

CREATE/DROP/ALTER 语句

【模式】

定义模式 CREATE SCHEMA

CREATE SCHEMA 模式名 AUTHORIZATION 用户名;

eg. 为用户 WANG 定义一个 S-T 模式

CREATE SCHEMA "S-T" AUTHORIZATION WANG;

eg. 为用户 WANG 定义一个模式

CREATE SCHEMA AUTHORIZATION WANG; //未定义模式名的情况下，默认模式名为用户名

删除模式 DROP SCHEMA

DROP SCHEMA 模式名 CASCADE; //级联：把该模式中所有的表、视图之类的一起删除

DROP SCHEMA 模式名 RESTRICT; //限制：若该模式下已定义了表或者视图等等，则拒绝执行删除语句

【基本表】

定义基本表 CREATE TABLE

CREATE TABLE 表名

(列名 1 数据类型 列级完整性约束条件, //如没有列级完整约束条件，可以不写

列名 n 数据类型 列级完整性约束条件,

表级完整性约束条件 1,

表级完整性约束条件 n

);

(1) 数据类型

CHAR(n) 长度为 n 的字符型

VARCHAR(n) 最大长度为 n 的变长字符型

NUMBER(n) 长度为 n 的数字型

INT 长整型 (4B)

SMALLINT 短整型 (2B)

BIGINT 大整型 (8B)

FLOAT(n) 精度至少为 n 位数字的浮点数

DATE 日期，格式为 YYYY-MM-DD

TIME 时间，格式为 HH:MM:SS

(2) 列级完整性约束条件

PRIMARY KEY //主码：当只有一个主码时，可直接在对应的属性列标注

NOT NULL //非空：表示该属性列不能取空值

UNIQUE //唯一值：表示该属性列只能取唯一值

CHECK(条件) //检查：检查该列是否满足某个条件，如 CHECK(某属性>20)

(3) 表级完整性约束条件

PRIMARY KEY(列名 1, 列名 n) //实体完整性：当主码由多个属性构成时，必须作为表级完整性进行定义

FOREIGN KEY(列名 1) REFERENCES 被参照表(列名 1) //参照完整性

eg.

CREATE TABLE TAB1

(Sno VARCHAR(10),

Cno NUMBER(10),

Grade INT NOT NULL,

PRIMARY KEY(Sno, Cno),

FOREIGN KEY(Sno) REFERENCES TAB2(Sno)

);

在模式中定义表

一个模式包含多种基本表，有三种方式在模式中定义基本表。

(1) 创建表时指出模式

```
CREATE TABLE 模式名.表名  
(列定义语句,  
完整性约束语句  
);
```

(2) 创建模式时直接定义表

```
CREATE SCHEMA 模式名 AUTHORIZATION 用户名  
CREATE TABLE 表名  
(列定义语句,  
完整性约束语句  
);
```

(3) 设置所属的模式

这样在创建表的时候不用给出模式名

修改基本表 ALTER TABLE

(1) 增加新的属性列

```
ALTER TABLE 表名 ADD 新列名 数据类型 完整性约束条件;
```

eg.向 SC 表中增加时间列，数据类型为日期型

```
ALTER TABLE SC ADD COLUMN Time DATE;
```

(2) 增加列级完整性约束条件

```
ALTER TABLE 表名 ADD 列级完整性约束条件;
```

eg.向 SC 表中增加 Cname 列必须取唯一值的约束条件

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

(3) 增加表级完整性约束条件

```
ALTER TABLE 表名 ADD 表级完整性约束条件;
```

eg.向 SC 表中增加 Cno 为外码，参照表是 Student 表

```
ALTER TABLE SC ADD FOREIGN KEY(Cno) REFERENCES Student(Cno);
```

(4) 删除列

```
ALTER TABLE 表名 DROP 列名 CASCADE; //级联：引用了该列的其他对象（例如视图）一起删除
```

```
ALTER TABLE 表名 DROP 列名 RESTRICT; //限制：若该列被其他对象引用，则拒绝删除
```

(5) 删除指定的完整性约束条件

```
ALTER TABLE 表名 DROP CONSTRAINT 完整性约束名 CASCADE; //级联
```

```
ALTER TABLE 表名 DROP CONSTRAINT 完整性约束名 RESTRICT; //限制
```

