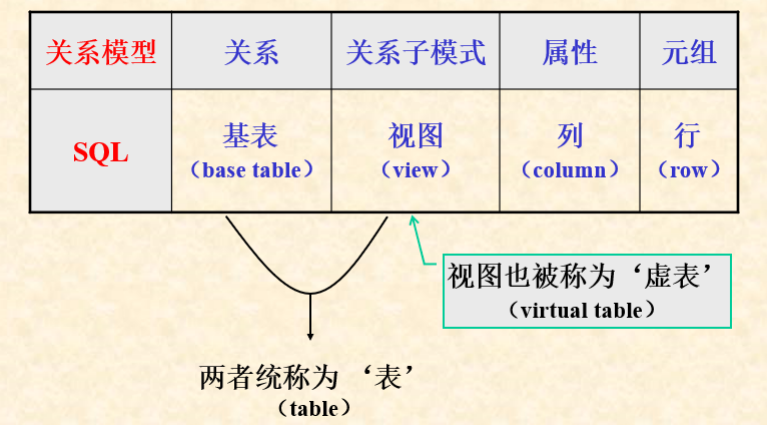
Chapter3 SQL92

3.1 SQL概貌

基本概念的变化



使用方式

自含式：独立的交互式命令行语言

嵌入式：嵌入到某种高级程序设计语言（被称为‘主语言’）中使用

嵌入方式：嵌入式SQL（ESQL），函数调用

SQL的功能

数据定义功能：–基表/视图的定义与删除 –索引/集簇的定义与删除

数据操纵功能：–数据的查询/插入/删除/修改 –简单的数值计算与统计功能

数据控制功能：–数据的完整性、安全性、并发控制、故障恢复等

数据交换功能：–会话、连接、游标、诊断、动态SQL

扩展功能

SQL的特点

1）综合统一

–集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动

2）高度非过程化

–用SQL进行数据操作，只要提出“做什么”，而无须指明“怎么做”

3）面向集合的操作方式

4）以同一种语法结构提供多种使用方式

–SQL既是独立的语言，又是嵌入式语言

5）语言简洁，易学易用

3.2 SQL数据定义功能

基表的创建命令

❑ 基表的创建命令

```
CREATE TABLE tablename (
    colname datatype [ NOT NULL ]
    { , colname datatype [ NOT NULL ] }
) ;
```

➢ 命令的格式定义符号

1) [...] /* 0或1个 */
2) { ... } /* 0到若干个 */
3) CREATE TABLE
4) NOT NULL

基表的修改命令

1) 对基表结构的修改

• 表中属性的增加/删除

```
ALTER TABLE <基表名> ADD <列名> <数据类型>;
ALTER TABLE <基表名> DROP <列名> ;
```

2) 数据完整性约束的修改

基表的删除命令

➢ 删除指定的表及其数据

```
DROP TABLE <基表名>;
```

3.3 SQL数据操纵功能

SQL的基本查询功能

```
目 标 子 句:   SELECT * | colname { , colname ... }
范 围 子 句:   FROM tablename { , tablename ... }
条 件 子 句:   [ WHERE search_condition ]
分 组 子 句:   [ GROUP BY colname { , colname ... }
分组查询子句: [ HAVING group_condition ] ]
排序输出子句: [ ORDER BY colname [ ASC | DESC ]
                { , colname [ ASC | DESC ] ... } ];
```

select

–可用‘ * ’来代替表中的所有属性

–可使用保留字‘distinct’来消除结果关系中的重复元组

where

包括'单个关系中的元组选择条件'以及'关系与关系之间的联接条件'都需要在WHERE子句中通过一定的逻辑表达式显式地表示出来。

在FROM子句中给出的关系只是表明此次查询需要访问这些关系，它们之间是通过笛卡儿乘积运算进行合并的

如果需要执行它们之间的'θ-联接'或'自然联接'运算，则需要在WHERE子句中显式地给出它们的联接条件

常用谓词

DISTINCT　一般用于select语句，用于消除重复结果（否则不会消除）

BETWEEN ... AND ...，NOT BETWEEN ... AND ... 注意：如果是between 18 and 21，则包括18和21

LIKE，NOT LIKE：LIKE val1 [ESCAPE val2]

