第3章 关系数据库语言SQL

3.1 数据库的用户接口

一、数据库操作与数据库语言

1.广义的数据库操作包括：

数据定义（data definition）在数据库中创建、撤销、修改数据模式
数据查询（data query）在数据库中查询/检索所需的数据
数据操纵（data manipulation）在数据库中增加、删除、修改数据
数据控制（data control）控制用户对数据库中数据的访问权限

2.数据库语言（database language）：DBMS提供的语言，以支持用户进行数据库操作。包括：

数据定义语言 (data definition language, DDL)定义、撤销和修改数据模式对象：表、视图、索引...
查询语言 (query language, QL)查询数据
数据操纵语言 (data manipulation language, DML)插入/删除/修改数据，简单的数值计算与统计功能
数据控制语言 (data control language, DCL)控制数据访问权限

3.数据库语言的特点

①面向记录（record-oriented）的语言 vs. 面向集合（set-oriented）的语言

层次、网状数据库向用户呈现包含数据的**逻辑属性+物理存储细节**的数据模式，因此使用**物理指针**进行**导航式访问**（navigational access），**一次操作一个记录**，操作过程繁琐，效率低下
　　相应的数据库语言称**面向记录的语言**（record-oriented language）
关系数据库的数据模式抽象级别高，使用**联想式访问**（associative access），即**按数据的内容（属性值）来访问数**据，**一次操作得到一个记录的集合**
　　相应的数据库语言称**面向集合的语言**（set-oriented language）

②命令式编程（imperative programming）范式 vs. 声明式编程（declarative programming）范式

早期的**层次、网状数据库**的语言是**命令式/过程性**的，使用这样的数据库语言进行数据库查询操作，用户（程序员）的负担很重；而且应用程序的可维护性差，程序与数据之间的独立性（independence）很差。
关系数据库提供了**非过程性（non-procedural）/声明式（性）**的数据库语言SQL，大大方便了用户对数据库的查询操作。
　　某些**过程性**机制（e.g. 流程控制、存储过程，等）是有用的。因此， SQL语言进行了**过程化扩充**（procedural extensions），提供了SQL的扩充模块：SQL/PSM (Persistent Stored Modules)

③交互式（interactive）使用方式 vs. 嵌入式（embedded）使用方式

SQL语言等数据库语言往往**不是计算完备的（computationally complete）**
在某些数据库应用中，要实现数据“管理”与“计算”的集成，可以将数据库语言嵌入（embedding）到程序设计语言（e.g. Java, C）中——这样的高级语言称宿主语言（host language）。
因此，数据库语言就有两种或两种使用方式：交互式和嵌入式。

二、用户接口与前端开发工具

1.用户接口（user interface）

用户接口是DBMS提供给用户操作数据库的界面
用户接口将用户对数据库的操作请求以数据库语言语句（和命令）的形式提交给系统，并接受系统的处理结果、将结果呈现给用户
用户接口提供了两种操作数据库的方式：交互方式和批处理方式（即编写应用程序）
用户接口风格可有：文本的和GUI

2.前端开发工具（front-end development tools）

DBMS厂商或第三方提供的数据库应用集成化开发工具 （e.g. ORACLE Developer/2000，Sybase Powerbuilder）

3.2 SQL语言概况

3.SQL的起源

SQL (/ˌɛsˌkjuːˈɛl/ S-Q-L, or /ˈsiːkwəl/ "sequel"; Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS).
SQL was initially developed at IBM in 1970s. The first version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasi-relational (准关系) database management system, System R.

2.SQL的标准 vs. SQL的厂商实现

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features.

- SQL-86 (SQL1, Standard Query Language, by ANSI, 1986); SQL-87 (by ISO, 1987, ISO 9075:1987)
- SQL-89 (Structured Query Language, by ANSI/ISO, 1989)
- SQL-92 (SQL2, by ANSI/ISO, 1992)
- SQL:1999 (SQL3, by ANSI/ISO, 1999)
- SQL:2003；SQL:2006；SQL:2008；SQL:2011
- SQL:2016 (by ANSI/ISO, 2016, ISO/IEC 9075:2016)

SQL实现：各个数据库厂商在其RDBMS中实现的SQL（语言名称、功能有别，且与SQL标准不同）

3.SQL的特点

关系数据库的标准化语言
集DDL、QL、DML、DCL于一体的数据库语言
非过程化的声明式（性）语言
进行了过程化扩充的数据库语言
嵌入式和交互式相似语法的语言
English-like、简单易学的语言

功能	SQL语句的动词
数据定义	CREATE, DROP, ALTER
数据查询	SELECT
数据操纵	INSERT, DELETE, UPDATE
数据控制	GRANT, REVOKE

3.3 SQL数据定义语言

数据定义语言（Data Definition Language, DDL）主要用于按照某种逻辑数据模型（e.g., 关系模型）的概念来定义与修改数据库的概念模式与外模式。
就关系数据库而言，使用SQL DDL来定义关系数据库的模式对象，如：基表（base table）、视图（view）、索引（index）、触发器（trigger），等。
DDL语句的编译结果成为数据库的元数据，存储于DBMS的数据字典（DD）中。

表 3-3　SQL 模式对象定义语句

语句关键字	模式对象	说　明
CREATE	TABLE、VIEW、TRIGGER、INDEX、CLUSTER、STORED PROCEDURE	创建基表、视图、触发器、索引、簇集、存储过程。
DROP	TABLE、VIEW、TRIGGER、INDEX、CLUSTER、STORED PROCEDURE	删除基表、视图、触发器、索引、簇集、存储过程。
ALTER	TABLE	修改基表，包括增加列、增加与删除主、外键等。