(6) 修改列

```
ALTER TABLE 表名 ALTER COLUMN 列名 数据类型;
```

eg.将 SC 表中原有的 Sage(假设是字符型)修改为整型

```
ALTER TABLE SC ALTER COLUMN Sage INT;
```

删除基本表 DROP TABLE

(1) DROP TABLE 表名 CASCADE; //级联：删除该表时，相关的依赖对象，例如视图，都会被删除

(2) DROP TABLE 表名 RESTRICT; //限制：删除该表时，若被其他表的约束所引用（例如其他表的 CHECK、FOREIGN KEY 等等），或者有视图等等，都不能被删除

【索引】

建立索引 CREATE UNIQUE/CLUSTER INDEX

(1) 建立唯一索引

CREATE UNIQUE INDEX 索引名 ON 表名(列名 1 次序, 列名 n 次序);

eg. 为 SC 表按学号升序和课程号降序建立唯一索引

CREATE UNIQUE INDEX SCno ON SC(Sno ASC, Cno DESC);

(2) 建立聚簇索引

CREATE CLUSTER INDEX 索引名 ON 表名(列名 1 次序, 列名 n 次序);

修改索引 (重命名) ALTER INDEX

ALTER INDEX 旧索引名 RENAME TO 新索引名;

eg. 将 SC 表的 SCno 索引名改为 SCSno

ALTER INDEX SCno RENAME TO SCSno;

删除索引 DROP INDEX

DROP INDEX 索引名;

eg. 删除 SC 表的 SCSno 索引;

DROP INDEX SCSno;

SELECT 语句

一般格式

SELECT DISTINCT/ALL 目标列表表达式	//要显示的属性列
FROM 表名/视图名	//查询的对象
WHERE 条件表达式	//查询条件
GROUP BY 列名 HAVING 条件表达式	//查询结果分组
ORDER BY 列名 次序;	//最终查询结果排序

【基本查询】

SELECT 目标列表表达式

(1) 查询指定列: SELECT 列名 1, 列名 n

eg. 查询 TAB 表的 X 属性列和 Y 属性列

SELECT X, Y

FROM TAB;

(2) 查询全部列: SELECT *

eg. 查询 TAB 表的全部记录

SELECT *

FROM TAB;

(3) 查询计算后的值: SELECT 表达式 //可以是算术表达式、字符串常量、函数等等

eg. 查询 TAB 表 (假定有一项属性 age 记录人们的年龄) 中人们的出生日期

SELECT 2022-age

FROM TAB;

(4) 改变查询结果的列标题: SELECT 列名 别名

eg. 查询 TAB 表中的 X 和 Y 属性列, 并在结果中用别名 x1 和 y1 显示

SELECT X x1, Y y1

FROM TAB;

(5) 取消查询结果中的重复行: SELECT DISTINCT 列名

eg. 查询 TAB 表中的 X 属性, 并去掉结果中的重复列

SELECT DISTINCT X, //如果没有用 DISTINCT, 则默认为 ALL

FROM TAB;

(6) 聚集函数

注意：当聚集函数遇到空值时，都跳过空值，只处理非空值

聚集函数只能用于 SELECT 语句和 GROUP BY 中的 HAVING 子句（见后部分）

① 统计元组的个数

COUNT (*) //某个元组的一个或部分取空值时，不影响统计结果

eg. 查询 Student 表中学生的总数

```
SELECT COUNT(*)
```

```
FROM SC;
```

② 统计某一系列值的个数

COUNT (DISTINCT/ALL 列名)

如果指定 DISTINCT，则表示计算时要取消重复值。若不指定，则默认为 ALL，表示不取消重复值

③ 计算某一系列值的平均数（该列必须为数值型）

AVG (DISTINCT/ALL 列名)

eg 计算 SC 表中的平均成绩（Grade）

```
SELECT AVG(Grade)
```

```
FROM SC;
```

④ 计算某一系列值的总和（该列必须为数值型）

SUM (DISTINCT/ALL 列名)

⑤ 计算某一系列值的最大值/最小值

MAX/MIN (DISTINCT/ALL 列名)

WHERE 条件表达式

(1) 比较大小

常用比较运算符：= > < >= <= !=(或者<>) !> !<

eg. 查询 SC 表中全体计算机学生的姓名

```
SELECT Sname
```

```
FROM SC
```

```
WHERE Sdept='cs';
```

eg. 查询 TAB 表中 X>20 的 Y

```
SELECT Y
```

```
FROM TAB
```

```
WHERE X>20;
```

(2) 确定范围

WHERE 列名 BETWEEN 最小值 AND 最大值;

eg. 查询 TAB 表上 age 在 20 到 30 之间的人的 name 和 sex

```
SELECT name, sex
```

```
FROM TAB
```

```
WHERE age BETWEEN 20 AND 30;
```