模版（pattern）：val1

–下划线（_）：可以匹配任意一个字符

–百分号（%）：可以匹配任意一个字符串（包括长度为0的空字符串）

–其它字符：只能匹配其自身

转义指示字符：val2

–紧跟在转义指示字符val2之后的'_'或'%'（包括转义字符自身）不再是通配符，而是其自身

IS NULL，IS NOT NULL　为空，不为空

自连接　如果在From中需要做表和自己的连接，此时需要进行换名操作

分层结构查询与集合谓词使用

子查询

独立子查询：比较简单，就不说了

相关子查询：

相关子查询的执行依赖于外部查询。多数情况下是子查询的WHERE子句中引用了外部查询的表。执行过程：

（1）从外层查询中取出一个元组，将元组相关列的值传给内层查询。（元组：就是表中的一条数据）

（2）执行内层查询，得到子查询操作的值。

（3）外查询根据子查询返回的结果或结果集得到满足条件的行。

（4）然后外层查询取出下一个元组重复做步骤1-3，直到外层的元组全部处理完毕

集合谓词：in，not in，all，any，exists

select语句间的运算

```
<子查询1> UNION [ ALL ] <子查询2>
<子查询1> INTERSECT [ ALL ] <子查询2>
<子查询1> EXCEPT [ ALL ] <子查询2>
```

SQL统计、计算、分类的功能

函数名称	参数类型	结果类型	说 明
COUNT	any (can be *)	numeric	count of rows
SUM	numeric	numeric	sum of argument
AVG	numeric	numeric	average of argument
MAX	char or numeric	same as argument	maximum value
MIN	char or numeric	same as argument	minimum value

1）在count中需要用distinct：消除重复元素（否则会重复计数）

【例4】 查询居住有公司客户的城市数

```
SELECT   COUNT ( distinct  city )
FROM     Customers
```

2）不能在WHERE子句中直接使用统计函数

不能在WHERE子句中直接使用统计函数

```
SELECT   cid
FROM     Customers
WHERE    discnt < max ( discnt )
```

```
SELECT   cid
FROM     Customers  c1
WHERE    discnt < ALL ( SELECT  max( c2.discnt )
                        FROM    Customers  c2 )
```

3）空值（NULL）在统计函数中的处理

在使用统计函数对一个集合中的元素进行统计计算时，将忽略其中的'空'值元素

在一个空集上进行统计计算时，其返回结果如下：

–COUNT() 返回 0，其它的统计函数均返回空值 NULL

❑ 在Customers表中插入一条记录

('c009', 'New_Customer', 'New York')

➢ 没有给出discnt属性的取值（系统自动为该属性赋上空值）

【例6】

```
SELECT  count(*),
        count(discnt),  count(distinct discnt),
        avg(discnt),  sum(discnt)
FROM    customers;
```

4）需要保证出现在select语句中但没有被聚集的属性只能是出现在group by子句中的那些属性

换句话说，任何没有出现在group by子句中的属性如果出现在select子句中的话，它只能出现在聚集函数内部，否则这样的查询就是错误的

❑ 例如，下面的查询是错误的

```
SELECT dept_name, ID, avg( salary )
FROM instructor
GROUP BY dept_name
```