1.基表模式的创建

```
CREATE TABLE <表名>
(  <列名> <数据类型>[ <列级完整性约束> ]
[, <列名> <数据类型>[ <列级完整性约束>] ]
…
[, <表级完整性约束> ]
);

<表名>：所要定义的基表（即关系）模式的名称
<列名>：组成该基表的各个列（即属性）的名称
<列级完整性约束>：涉及相应属性列的完整性约束
<表级完整性约束>：涉及一个/多个属性列的完整性约束
```

常用完整性约束

实体完整性约束：PRIMARY KEY
唯一性约束：UNIQUE
非空值约束：NOT NULL
引用完整性约束：FOREIGN KEY

```
[例1]  创建一个“部门”表dept，它由部门号deptno、部门名dname、城市loc三个属性组成。其中，部门号是主键，部门名取值不能为空且唯一，城市只能取值于‘Shanghai’, ’Nanjing’, ‘Wuhan’, ‘Xian’, ‘Beijing’ 。
相应SQL DDL语句如下：
CREATE TABLE dept
( deptno INT PRIMARY KEY,
  dname VARCHAR(12)  NOT NULL UNIQUE,
```

```
  loc VARCHAR(10) CHECK ( loc IN ('Shanghai', 'Nanjing', 'Wuhan', 'Xian',
'Beijing'))
);//定义的全部是列级完整性约束

[例2] 创建一个"职员"表emp，它由工号、姓名、工种、主管经理、薪水、佣金、所在部门等属性组成。相应
SQL DDL语句如下：
CREATE TABLE emp
      ( empno   INT PRIMARY KEY,
        ename   VARCHAR(10)  NOT NULL,
        job     VARCHAR( 9),
        mgr     INT  REFERENCES  emp(empno),
        sal     DEC(7,2)  CHECK  (sal>1000.0),
        comm    DEC(7,2)  DEFAULT  NULL,//DEFAULT是缺省值，不是完整性约束!
        deptno  INT  NOT NULL  REFERENCES  dept(deptno)ON UPDATE CASCADE
       )；//全部是列级完整性约束
CREATE TABLE emp
      ( empno   INT NOT NULL,
        ename   VARCHAR(10)  NOT NULL,
        job     VARCHAR( 9),
        mgr     INT  REFERENCES  emp(empno),
        sal     DEC(7,2)  CHECK  (sal>1000.0),
        comm    DEC(7,2)  DEFAULT  NULL,
        deptno  INT  NOT NULL,
        PRIMARY KEY (empno),//定义的是表级完整性约束
        FOREIGN KEY (deptno) REFERENCES  dept(deptno)ON UPDATE CASCADE
        )；//当PK或FK由多个属性组成时，只能定义表级完整性约束
```

2.基表模式的修改与撤消

增加列

```
ALTER TABLE <表名> [ ADD <新列名> <数据类型> [ 完整性约束 ] ];

<表名> ：要修改的基表名称
ADD子句：增加新列（和新的完整性约束）

[例3] 向emp表增加"性别" 列，其数据类型为长度为2位的定长字符串，性别不能为NULL。
    ALTER TABLE emp ADD gender CHAR(2) NOT NULL；
```

撤消基表

```
DROP TABLE <表名>;

基表定义被撤消，且表中数据、表上的索引全被删除
系统从数据字典中删去有关该基表及其索引的元数据

[例4] 撤消emp表：     DROP TABLE emp;

如何实现从基表中删除属性列定义？
    把基表中要保留的列及其内容复制到一个新基表中
    删除原基表
    再将新基表重命名（ RENAME ）为原表名
如何实现从基表中删除数据？——DELETE语句
```

补充定义主键

```
ALTER TABLE <表名> ADD PRIMARY KEY(<列名清单>);

要求定义为主键的<列名清单>中的每个列必须满足NOT NULL，全部列的组合取值必须唯一
```

撤销主键定义

```
ALTER TABLE <表名> DROP PRIMARY KEY;

暂时撤销主键，在插入新元组时可提高系统的性能
```

补充定义外键

```
ALTER TABLE <表名1>
    ADD FOREIGN KEY [ <外键名> ](<列名清单>)
             REFERENCES <表名2> (<列名清单>)
             [ ON DELETE { RESTRICT | CASCADE | SET NULL } ];
```

撤销外键定义

```
ALTER TABLE <表名1> DROP <外键名>;
```

修改属性列的数据类型

```
ALTER TABLE <表名> MODIFY <列名> <数据类型>;

[例5]  将emp表中职工号改为长度为8位的字符串类型。
    ALTER TABLE emp MODIFY empno CHAR(8);
注：修改原有属性列定义有可能会破坏列中已有数据
```

3.其他模式对象的定义与撤消

别名（alias）or 同义词（synonym ）

```
用简单的别名代替全名，书写和输入都比较方便
由于各个用户对同一数据对象可能有不同的命名习惯（如：salary vs. wages），若为不同用户定义不同的别名，则各个用户可以保留自己习惯的命名

定义别名
CREATE SYNONYM <标识符> FOR { <基表名>|<视图名> };
撤销别名
DROP  SYNONYM <标识符> ;
```

索引的建立与撤销

索引是物理存储路径，不属于逻辑数据模式
建立索引是加快数据查询速度的有效手段
建立索引
    DBA或表的创建者根据需要来建立
    大多数DBMS自动为以下属性列建立索引：
        PRIMARY KEY （或UNIQUE列）
维护索引
    DBMS自动完成
使用索引
    数据访问时DBMS自动选择是否使用索引、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>] ]…);
用<表名>指定要建索引的基表
索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
用<次序>指定索引值的排列次序：
    ASC表示升序（缺省情况）， DESC表示降序。
用关键字UNIQUE表明此索引的每一个索引值只对应唯一的数据记录
用关键字CLUSTER表示要建立的索引是“聚簇索引”