WHERE 列名 NOT BETWEEN 最小值 AND 最大值;

eg. 查询 TAB 表上 age 不在 20 到 30 之间的人的 name 和 sex

```
SELECT name, sex
```

```
FROM TAB
```

```
WHERE age NOT BETWEEN 20 AND 30;
```

(3) 确定集合

WHERE 列名 IN ('列名 1', '列名 n');

eg. 在 SC 表的 Sdept 列中查找属于计算机专业 (CS)、数学专业 (MA)、信息专业 (IS) 的学生姓名 (S_name)

```
SELECT S_name
```

```
FROM SC
```

```
WHERE Sdept IN ('CS', 'MA', 'IS');
```

WHERE 列名 NOT IN ('列名 1', '列名 n');

eg. 在 SC 表的 Sdept 列中查找既不是计算机专业 (CS) 也不是数学专业 (MA) 的学生姓名 (S_name)

```
SELECT S_name
```

```
FROM SC
```

```
WHERE Sdept NOT IN ('CS', 'MA');
```

(4) 字符匹配

通配符: 写在字符串当中, 用来求一些有特殊条件的字符串

% 表示任意长度 (可以为 0) 的字符串。如 a%b, 表示以 a 开头, b 结尾的任意长度字符串

_ 表示单个字符。如 a_b, 表示以 a 开头 b 结尾的长度为 3 的字符串

注意: 在 ASCII 码表中, 一个汉字的长度为 2

转义字符: 字符串中紧跟在转义字符后的字符 '%' 或 '_' 不再具有通配符的含义

设置转义字符的语句为 **ESCAPE '符号'**, 该符号可以自己设置, 一般采用 '\'

例如需要查找的字符串为 '50%', 那么应写 '50\%', 否则会查找以 50 开头的不定长字符串

WHERE 列名 LIKE '字符串' ESCAPE '\';

eg. 在 Student 表中查找所有姓刘且全名为 3 个汉字的学生的 Sname、Sno 和 Ssex

```
SELECT Sname, Sno, Ssex
```

```
FROM Student
```

```
WHERE Sname LIKE '刘_ _ _';
```

eg. 在 SC 表中查找课程名 (Cname) 为 DB_Design 的课程号 (Cno)

```
SELECT Cno
```

```
FROM SC
```

```
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

WHERE 列名 NOT LIKE '字符串' ESCAPE '\';

eg. 在 Student 表中查找所有不姓刘的学生的 Sname

```
SELECT Sname
```

```
FROM Student
```

```
WHERE Sname NOT LIKE '刘%';
```

(5) 空值查询

WHERE 列名 IS NULL;

eg. 在 SC 表中查询缺少成绩 (Grade) 的学生的姓名 (Sname)

```
SELECT Sname
```

```
FROM SC
```

```
WHERE Grade IS NULL;
```

WHERE 列名 IS NOT NULL;

eg. 查询全部有成绩 (Grade) 的学生的姓名 (Sname)

```
SELECT Sname
```

```
FROM SC
```

```
WHERE Grade IS NOT NULL;
```

(6) 多重条件查询

WHERE 条件表达式 AND 条件表达式;

WHERE 条件表达式 OR 条件表达式;

可以用 AND 或者 OR 将上述各类条件表达式组合在一起。其中, AND 的优先级大于 OR

GROUP BY 列名 HAVING 条件表达式

用于将查询结果按某一列或多列的值分组, 值相等的为一组

目的是细化聚集函数的作用对象, 分组后聚集函数将作用于每一个组, 每一组都有一个函数值

(1) GROUP BY 列名

eg. 求 SC 表中, 各个课程号 (Cno) 下相应的选课人数 (Sno)