➢ 在一个特定分组（通过dept_name定义）中的每位教师都有不同的ID，每个分组只能输出一个元组，无法确定输出哪个

5）Having 语句

❑分组查询子句： HAVING group_condition

➢根据GROUP BY子句的分组结果，定义分组查询条件

–在HAVING子句中给出的查询条件是定义在分组后的元组集合上的，通常是根据该集合上的某种统计计算的结果来定义查询条件

–只有满足条件group_condition的元组集合才会被保留下来，用于生成最终的查询结果

where->group->having->select->order

3.4 SQL的更新功能

删除功能

❑删除功能

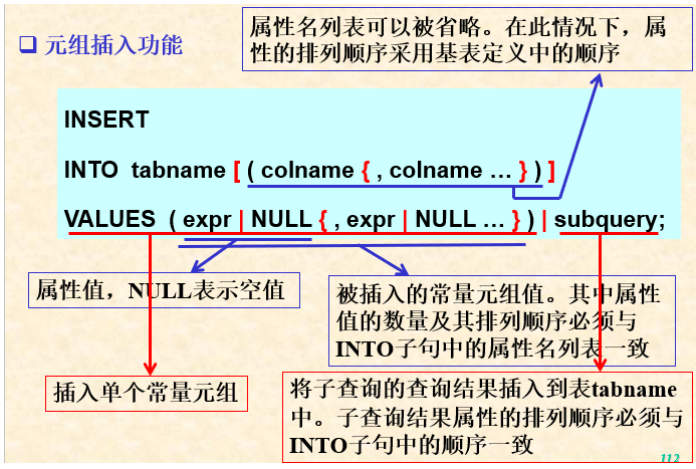
```
DELETE FROM table_name
[ WHERE search_condition ] ;
```

➢在table_name表中删除符合条件search_condition的元组

–在省略WHERE子句的情况下，删除表table_name中的所有元组

➢WHERE子句的构造方式与映像语句中的WHERE子句一样，也可以在其中嵌入子查询（subquery）

插入功能



修改功能

```
UPDATE table_name
SET colname = expr | NULL | subquery, .....
[ WHERE search_condition ] ;
```

➢修改指定基表table_name中满足WHERE条件的元组

–即：用SET子句中的赋值语句修改相关元组上的属性值

3.5 视图

❑ 视图（view）

➢ 由若干张表经映像语句构筑而成的表

➢ 又称为：导出表（drived table）

❑ 视图（view）与基表（base table）的区别

➢ 被称为视图的二维表本身（结构与数据）并不实际存在于数据库内，而仅仅保留了其构造信息（有关视图的定义信息）。因此视图又被称为‘虚表’（virtual table）

➢ 当用户执行视图上的访问操作时，数据库管理系统将根据视图的定义命令将用户对于视图的访问操作转换成相应基表上的访问操作

视图定义

```
CREATE VIEW <视图名> [ ( <列名> { , <列名> ... } ) ]
    AS <映像语句> [ WITH CHECK OPTION ]
```

➢ 创建一个以<视图名> 为表名的视图，对应的数据查询语句是<映像语句>。以<映像语句>作查询所得到的查询结果即是该视图中的元组

➢ 如果没有给视图中的属性命名，则用<映像语句>的SELECT子句中的属性名作为视图属性的属性名。否则视图中的属性必需与<映像语句>的SELECT子句中的结果属性一一对应

➢ WITH CHECK OPTION用于约束视图上的修改操作：如果允许在该视图上执行更新操作，则其更新后的结果元组仍然必需满足视图的定义条件。即通过该视图插入或修改后的新元组能够通过该视图上的查询操作查出来

嵌套定义：可以利用已有的视图定义新的视图

【例】定义一个由课程名及该课程的平均成绩构成的视图

```
CREATE VIEW C_G ( cn, Cavg )
    AS SELECT cn, AVG( G )
       FROM   S_C_G
       GROUP BY cn
```

经定义已经存在的视图

```
CREATE VIEW S_C_G ( sn, cn, G )
   AS SELECT sn, cn, G
      FROM   S, C, SC
      WHERE  S.sno = SC.sno and C.cno = SC.cno
```

视图删除

```
DROP VIEW <视图名>
```

➢ 在执行视图的删除操作时，将连带删除定义在该视图上的其它视图

➢ 例：执行 ‘DROP VIEW S_C_G’命令将连带删除之前用下述命令创建的视图 C_G 和视图 S_C_G

视图上的操作

➢ 对视图可以作查询操作

– 视图上的查询操作将首先被改写为基表上的查询操作，然后才能得到执行

➢ 一般不允许执行视图上的更新操作，只有在特殊情况下才可以进行：

– 视图的每一行必须对应基表的惟一一行

– 视图的每一列必须对应基表的惟一一列

视图的优点

提高了数据独立性，简化用户观点，提供自动的安全保护功能

有了视图之后，数据独立性大为提高，无论基表扩充还是分解，均不影响对概念数据库的认识。只需重新定义视图内容，而不改变面向用户的视图形式，因而保持了关系数据库逻辑上的独立性。同时视图也简化了用户观点，用户不必了解整个模式，仅需将注意力集中于它所关注的领域，大大方便了使用。此外，视图还提供了自动的安全保护功能。