唯一值索引（ UNIQUE ）
对于已含重复值的属性列不能建UNIQUE索引
对某个列建立UNIQUE索引后，插入新记录时DBMS会自动检查新记录在该列上是否取了重复值——这相当于增加了一个UNIQUE约束
簇集索引（ CLUSTER ）
建立簇集索引后，基表中数据也需要按指定的簇集属性值的升序或降序存放——簇集索引的索引项顺序与表中记录的物理顺序一致

[例6] 为dept和emp两个表建立索引。其中dept表按deptno升序建立唯一索引，emp表按deptno降序建立唯一索引。
CREATE UNIQUE INDEX dno ON dept(deptno);
CREATE UNIQUE INDEX eno ON emp(deptno DESC);
```

撤销索引

```
DROP INDEX <索引名>;
撤销索引时，系统会从数据字典中删去有关该索引的定义（元数据）

[例7]  撤销emp表上的eno索引。
DROP INDEX  eno;
```

3.4 SQL数据查询语言

SQL数据查询语言（QL）使用查询语句（SELECT语句）查询数据库中的数据
在查询语句中：
SELECT子句【必需】指定需查询的项目
FROM子句【必需】指定被查询的基表或视图
WHERE子句规定查询条件
GROUP BY子句说明如何对查询结果进行分组
ORDER BY子句说明如何对查询结果进行排序

一、SELECT语句的语法

```
SELECT [ALL | DISTINCT] <列表达式> [, <列表达式>…]
FROM <表标识>  [<别名>] [, <表标识> [<别名>] … ]
   [ WHERE <查询条件> ]
   [ GROUP BY <列标识> [, <列标识>…] [ HAVING <分组条件> ] ]
   [ ORDER BY <列标识> | <序号> [ ASC | DESC ]
                       [, <列标识> | <序号> [ ASC | DESC ]… ] ];
```

SELECT子句：指定要查询出的结果：列表达式（属性列）的值
FROM子句：指定查询的数据源：基表或视图
WHERE子句：指定查询条件：逻辑表达式（其中可嵌套SELECT查询）
GROUP BY子句：按指定列的值对查询结果进行分组，列值相等的被分为一个组。通常会在每组中再用聚集函数计算其总计信息；HAVING短语：进一步筛选出满足“分组条件” 的组作为查询结果
ORDER BY子句：对查询结果按指定列值的升序或降序进行排序输出

1.选择子句：指定要查询出的结果

语法格式：SELECT [ALL | DISTINCT] <列表达式> [, <列表达式>...]

说明：ALL（缺省情况）：不去掉重复行地返回查询结果（查询结果为bag）
DISTINCT：对查询结果中的重复行只返回其中一行，即：消除结果关系中重复的元组（查询结果为set）

<列表达式>：是算术表达式，用于投影表中的列，或进一步对列值进行简单计算。定义如下：
列标识，形如[<表名> | <别名>.] <列名>，是一个<列表达式>
列标识的SQL（聚集）函数是一个<列表达式>
由<列表达式>、常量、算术运算符（+，-，X，/）及括号所组成的算术表达式是一个<列表达式>
“*” 表示一个表中的所有列，是一个特殊的<列表达式>
在有的SQL实现中， <列表达式>的值可以重命名，方法为：<列表达式> AS <新列名>

2.来源子句：指明查询的数据来源（基表或视图）

格式：FROM <表标识> [<别名>] [, <表标识> [<别名>] ...]

说明：<表标识>与<列标识>的区别如下：
<表标识>用以标识一个表（基表、视图或快照）形式：[<模式名>.] <表名>
<列标识>用以标识一个表中的一个列 形式：[<表名>|<别名>.] <列名>
可以在FROM子句中对一个表重新命名（定义别名）：<表名> <别名> 主要用于表（关系）的自连接运算

3.条件子句：是查询语句中的可选部分，用于定义查询条件（即结果关系中的元组必须满足的条件）

语法格式：WHERE <查询条件>

说明：“单个关系中的元组选择条件”和“两个关系之间的连接条件”都需要在条件子句中用逻辑表达式表示
查询条件具体包括：
简单条件：比较、BETWEEN、LIKE、IN和EXISTS
复合条件：由简单条件、逻辑运算符（NOT, AND, OR）及括号所组成的逻辑表达式
查询条件中还允许（多层）嵌套子查询

简单条件

```
比较条件——用于比较大小，可采用三种形式：
```

```
        <列标识>  IS [NOT] NULL | θ <列表达式> | θ [ALL | ANY | SOME] (<子查询>)
其中， θ为关系运算符。<子查询>中还可进一步嵌套<子查询>
BETWEEN条件——用于确定范围：
    <列标识>   [NOT] BETWEEN <列表达式1> AND <列表达式2>
LIKE条件——用于字符匹配：
    <列标识>   [NOT] LIKE ‘xx…x’
   其中，x可为字符（精确匹配）、_（单字符匹配）、%（任意多个字符匹配）。
IN条件——用于属于判断：
   <列标识> [NOT] IN (常量1, 常量2, …, 常量n) | (<子查询>)
   其中， <子查询>中还可进一步嵌套<子查询>
EXISTS条件——用于存在判断：
   [NOT] EXISTS （<子查询>）
   其中， <子查询>中还可进一步嵌套<子查询>
```

二、各种条件查询举例

下列2个关系（基表）作为查询的数据源：
dept (deptno, dname, loc)
emp (empno, ename, job, mgr, sal, comm, deptno)

无条件查询