```
SELECT Cno, COUNT(Sno)
FROM SC
```

GROUP BY Cno; //表示具有相同 Cno 值的元组为一组, 对每一组用 COUNT 进行计算, 求得该组的人数

(2) GROUP BY 列名 HAVING 筛选条件

与 WHERE 的区别:

- HAVING 用于从组中选择满足条件的组
- WHERE 用于从基本表或视图中选择满足条件的元组 (注意: WHERE 子句不可以接聚集函数)

eg. 在 SC 表中查询平均成绩 (Grade) 大于等于 90 的学生学号 (Sno) 和平均成绩 (Grade)

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

ORDER BY 次序

ORDER BY 列名 1 列名 n ASC

对查询结果按照一个或多个属性列的升序排列 (若不表明次序, 默认为升序)

ORDER BY 列名 1 列名 n DESC

对查询结果按照一个或多个属性列的降序排列

eg. 查序 SC 表中 Cno 为 3 的学生的 Sno 和 Grade, 结果按照 Grade 的降序排列

```
SELECT Sno, Grade
FROM SC
WHERE Cno='3'
ORDER BY Grade DESC;
```

【连接查询】

两表连接查询

WHERE 表名 1.列名 1 比较运算符 表名 2.列名 2; //当列名在参与连接的各表中唯一时, 可省去表名前缀

eg. 查询 Student 表中学生的情况以及 SC 表中他们对应的选课情况, 要求在一个查询结果中展示

```
SELECT Student.*, SC.* //若两个表中有相同名的属性列, 自然连接
FROM Student, SC
```

WHERE Student.Sno = SC.Sno; //用 Sno 作为连接字段, 将两个表连接在一起

/*若想获得自然连接, 则列举全部属性列, 并去掉一个相同的属性列即可。可以将上述 SELECT 语句改写如下*/

```
SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
```

eg. 在 Student 表和 SC 表中, 查询选修了 2 号课程 (Cno='2') 且成绩 (Grade) 在 90 分以上的学生学号 (Sno) 和姓名 (Sname)

```
SELECT Student.Sno, Sname
FROM Student, SC
WHERE Student.Sno=SC.Sno AND SC.Cno='2' AND SC.Grade>90; // AND 用作多重条件
```

单表连接查询

eg. 在 Course 表中查询先修课的先修课 (即间接先修课), 其中课程号为 Cno, 先修课程号为 Cpno

```
SELECT FIRST.Cno, SECOND.Cpno //利用下述别名进行选择
FROM Course FIRST, Course SECOND //为这个表取两个别名
WHERE FIRST.Cpno=SECOND.Cno;
```

外连接查询

将悬浮元组保留在结果关系中, 没有属性值的位置填上 NULL

(1) 左外连接查询

FROM 表名 1 LEFT OUTER JOIN 表名 2 ON(连接条件); //也可以将 ON 换成 USING, 去掉结果中的重复值

(2) 右外连接查询

FROM 表名 1 RIGHT OUTER JOIN 表名 2 ON(连接条件);

eg. 以 Student 表为主体列, 排出每个学生的基本情况和选课情况 (SC 表中), 没选课的学生依旧保留在结果中

SELECT Student.Sno, Sname, Ssex, Sage, Sdept, Cno, Grade

FROM Student LEFT OUTER JOIN SC ON (Student.Sno=SC.Sno); //左外连接

多表连接查询

WHERE 表名 1.列名 1=表名 2.列名 2 AND 表名 2.列名 2=表名 3.列名 3

多表连接一般是先进行两个表的连接操作, 再将其连接结果与第三个表执行连接

eg. 从 Student 表、SC 表、Course 表中查询每个学生的学号 (Sno)、姓名 (Sname)、课程 (Cname) 和成绩 (Grade)

SELECT Student.Sno, Sname, Cname, Grade

FROM Student, SC, Course

WHERE Student.Sno=SC.Sno AND SC.Cno=Course.Cno;

【嵌套查询】

查询块: SELECT-FROM-WHERE

嵌套查询: 将一个查询块嵌套在另一个查询块的 WHERE 子句或者 HAVING 子句

其中, 上层的查询块称为外层查询/父查询; 下层的查询块称为内层查询/子查询

注意: 子查询的 SELECT 语句中不能使用 ORDER BY 子句, ORDER BY 子句只能对最终查询结果排序

IN-子查询

父查询与子查询之间用 IN 连接

WHERE 列名 IN (子查询);

eg. 查找与刘晨同一个专业的同学

① **SELECT Sdept**

FROM Student

WHERE Sname='刘晨';

查找结果为刘晨在 CS 专业, 之后再查找 CS 专业的学生

② **SELECT Sno, Sname, Sdept**

FROM Student

WHERE Sdept='CS';

→将上述①②构造为嵌套查询

SELECT Sno, Sname, Sdept

FROM Student

WHERE Sdept IN

(SELECT Sdept

FROM Student

WHERE Sname='刘晨'); //本例的子查询条件不依赖于父查询, 这类子查询称为**不相关子查询**

比较运算符-子查询

父查询与子查询之间用比较运算符连接

WHERE 列名 比较运算符 (子查询); //当用户能确切知道内层查询返回的是单个值时, 可用比较运算符连接

eg. 在 SC 表中, 找出每个学生 (Sno) 超过他自己选修课程平均成绩 (Grade) 的课程号 (Cno)

SELECT Sno, Cno

FROM SC x

//x 是表 SC 的别名, 又称为元组变量, 可以用来表示 SC 的一个元组

WHERE Grade >=

(SELECT AVG(Grade)

FROM SC y

WHERE y.Sno=x.Sno); /*本例的子查询条件依赖于父查询, 这类子查询称为**相关子查询**, 整个查询称为**相关嵌套查询***/

ANY/ALL-子查询

WHERE 列名 比较运算符 ANY/ALL (子查询); //有的系统中 ANY 用 SOME 代替

谓词	语义	与聚集函数或 IN 的等价转换
>ANY	大于子查询结果中的某个值	>MIN
>ALL	大于子查询结果中的所有值	>MAX
<ANY	小于子查询结果中的某个值	<MAX
<ALL	小于子查询结果中的所有值	<MIN
>=ANY	大于等于子查询结果中的某个值	>=MIN
>=ALL	大于等于子查询结果中的所有值	>=MAX
<=ANY	小于等于子查询结果中的某个值	<=MAX
<=ALL	小于等于子查询结果中的所有值	<=MIN
=ANY	等于子查询结果中的某个值	IN
=ALL	等于子查询结果中的所有值 (通常无实际意义)	--
!=(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值	NOT IN

eg. 在 Student 表中, 查询不是 CS 专业的学生中, 比 CS 专业任意一个学生年纪小的学生姓名和年龄

