno AND/*连接谓词*/
    SC.Cno= '2' AND SC.Grade > 90; /*其他限定条件 */
```

嵌套查询：

IN、比较、ANY/ALL、EXISTS

[例38] 查询选修2号课程的学生姓名。 **IN**

```
SELECT Sname /*外层查询/父查询*/
FROM Student
WHERE Sno IN
    (SELECT Sno /*内层查询/子查询*/
    FROM SC
    WHERE Cno= '2');
```

[例41] 找出每个学生超过他选修课程平均成绩的课程号。 **比较**

```
SELECT Sno， Cno
FROM SC x
WHERE Grade >=(SELECT AVG(Grade)
FROM SC y
WHERE y.Sno=x.Sno);
```

需要配合使用比较运算符

> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值
>= ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

[例42] 其他系中比计算机科学某一学生年龄小的学生姓名和年龄 ANY/ALL

```
SELECT Sname, Sage
FROM Student
WHERE Sage < ANY (SELECT Sage
                  FROM Student
                  WHERE Sdept= 'CS')
        AND Sdept <> 'CS'; /*父查询块中的条件 */
```

EXISTS ： 带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。

[例44]查询所有选修了1号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE EXISTS
    (SELECT *
    FROM SC
    WHERE Sno=Student.Sno AND Cno= '1');
```

- 用EXISTS/NOT EXISTS实现全称量词(难点)

SQL语言中没有全称量词∀ （For all）

可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$(\forall x)P \equiv \neg (\exists x(\neg P))$

[例46] 查询选修了全部课程的学生姓名。不存在没有被选修的课程（双重否定）

```
SELECT Sname             /*1.任意一个学生A*/
FROM Student
WHERE NOT EXISTS         /*2.不存在一个课程a*/
    (SELECT *
    FROM Course
    WHERE NOT EXISTS     /*3.在选课记录表里不存在A对a的选课记录*/
        (SELECT *
        FROM SC
        WHERE Sno= Student.Sno
        AND Cno= Course.Cno
        )
    );
```

集合查询

[例48] 查询计算机科学系的学生或年龄不大于19岁的学生。

```
SELECT *
FROM Student
WHERE Sdept= 'CS'
UNION
SELECT *
FROM Student
WHERE Sage<=19;
```

并集 UNION　　**交集** INTERSECT　　**差集** EXCEPT

A EXCEPT B = A - B

Select语句一般形式

```
SELECT [ALL|DISTINCT]
<目标列表达式> [别名] [，<目标列表达式> [别名]] …
FROM <表名或视图名> [别名]
[，<表名或视图名> [别名]] …
[WHERE <条件表达式>]
[GROUP BY <列名1>
[HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]
```

数据更新

插入数据

- 两种插入数据方式
  1.插入元组
  2.插入子查询结果(可以一次插入多个元组)

插入元组:

格式:

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

Into子句没有出现的属性列，新元组在这些列上取Null;
在表定义时说明了Not Null的属性列不能取空，否则报错;

【例1】 将一个新学生元组（学号：200215128；姓名：陈春；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", "陈春", "男", "IS", 18)

                or
 INSERT
  INTO Student
  VALUES ("200215128", "陈春", "男", "IS", 18)
```

插入子查询:

[例4] 对每一个系，求学生的平均年龄，并把结果存入数据库

```
INSERT
INTO Dept_age(Sdept, Avg_age)
SELECT Sdept, AVG(Sage)
FROM Student
GROUP BY Sdept;
```

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

修改元组:

[例5] 将学生200215121的年龄改为22岁

```
UPDATE Student
SET Sage=22
WHERE Sno="200215126";
```

[例6] 将某个学生的年龄增加1岁

```
UPDATE Student
SET Sage=Sage+1
WHERE Sno="200215126";
```

带子查询修改:

[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'=     /*'CS'='CS'*/
    (SELETE Sdept
    FROM Student
    WHERE Student.Sno = SC.Sno);
```

RDBMS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

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

[例9] 删除所有的学生选课记录。

```
DELETE
FROM SC;
```

视图

视图(View)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，不存放视图对应的数据
- 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

什么时候使用视图

- 经常用到的查询，或较复杂的联合查询应当创立视图，以优化查询效率
- 查询逻辑复杂，或别的查询需要查询这个查询的结果时
- 如果你的查询需要连接几个表的数据的时候你可以做成视图，那么你在业务端只需要查询这张视图就可以了
- 涉及到权限管理方面，用来保护敏感数据
- 比如某表中的部分字段含有机密信息，不应当让低权限的用户访问，这时候给这些用户提供一个适合他们权限的视图，供他们阅读自己的数据即可