```
1) 查询全部职员的工号与姓名：
         SELECT empno, ename  FROM emp;
2) 查询全部部门的编号、名称、所在地：
     SELECT deptno, dname, loc FROM dept;
或 SELECT  *  FROM dept;
3) 查询职员的所有可能的工种：
              SELECT DISTINCT job FROM emp;
4) 查询每个职员提薪20%后的薪水：
        SELECT empno, ename, sal*1.2 FROM emp;
5) 查询所有工种的薪水：
         SELECT ALL job, sal FROM emp;
         上述语句等价于：SELECT job, sal FROM emp;
         若想去掉结果中的重复行，则使用DISTINCT谓词：
   SELECT DISTINCT job, sal FROM emp;
注意 ：DISTINCT的作用对象是结果中的全体列所组成的行，而非单个列!
下列是错误的写法：
   SELECT DISTINCT job, DISTINCT sal  FROM emp;
```

比较条件查询

```
6) 查询所有销售人员的姓名、所在部门号：
      SELECT ename, deptno FROM emp WHERE job = ‘salesman’;
7) 查询薪水超过5000的职员：
      SELECT empno, ename, sal  FROM emp WHERE sal > 5000.0;
8) 查询没有佣金的职员：
      SELECT empno, ename FROM emp WHERE comm IS NULL;
错：SELECT empno, ename FROM emp WHERE comm = NULL;
```

范围查询

```
9) 查询薪水在3000与5000之间的职员：
       SELECT empno, ename  FROM emp
        WHERE sal BETWEEN 3000 AND 5000;
```

空值（NULL）注意事项

除IS [NOT] NULL之外，空值不满足任何其他查找条件
如果NULL参与算术运算，则该算术表达式的值为NULL
如果NULL参与比较运算，则结果视为false，•但在SQL-92标准中视为unknown
如果NULL参与聚集函数（aggregate functions）的运算，则除count(*)外的其它SQL聚集函数都忽略NULL

字符匹配查询

% （百分号） 代表任意长度（长度可以为0）的字符串
例：a%b表示以a开头，以b结尾的任意长度的字符串。如acb，addgb，ab 等都满足该串匹配
_ （下划线） 代表任意单个字符
例：a_b表示以a开头，以b结尾的长度为3的任意字符串。如acb、afb等都满足该串匹配

```
10）找出姓名以M打头的所有职员。
      SELECT deptno, ename  FROM emp
      WHERE ename LIKE ‘M%’;
11）找出姓名第三个字母为r的所有职员。
      SELECT deptno, ename  FROM emp
      WHERE ename LIKE ‘_ _r%’;
12）查询课程名称含有Database的课程号和学分。
      SELECT cno, credit FROM course
      WHERE cname LIKE ‘%Database%’;
```

属于判断查询

```
13）找出既不是经理又不是销售员的职员的薪水。
      SELECT ename, job, sal  FROM emp
      WHERE job NOT IN ( ‘manager’, ‘salesman’ ) ;
      如果不使用IN，则需要这样来写查询条件（麻烦）：
      WHERE (job < > ‘manager’) AND (job < > ‘salesman’ ) ;
```

连接查询

```
14）（两表/多表连接）查询职员Allen的工作所在地：连接条件
           SELECT ename, loc  FROM emp, dept
           WHERE ename = ‘Allen’ AND emp.deptno = dept.deptno;
15）（单表连接/自连接）查询薪水比Jones薪水高的职员：为基表定义别名以实现自连接
           SELECT x.empno, x.ename
           FROM emp x, emp y
           WHERE x.sal > y.sal AND y.ename = ‘Jones’;
16）（单表连接/自连接）查询薪水超过其部门经理的职员的姓名、以及该部门经理的姓名： 为基表定义别名以实现自连接
           SELECT worker.ename, manager.ename
           FROM emp worker, emp manager
           WHERE worker.mgr = manager.empno AND worker.sal > manager.sal;
```

存在查询

```
17）查询所有已雇用职员的部门。
    SELECT deptno, dname  FROM dept
    WHERE EXISTS (   SELECT *  FROM emp         //子查询（subquery）或称嵌套查询（nested query）
                     WHERE emp.deptno = dept.deptno   );
```

子查询 / 嵌套查询

```
18）查询与Jones相同工种的所有职员。
    SELECT empno, ename  FROM emp
    WHERE job = (  SELECT job   FROM emp        //通常情况下，运算符“=” 改用“IN” 更安全!如果有多个Jones，此条件就出错了!
                   WHERE ename = ‘Jones’   );
注意：子查询 的SELECT语句不能有ORDER BY子句；子查询可进一步嵌套。