```
SELECT Sname, Sage
```

```
FROM Student
```

```
WHERE Sage<ANY
```

```
(SELECT Sage
```

```
FROM Student
```

```
WHERE Sdept='CS')
```

```
AND Sdept <> 'CS'; //注意这是父查询块中的条件
```

EXISTS-子查询

EXISTS 代表存在量词 (与之对应的为 NOT EXISTS)

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'

逻辑蕴涵表达:

eg1. 在 SC 表中查询至少选修了 1 号学生选修的全部课程 (Cno) 的学生的学号 (Sno)

本例可进行如下分析:

查询学号为 x 的学生, 对所有的课程 y, 只要 1 号学生选修了课程 y, 则 x 也选修了 y

令 p 表示"学生 1 号选修了课程 y"

令 q 表示"学生 x 选修了课程 y"

则上述查询可以表示为 $(\forall y)p \rightarrow q$

通过等价转换, 可得 $(\forall y)p \rightarrow q \equiv (\exists y(\neg(p \rightarrow q))) \equiv \neg(\exists y(\neg(\neg p \vee q))) \equiv \neg \exists y(p \wedge \neg q)$

因此最终用 SQL 实现的表达式为 $\neg \exists y(p \wedge \neg q)$, 其语义: 不存在这样的课程 y, 学生 1 号选修了 y, 而学生 x 没有选

```
SELECT DISTINCT Sno
```

```
FROM SC SCX
```

```
WHERE NOT EXISTS
```

```
(SELECT * //由 EXISTS 引出的子查询, 其目标列表达式通常都用*
```

```
FROM SC SCY
```

```
WHERE SCY.Sno='1' AND NOT EXISTS
```

```
(SELECT *
```

```
FROM SC SCZ
```

```
WHERE SCZ.Sno=SCX.Sno AND SCZ.Cno=SCY.Cno)
```

```
);
```


eg2. 基于 SC 表, 查询选修了全部课程 (Course 表) 的学生姓名 (Student 表)

本例可进行如下分析

令 p 表示"课程 x 被学生 y 选修了", 则有 $(\forall x)p \equiv \neg(\exists x(\neg p))$, 其语义: 查询没有任何课程是其不选修的学生 y

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
    (SELECT *
     FROM Course
     WHERE NOT EXISTS
        (SELECT *
         FROM SC
         WHERE Sno=Student.Sno AND Cno=Course.Cno)
    );
```