定义视图

建立视图

语句格式

```
CREATE VIEW
<视图名> [(<列名> [，<列名>]…)]
AS <子查询>
[WITH CHECK OPTION]；
```

WITH CHECK OPTION表示对视图进行UPDATE、INSERT和DELETE等操作时要保证满足视图定义中的谓词条件（即，子查询中的条件表达式）

[例1] 建立信息系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有信息系的学生。

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

对IS_Student视图的更新操作：

- 修改操作：自动加上Sdept= 'IS'的条件
- 删除操作：自动加上Sdept= 'IS'的条件
- 插入操作：自动检查Sdept属性值是否为'IS'
  - 如果不是，则拒绝该插入操作
  - 如果没有提供Sdept属性值，则自动定义Sdept为'IS

基于多个基表的视图

[例3] 建立信息系选修了1号课程的学生视图。（成绩单）

```
CREATE VIEW IS_S1(Sno，Sname，Grade)
AS
SELECT Student.Sno，Sname，Grade
FROM Student，SC
WHERE Sdept= 'IS' AND
Student.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；
```

删除视图

语句的格式：

```
DROP VIEW <视图名>；
```

- 该语句从数据字典中删除指定的视图定义
- 如果该视图上还导出了其他视图，使用CASCADE级联删除语句，把该视图和由它导出的所有视图一起删除
- 删除基表时，由该基表导出的所有视图定义都必须
  显式地使用DROP VIEW语句删除

级联删除：

```
DROP VIEW IS_S1 CASCADE;
```

查询视图

RDBMS实现视图查询的方法
视图消解法（View Resolution），步骤：

- 进行有效性检查
- 转换成等价的对基本表的查询
- 执行修正后的查询

[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；
```

[例10] 查询选修了1号课程的信息系学生

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

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。

[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；

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;
```

目前多数关系数据库管理系统对行列子集视图的查询均能正确转换，但是对非行列子集视图的查询不一定能转换，因此这类查询应当直接对基本表进行

更新视图

[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_S中插入一个新的学生记录：200215129，赵新，20岁
INSERT
INTO IS_Student
VALUES(‘95029’，‘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');

[例14] 删除信息系学生视图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：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：视图S_G为不可更新视图。
UPDATE S_G
SET Gavg=90
WHERE Sno= ‘200215121’;
这个对视图的更新无法转换成对基本表SC的更新，系统无法修改各科成绩，以使平均成绩成为90

- **允许对行列子集视图进行更新**

  行列子集视图是满足以下条件的视图

  - 从单个基本表导出，且只是去掉了基本表的某些行或某些列，但是保留了基本表的主码
- **对其他类型视图的更新不同系统有不同限制**

视图的作用

1. **视图能够简化用户的操作**
    - 简化查询等操作
2. **视图使用户能以多种角度看待同一数据**
    - 便于多用户共享同一数据库时
3. **视图对重构数据库提供了一定程度的逻辑独立性**
    - 有时数据库的逻辑结构改变了，但是应用程序不用变
4. **视图能够对机密数据提供安全保护**
    - 机密数据不出现在低权限用户的视图中
5. **适当的利用视图可以更清晰的表达查询**

触发器

定义触发器

CREATE TRIGGER语法格式

```
CREATE TRIGGER <触发器名>
{BEFORE | AFTER} <触发事件> ON <表名>
FOR EACH {ROW | STATEMENT}
[WHEN <触发条件>]
<触发动作体>
```

定义触发器的语法说明:

1. 创建者：表的拥有者
2. 触发器名
3. 表名：触发器的目标表
4. 触发事件：INSERT、DELETE、UPDATE AFTER/BEFORE
5. 触发器类型

- 行级触发器（FOR EACH ROW）
- 语句级触发器（FOR EACH STATEMENT）
  [例11]建立老师表TEACHER，要求每个教师的应发工资不低于3000元，应发工资是工资列Sal与扣除项Deduct之和。

```
create table TEACHER
( Eno NUMERIC(4) PRIMARY KEY,
Ename CHAR(10),
Job CHAR(8),
Sal NUMERIC(7,2),
Deduct NUMERIC(7,2),
Deptno MUMERIC(2),
CONSTRAINT TEACHERFKEY FOREIGN KEY(Deptno),
REFERENCES DEPT(Deptno),
CONSTRAINT C1 CHECK(Sal+Deduct>=3000)      约束命名子句
);
```

［例18］ 定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
BEFORE INSERT OR UPDATE ON Teacher
/*触发事件是插入或更新操作*/
FOR EACH ROW /*行级触发器*/
BEGIN /*定义触发动作体，是PL/SQL过程块*/
IF (new.Job='教授') AND (new.Sal < 4000) THEN
new.Sal :=4000;
END IF;
END;
```

激活触发器

一个数据表上可能定义了多个触发器

- 同一个表上的多个触发器激活时遵循如下的执行顺序：
  （1） 执行该表上的BEFORE触发器；
  （2） 激活触发器的SQL语句；
  （3） 执行该表上的AFTER触发器

删除触发器

删除触发器的SQL语法：

```
DROP TRIGGER <触发器名> ON <表名>;
```

触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。
［例21］ 删除教师表Teacher上的触发器Insert_Sal

```
DROP TRIGGER Insert_Sal ON Teacher;
```