19）查询比30号部门中所有职员享受更高薪水的职员。
    SELECT empno, ename, deptno   FROM emp
    WHERE deptno < > 30 AND sal > ALL ( SELECT sal FROM emp     //ALL:子查询结果前使用“谓词”以便关系运算
                                        WHERE deptno = 30) ;
20）  相关子查询（correlated subquery）
    查询薪水超过其所在部门平均薪水的所有职员。
     SELECT empno, ename, sal   FROM emp x
      WHERE sal > ( SELECT AVG (sal)
                    FROM emp y
                    WHERE  y.deptno = x.deptno) ;
     SQL聚集函数（aggregate functions），包括：AVG(), MAX(), MIN(), COUNT(), SUM()；
                                        聚集函数中的列名前还可使用谓词all或distinct。
     SQL还有标量函数（scalar functions），例如：NOW(), LEN(), UCASE(), LCASE()，等
```

三、查询结果分组

SELECT语句中，可使用GROUP BY子句对已选择的行进行分组，HAVING子句用于进一步选择已分的组，对每个已选中的组在查询结果中只返回其单行总计信息。

GROUP BY将结果表中的元组按指定列上值相等的原则进行分组，然后在每一分组上使用聚集函数，得到单一值。

HAVING对分组进行选择，只将聚集函数作用到满足条件的分组上。

若**GROUP BY**子句后有多个列名，则先根据第一列分组，再根据第二列分组...，一直分组下去。

使用**GROUP BY**子句进行分组后，可细化SQL函数的作用对象

未对查询结果分组，SQL函数将作用于整个查询结果

对查询结果分组后，SQL函数将分别作用于每个分组

使用**HAVING**子句筛选最终输出结果

只有满足**HAVING**子句中**<分组条件>**的组才会被输出

HAVING子句与WHERE子句的区别：作用对象不同！

WHERE <选择条件> 作用于基表或视图中的原始数据

HAVING <分组条件> 作用于查询结果分组中的数据

```
21）查询每个部门的薪水最大值、最小值和平均值。
      SELECT deptno, MAX(sal), MIN(sal), AVG(sal)
      FROM emp GROUP BY deptno;
22）查询整个公司的薪水最大值、最小值和平均值。
      SELECT MAX(sal), MIN(sal), AVG(sal)  FROM emp;
23）查询每个部门中“clerk” 人员的人数、平均薪水。
      SELECT deptno, COUNT(*), AVG(sal)  FROM emp
      WHERE  job = ‘clerk’
      GROUP BY deptno;
24）查询每个部门中“salesman”人员的最高薪水、最低薪水，要求最高薪水与最低薪水相差超过1000。
      SELECT deptno, MAX(sal), MIN(sal)  FROM emp
      WHERE job = ‘salesman’
```

```
        GROUP BY deptno HAVING MAX(sal) - MIN(sal) > 1000.0;
25) 查询每个部门中每个工种有多少职员。
        SELECT deptno, job,  COUNT(job)
         FROM emp
         GROUP BY deptno,  job;
       可能的查询结果是：
  DEPTNO       JOB            COUNT(JOB)
   11                  clerk       2
   11                manager       1
   13                manager       2
   13                analyst       4
   13                  clerk       3
   12                manager       1
   12                  clerk       1
   12                salesman      5
```

空值NULL在聚集函数计算中的作用

基表SC

S#	C#	G
s1	c1	80
s1	c2	90
s1	c3	85
s2	c1	85
s2	c2	NULL
s3	c2	NULL

```
SELECT   sum(G)
FROM  SC        结果340

SELECT  count(G)
FROM  SC        结果4

SELECT  count(distinct G)
FROM  SC        结果3

SELECT  count(*)
FROM  SC        结果6
```

四、查询结果排序

对查询结果排序
使用ORDER BY子句

ORDER BY <列标识> | <序号> [ASC | DESC]
[, <列标识> | <序号> [ASC | DESC]...]

可以按一个或多个属性列排序
升序（默认）：ASC；降序：DESC；
排序时遵循“**NULL值最大**”原则
ASC：列为**NULL**值的元组最后显示
DESC：列为**NULL**值的元组最先显示

```
26) 查询每个部门中每个工种有多少职员，要求查询结果按deptno升序、job降序输出。
      SELECT deptno,  job,  COUNT(job)    FROM emp
      GROUP BY deptno,  job
```

```
      ORDER BY deptno, 2 DESC ;              //ORDER BY中的序号2可代替SELECT中的第2个列标识
      可能的结果是：              //先按deptno的升序排列，再按job的降序排列
DEPTNO       JOB         COUNT(JOB)
11                manager     1
11                  clerk     2
12                salesman    5
12                manager     1
12                 clerk      1
13               manager      2
13                 clerk      3
13                Analyst     4
```

五、集合操作查询

标准SQL直接支持的集合操作：
并操作（UNION）

一般商用数据库支持的集合操作：
并操作（UNION）
交操作（INTERSECT）
差操作（MINUS）

这些运算符可联合多个**SELECT**查询，括号可改变缺省的运算次序。

并操作（UNION）

```
形式：<查询> UNION <查询>
要求并兼容：参加UNION操作的各结果关系（表）的属性数必须相同，且相应的属性域也必须相同

27）列出所有老销售人员及刚刚雇用的新销售人员名单。
            SELECT empno, ename  FROM emp
            WHERE job = ‘salesman’
            UNION
            SELECT empno, ename  FROM new_salesman ;