【集合查询】

多个 SELECT 语句的结果可以进行集合的并 (UNION)、交 (INTERSECT)、差 (EXCEPT) 操作

注意: 参加集合操作的各查询结果的列数必须相同; 对应项的数据类型也必须相同

SELECT 语句 1 UNION/INTERSECT/EXCEPT SELECT 语句 2

UNION 并操作

UNION 合并查询结果时, 系统会自动去掉重复元组, 若需保留, 则采用 UNION ALL

eg1. 在 Student 表中查询 CS 专业的学生和年龄不大于 19 岁的学生

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

eg2. 在 SC 表中查询选修了课程 1 或者课程 2 的学生

```
SELECT Sno
FROM SC
WHERE Cno='1'
UNION
SELECT Sno
FROM SC
WHERE Cno='2';
```

INTERSECT 交操作

eg. 查询既选修了课程 1 又选修了课程 2 的学生

```
SELECT Sno
FROM SC
WHERE Cno='1'
INTERSECT
SELECT Sno
FROM SC
WHERE Cno='2';
```

EXCEPT 差操作

eg. 查询 CS 专业的学生与年龄不大于 19 岁的学生的差集

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

【基于派生表的查询】

子查询出现在 FROM 子句时，子查询将生成临时的派生表，成为主查询的查询对象。

FROM (子查询) AS 别名(属性列名 1, 属性列名 n);

//如果子查询中没有聚集函数，派生表可以不指定属性列，子查询 SELECT 子句后面的列名为其默认属性

//AS 可以省略，但必须为派生表关系指定一个别名;

eg1.找出每个学生超过他自己选修课程平均成绩的课程号

```
SELECT Sno, Cno  
FROM SC, (SELECT Sno, Avg(Grade) FROM SC GROUP BY Sno) AS Avg_sc(avg_sno, avg_grade)  
WHERE SC.Sno = Avg_sc.avg_sno AND SC.Grade>=Avg_sc.avg_grade;
```

eg2.查询所有选修了 1 号课程的学生姓名

```
SELECT Sname  
FROM Student, (SELECT Sno FROM SC WHERE Cno='1') AS SC1  
WHERE Student.Sno=SC1.Sno;
```

INSERT/UPDATE/DELETE 语句

插入数据 INSERT

(1) 插入元组

INSERT

INTO 表名 (列名 1, 列名 n)

VALUES (常量 1, 常量 n); //字符串常量要用单引号 (') 括起来

假设现有 TAB1 表，共有 C1 到 C4 四列，其中 C4 列是字符串常量

situation1. 明确给出新增元组要在哪些属性上赋值 (插入数据包含全部属性列)

```
INSERT  
INTO TAB1 (C1, C2, C3, C4)  
VALUES (1, 2, 3, '4');
```

situation2. 仅指出在 TAB1 表上插入元组 (插入数据包含全部属性列)

```
INSERT  
INTO TAB1  
VALUES (1, 2, 3, '4');
```

 //这种情况表示要在 TAB1 表全部各列赋值，且插入数据的顺序必须和列的顺序对应

situation3. 明确给出新增元组要在哪些属性列上赋值 (插入数据不包含全部属性列)

```
INSERT  
INTO TAB1 (C1, C2, C3)  
VALUES (1, 2, 3);
```

 //这种情况下，C4 列会被赋为 NULL

注意：当表定义说明了 NOT NULL 时，不赋值会出错

situation4. 仅指出在 TAB1 表上插入元组 (插入数据不包含全部属性列)

INSERT

INTO TAB1

VALUES (1, 2, 3, NULL); //这种情况必须明确给出未赋值的属性列为 NULL

(2) 插入子查询结果

INSERT

INTO 表名 (属性列 1, 属性列 n)

子查询; //子查询嵌套在 INSERT 语句中生成要插入的批量数据

eg. 假设现有 TAB1 表 (如上), 并按 C1 列分组求 C2 列的平均值, 并存入 TAB2 表 (其中 TAB2 表的 C1 列存放 C1, avg_C2 列存放 C2 列的均值)

INSERT

INTO TAB2 (C1, avg_C2)

SELECT C1, AVG(C2)

FROM TAB1

GROUP BY C1;