查询语句的逻辑处理顺序：
    “合并”/“连接”FROM子句中表（或视图）中的元组
    利用WHERE子句中的<选择条件>来选择元组，丢弃不满足<选择条件>的元组
    根据GROUP BY子句对保留下来的元组进行分组
    利用HAVING子句中的<分组条件>来选择元组分组，丢弃不满足<分组条件>的元组分组
    根据SELECT子句中的<列表达式>进行计算（注：含聚集函数的计算，如SUM() ），生成结果关系中的元组
    根据ORDER BY子句对查询结果进行排序输出
```

3.5 SQL数据操纵语言

SQL数据操纵语言（DML）使用插入（INSERT）、删除（DELETE）、修改（UPDATE）语句来更新数据，语法格式见表3-6

表 3-6　SQL 数据操纵语句

语句关键字	语法格式	说　明
INSERT	INSERT INTO <表名> [(<属性名> [{, <属性名>}])] {VALUES (<常量> [{, <常量>}]) | <查询语句>};	向基表中插入数据。属性列的顺序可与基表定义中的顺序不一致；属性列表可以被省略。
DELETE	DELETE [<表创建者名>.] <表名> [WHERE <删除条件>];	从基表中删除数据。若 WHERE 子句缺省，则删除基表中所有元组，但基表仍作为一个空表存在于数据库中。
UPDATE	UPDATE [<表创建者名>.]<表名> SET <属性名＝表达式> [{, <属性名＝表达式>}] [WHERE <更新条件>];	根据更新条件，将基表中相应元祖的属性值更新为表达式的值。

一、插入数据

单行直接插入：一次插入一个元组

语句格式：INSERT INTO <表名> [(<属性名1> [, <属性名2 >] ...)]
　　VALUES (<常量1> [, <常量2>] ...);

属性名的顺序可与表定义中的顺序不一致。
属性表可以被省略。在此情况下，表示要插入的是一个完整元组，各属性的排列顺序采用基表定义中的排列顺序。
常量列表表示被插入的常量元组值。其中属性值的数量及其排列顺序必须与INTO子句的属性表中一致。
插入的属性值可为NULL（除非已定义非空完整性约束）。

功能：将给定属性值的一个新元组插入到指定基表中。

28）在dept表中增加一个新部门。

```
INSERT INTO dept
 VALUES  (14, 'production', 'Nanjing');
```

29）在emp表中增加一个销售员298，部门为14，部门经理为158。

```
INSERT INTO emp (empno, ename, job, mgr, deptno)
VALUES (298, 'Li Si', 'salesman', 158, 14);
```

注：新插入的元组在sal和comm属性列上均取NULL值

等价于：

```
INSERT INTO  emp
VALUES (298, 'Li Si', 'salesman', 158, NULL, NULL, 14);
```

多行间接插入：一次插入多个元组

语句格式：INSERT INTO <表名> [(<属性1> [, <属性2>] ...)]
　　<SELECT查询> ;

功能：将SELECT查询结果插入指定基表中

30）（多行间接插入）将emp表中manager员工或佣金超过其薪水50%的员工的数据拷贝到bonus表中。

```
INSERT INTO bonus (e-name, work, salary, comm)
   SELECT ename, job, sal, comm
   FROM emp
   WHERE job = 'manager'  OR  comm > 0.5*sal ;
```

注：商用DBMS一般还提供非SQL手段的批量数据插入工具。
e.g. Oracle SQL*Loader可将文本文件、Excel数据批量插入基表

二、修改数据

语句格式

UPDATE <表名>
SET <列名> = <表达式> [, <列名> = <表达式>] ...
[WHERE <更新条件>];

功能
修改指定基表中满足WHERE子句中<更新条件>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。

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

31）（全部修改） 将emp表中所有员工的佣金置为NULL。

```
UPDATE emp  SET comm = NULL ;
```

32）（条件修改）将Jones提升为14号部门的经理，其薪水增加2000。

```
UPDATE emp  SET job = 'manager',  sal = sal+2000.0,  deptno = 14
WHERE ename = 'Jones' ;
```

33）（复杂修改）公司撤消在西安和武汉的部门，其员工全部调入公司总部，职务不变，薪水和佣金分别是原部门平均值的1.1倍和1.5倍。

```
UPDATE  emp x
SET (sal, comm) = ( SELECT 1.1*AVG(sal), 1.5*AVG(comm) FROM emp y
                            WHERE y.deptno = x.deptno ) ,
                deptno = ( SELECT deptno FROM dept
                                  WHERE dname = 'headquarters' )
WHERE  x.deptno IN ( SELECT deptno FROM dept
                                 WHERE loc IN ( 'Xian' , 'Wuhan' ) ;
```

三、删除数据

语句格式

DELETE FROM <表名> [WHERE <删除条件>];

功能
删除指定基表中满足WHERE子句中<删除条件>的元组
没有WHERE子句：**表示要删除基表中所有元组，但基表本身仍作为一个空表存在于数据库中。**

34）（全部删除）删除emp表中全部数据。

```
DELETE FROM emp ;
```

注：请区别DROP TABLE语句。

35）（条件删除） 删除 emp表中无佣金及佣金低于500的salesman数据

```
DELETE FROM emp
WHERE job = 'salesman' AND (comm IS NULL OR comm < 500.0) ;
```

注：用户使用INSERT、UPDATE、DELECT 语句进行数据更新的过程中，可能会违反已在相关基表上定义的完整性约束，因此，DBMS在执行数据更新语句时会检查并维护完整性约束！

3.6 SQL中的视图

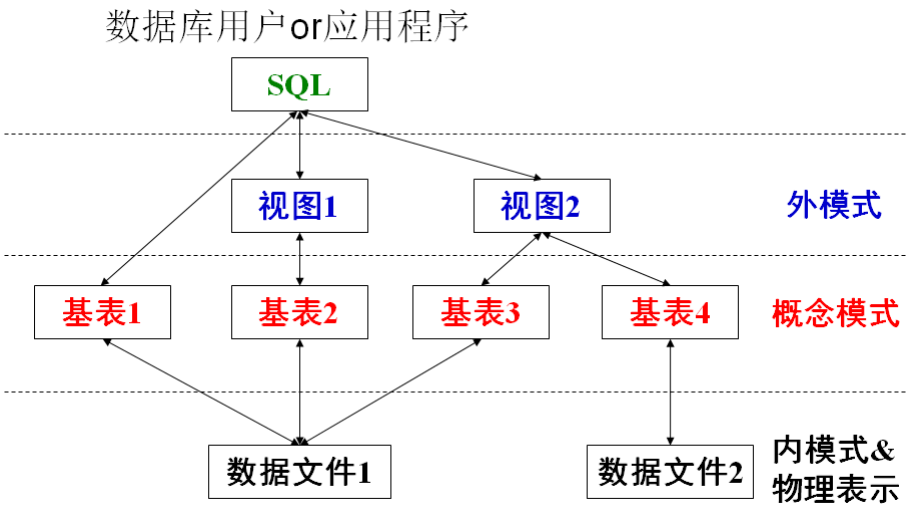
一、视图的概念

1.视图（view）

视图是由其他表（基表/视图）导出的虚表（virtual table），又称为导出表（derived table），可用于定义外模式

2.视图与基表（base table）的区别与联系

视图如同基表一样，是由行、列组成的二维表，在其中可查询数据、**有限制地更新数据**
视图中不直接包含数据（即不对应物理数据文件），其数据包含在导出它的基表中
基表中的数据发生了变化，由该基表定义的视图中的数据也会随之变化
数据库中只存放视图的定义，不会因为定义了视图而出现数据冗余
当用户在视图上的查询时，DBMS会根据视图定义将视图上的查询转换为用于定义视图的有关基表上的查询——**视图消解**



3.视图的作用

1）视图能够**简化**用户的查询操作
将多个基表进行连接后定义视图
利用基表上复杂的（嵌套）查询来定义视图
将基表中属性值进行计算后定义视图
2）视图使用户能以**多种角度**看待同一数据
不同的视图能够使不同用户以不同方式看待同一数据，满足数据库的共享需要
3）视图对数据库提供了一定程度的逻辑独立性
4）视图能够对机密数据提供一定程度的安全保护
对不同用户定义不同视图，使每个用户只能看到其该看到的那部分数据

二、定义与撤销视图

1.创建视图

CREATE VIEW <视图名> [(<属性名1> [, <属性名2>]...)]
AS <SELECT查询>;

功能：创建一个给定<视图名>（及其属性名列表）的视图，查询结果"看成"是视图中的元组，但DBMS并不真正执行该查询，只是将视图的定义（作为元数据）存储于数据字典
视图的**属性名列表**中各个属性必须分别与SELECT子句中的各个属性在属性值的语法与语义上相容；若省略视图的**属性名列表**，则直接用SELECT子句中的全部属性名作为该视图的属性名
需要明确指定视图的属性名的情形：

SELECT子句中的查询目标是SQL函数或列表达式
需要在视图中为基表中属性启用新的、更合适的属性名

```
36）创建全体员工工作所在地的视图。
       CREATE VIEW emp_loc
        AS  SELECT empno, ename, loc
               FROM emp, dept
               WHERE emp.deptno = dept.deptno ;

37）创建11号部门全体员工年薪视图。
      CREATE VIEW emp_ann_sal (eno, ename, ann_sal)
       AS  SELECT empno, ename, 12*sal
              FROM emp
              WHERE deptno = 11 ;
```

2.撤销视图

DROP VIEW <视图名>;

该语句从数据字典中删除指定的视图定义
通过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
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：DROP VIEW emp_loc ;

三、视图上查询数据

对用户而言：如同在基表中查询数据一样，使用SELECT语句
对系统而言：DBMS需进行**视图消解**（resolution）：

从数据字典中取出视图定义
视图上查询转换成基表上查询
执行基表上查询，返回查询结果

- 查询视图例子
  - 38) 查询在南京工作的全体员工名单。

```
SELECT ename
FROM emp_loc
WHERE loc = 'Nanjing';
```

DD中的视图定义

```
CREATE VIEW emp_loc
AS SELECT empno, ename, loc
     FROM emp, dept
     WHERE emp.deptno = dept.deptno;
```

消解

```
SELECT ename
FROM emp, dept
WHERE loc = 'Nanjing' AND emp.deptno = dept.deptno;
```

  - 注：使用“列表达式”或“GROUP BY/聚集函数”定义的视图，其消解过程很麻烦。DBMS会先将那样的视图形成一个临时表，然后在临时表上执行查询。

四、视图上更新数据

- 视图上更新就不象查询那样容易消解了。
  - 39) 假如创建了部门平均薪水的视图。

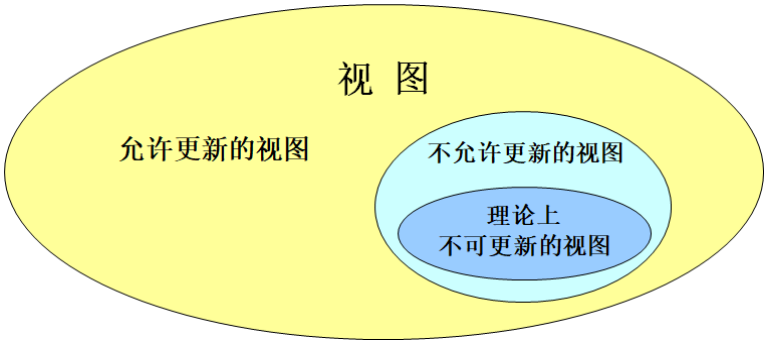
```
CREATE VIEW  empAvgSal  (deptno, avg_sal)
 AS   SELECT deptno, AVG(sal)
      FROM   emp
      GROUP BY deptno;
```

  - 以下视图上更新就无法消解：