修改数据 UPDATE

UPDATE 表名

SET 列名 1=表达式 1, 列名 n=表达式 n

WHERE 条件; //修改指定表中满足 WHERE 子句条件的元组; 若省略 WHERE, 表示要修改表中的所有元组

situation1. 修改某一个元组的值

UPDATE TAB1

SET C4='0'

WHERE C1=1;

situation2. 修改多个元组的值

UPDATE TAB1

SET C3=C3+1;

situation3. 带子查询的修改语句

UPDATE TAB1

SET C4='0'

WHERE C1 IN

(SELECT C1

FROM TAB2

WHERE avg_C2=2;

删除数据 DELETE

DELETE

FROM 表名

WHERE 条件; //删除指定表中满足 WHERE 子句条件的元组; 若省略 WHERE, 表示要删除表中的所有元组

注意: DELETE 语句删除的是表中的数据, 并不是表的定义, 表的定义仍在数据字典当中

situation1. 删除某一个元组的值

DELETE

FROM TAB1

WHERE C1=1;

situation2. 删除多个元组的值

DELETE

FROM TAB1; //TAB1 将变成空表

situation3. 带子查询的删除语句

```
DELETE
FROM TAB1
WHERE C1 IN
    (SELECT C1
     FROM TAB2
     WHERE avg_C2=2;
```

VIEW 视图

定义视图 CREATE VIEW

CREATE VIEW 视图名 (列名 1, 列名 n) //若省略列名, 则该视图由子查询中 SELECT 的目标列字段组成
AS 子查询

WITH CHECK OPTION; //若添加该句, 则表示对视图进行增删改时要满足子查询中的条件表达式

在以下情况中必须明确指定组成视图的列名:

1. 某个目标列不是单纯的列名, 而是聚集函数或列表表达式
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

行列子集视图: 由单个基本表导出, 仅去掉了基本表的某些行和某些列, 但保留了主码

若某些视图是建立在另一个表的全部属性列上的, 即视图与基本表的各列是一一对应的。那么当修改基本表的结构时, 基本表和视图的映像关系会被破坏。这种情况最好在修改基本表后删除该视图, 然后重建该视图

eg1. 建立 TAB1 的视图

```
CREATE VIEW V_TAB1
AS
SELECT C1, C2, C3, C4
FROM TAB1
WHERE C1=1;
```

eg2. 建立 C4 为 4 时 TAB1 的视图, 并保证以后每次增删改时都要满足 C4 为 4 的条件

```
CREATE VIEW V_TAB2
AS
SELECT C1, C2, C3, C4
FROM TAB1
WHERE C4='4'
WITH CHECK OPTION;
```

eg3. 建立在一个或多个已定义号的视图上

```
CREATE VIEW V_TAB3
AS
SELECT C1, C2, C3
FROM V_TAB1
WHERE C2=2;
```

eg4. 为减少冗余数据, 定义基本表时一般只存放基本数据。当需要使用计算得出的派生数据时, 可以设置在视图中的派生属性列上, 也称为虚拟列。带虚拟列的视图也称为带表达式的视图

```
CREATE VIEW V_TAB4(C1, new_C2)
AS
SELECT C1, 10+C2
FROM TAB1;
```

eg5. 分组视图：带有聚集函数和 GROUP BY 子句

```
CREATE VIEW V_TAB5(C1, avg_C2)
```

```
AS
```

```
SELECT C1, AVG(C2)
```

```
FROM TAB1
```

```
GROUP BY C1;
```

删除视图 DROP VIEW

DROP VIEW 视图名 CASCADE; //若使用 CASCADE 级联删除语句，则将把该视图导出的所有视图一并删除

eg1.

```
DROP VIEW V_TAB2;
```

eg2.

```
DROP VIEW V_TAB1 CASCADE; //由 V_TAB1 导出的 V_TAB3 也一并删除
```

查询视图和更新视图

视图定义后，对视图进行查询和更新的语句和语法与基本表相同

视图的查询与更新最终都会转换为对基本表的查询和更新，这一过程也被称为视图消解

一般来说，行列子集视图的查询和更新都可以顺利转换，其他则不一定

(视图消解参考教材 P122-P126)

空值

判断一个属性是否为空值

属性 IS NULL

属性 IS NOT NULL

空值的运算

空值与另一个值的算术运算结果为空值

空值与另一个值的比较运算结果为 UNKNOWN

在查询语句中，只有使 WHERE 和 HAVING 子句的选择条件为 TRUE 的元组才会被选出作为输出结果（即不包括 UNKNOWN 的情况）