```
UPDATE empAvgSal
SET avg_sal = avg_sal + 1000.0
WHERE deptno = 13 ;
```

  - 因此，视图上更新必须有所限制！

一般地，严格规定：含主键的单表行列子集视图是允许更新的



注：SQL:1999扩大了可更新视图（updatable view）的范围 ——取决于函数依赖（functional dependency）！

3.7 嵌入式SQL与SQL过程化扩充（简介）

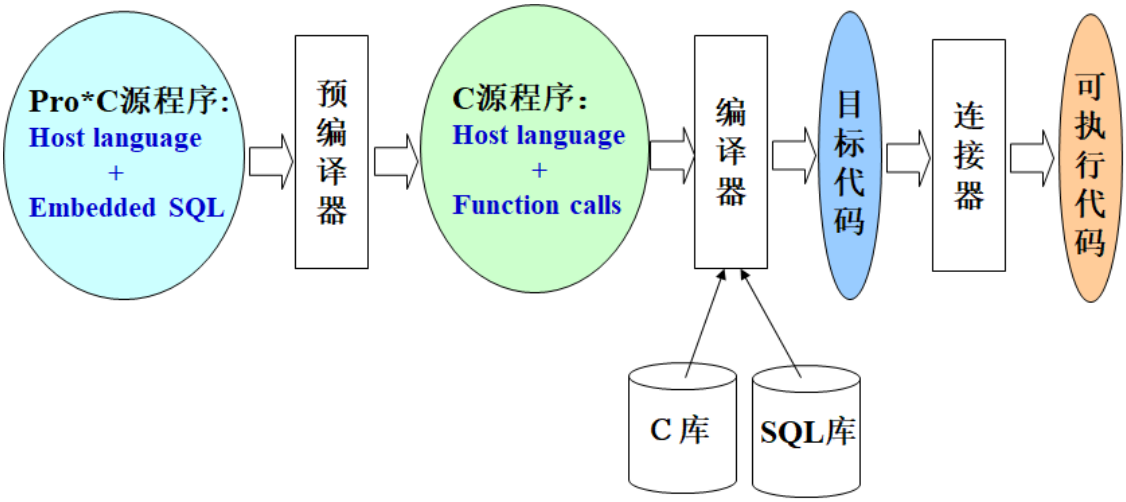
交互式SQL（Interactive SQL）是计算不完备的、非过程化数据库语言。然而，开发一个完整的数据库应用程序时，往往需要数据“管理”与数据“计算”的集成、数据库操作与其他处理（如流程控制）的集成。因此，就有**嵌入式SQL（embedded SQL）和SQL过程化扩充（procedural extensions）**

一、嵌入式SQL

将SQL语句嵌入到宿主语言（e.g. C, FORTRAN）程序中，进行混合编程。SQL负责“数据库操作”， 宿主语言负责“其他处理”。e.g. Oracle Pro*C, Pro*FORTRAN
然而，需解决以下几个问题：

1、如何将包含外语（SQL）的宿主语言源程序编译、连接成可执行代码?

借助预编译器



2、预编译器如何识别SQL语句?

SQL语句前加特殊标志：
EXEC SQL <SQL语句>
例：EXEC SQL DROP TABLE emp;

3、DBMS如何识别SQL语句中的宿主语言变量?

宿主语言变量前加标志（冒号）：

4、程序如何访问数据库?

合法的数据库用户，通过CONNECT语句连接到数据库：

EXEC SQL CONNECT :username INDENTIFIED BY :password ;

5、程序工作单元与数据库工作单元之间如何通讯?

通过SQL通讯区SQLCA

例如： Oracle的《 DBA手册》中说明SQLCA 包含如下信息：

The SQLCA contains run-time information about the execution of SQL statements, such as:

Oracle error codes
warning flags
event information
rows-processed count
diagnostics

6、程序中如何逐行处理SELECT返回的多行结果?

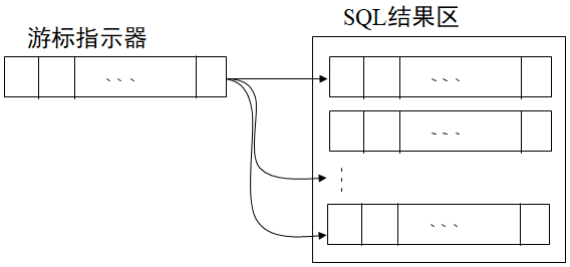
通过游标（cursor）机制（扩充的SQL语句 EXEC SQL...）：

定义游标：DECLARE <游标名> CURSOR FOR <SELECT语句> ;

打开游标：OPEN <游标名> ;

逐行取数：FETCH <游标名> INTO <主变量列表>;

关闭游标：CLOSE <游标名> ;



二、SQL过程化扩充

为弥补SQL在流程控制方面的不足， 从SQL-92标准开始，增加了一个关于存储过程的补充标准：□SQL-92/PSM (Persistent Stored Modules)

后续SQL标准版本中直接将SQL/PSM作为SQL的一部分（Part 4: SQL/PSM）

SQL/PSM主要包括**过程化结构**（主要语句见表3-9）、**存储过程**与**函数**

存储过程是指使用CREATE PROCEDURE语句事先定义好的过程，经编译后存储在DBMS中，供应用调用

函数由CREATE FUNCTION语句定义

表 3-9 持久存储模块（PSM）中的主要语句类型

语句类型	语句说明
赋值语句	使用“=”运算将一个 SQL 值表达式的结果赋值到一个局部变量、表列或 UDT 属性。
条件语句	使用 IF 语句根据条件选择动作的执行。
选择语句	使用 CASE 语句进行多条件选择。
循环语句	使用 FOR、WHILE、REPEAT 三种语句来定义可根据条件重复执行的一组 SQL 语句。
调用语句	使用 CALL 语句进行过程调用；使用 RETURN 语句指定 SQL 函数或方法的返回值。
条件处理	使用 DECLARE…HANDLER/CONDITION 和 SIGNAL/RESIGNAL 语句来定义例外或完成条件及其后处理动作。