# Oracle数据库基础知识

## 数据库权限

权限（Pirvilege）是指执行特定类型SQL命令或访问其他方案对象的权利。角色（Role）是权限管理的一种解决方案，是一组相关权限的集合。用户（User）是能够访问数据库的人员。用户权限可以直接或间接的被授予，用户的权限信息被保存在数据字典中。

按照权限所针对的控制对象，可以将权限分为**系统权限**和**对象权限**两种。

系统权限（System Privilege）：是指在系统级控制数据库的存取和使用的机制，即执行特定SQL命令的权利，它用于控制用户可以执行的一个或一组数据库操作。这些权限完全不涉及对象，而是设计运行批处理作业、改变系统参数、创建角色、甚至是连接到数据库自身等方面。可以将系统权限授予用户、角色和公共用户组。

对象权限（Object Privilege）：是指在系统级控制数据库的存取和使用的机制，即访问其他方案对象的权利，它用于控制用户对其他方案对象的访问。用户可以直接访问其方案对象，但如果要访问其他方案的对象，则必须要具有对象权限。对象权限是用户之间的表、视图、序列等模式对象的相互存取操作的权限。对属于某一用户模式的所有模式对象，该用户对这些模式对象具有全部的对象权限，也就是说，模式的拥有者对模式中对象具有全部对象权限。同时模式的拥有者还可以将这些对象权限授予其他用户。

在Oracle数据库中，用户SYSTEM和SYS是数据库管理员，它具有DBA所有系统权限，包括SELECT ANY DICTIONARY权限，所以SYSTEM和SYS用户可以查询数据字典以“DBA\_”开头的数据字典视图、创建数据库结构等。

Oracle数据库系统中的用户包括最终用户（即连接、登录和操作数据库人员）、应用程序开发人员（使用Oracle数据库进行应用程序开发的人员）、数据库管理员（DBA）等，其中最高的权限属于SYSDBA。

SYSDBA具有控制Oracle一切行为的特权，如创建、启动、关闭、恢复数据库等。使数据库归档/非归档，备份表空间等关键性动作只能通过具有SYSDBA权限的用户来执行。这些任务即使是普通DBA角色也不行。

在Oracle11g中，有206个系统权限，可以在数据字典表**SYSTEM\_PRIVILEGE\_MAP**中看到所有这些权限：

SQL> CONNECT SYS/密码 ASSYSDBA;

SQL> select \* from SYSTEM\_PRIVILEGE\_MAP where NAME like '%CREATE%'; --找出和CREATE有关的系统权限

注意：系统权限中有一种ANY权限，具有ANY权限的用户可以在任何用户模式中进行操作。

将权限授予用户包括**直接授权**和**间接授权**两种方式。其中，直接授权是直接把权限授予用户；而间接授权是先把权限授予角色，再将角色授予用户。

直接授权（指通过GRANT语句授权）：

（1）系统权限授权

创建用户后，如果没有给用户授予相应的系统权限，则用户不能连接到数据库，因为该用户缺少创建会话的权限。系统权限是由数据库管理原为用户授予的：

GRANT系统权限TO {PUBLIC | role | username} [WITH ADMIN OPTION]

添加WITH ADMIN OPTION选项表示该用户可以将其所有权限再授予其他用户，并且还可以将授予的权限再收回。

【例1.1】给已经创建的ACD用户授予SYSDBA系统权限

SQL> CONNECT SYS/密码 AS SYSDBA;

SQL> GRANT SYSDBA TO ACD;

SQL> CONNECT ACD/密码 AS SYSDBA;

【例1.2】创建一个ACD1用户，使其具有登录、连接的系统权限。

SQL> CONNECT SYS/密码 AS SYSDBA;

SQL> CREATE USER ACD1 IDENTIFIED BY ACD1

2 DEFAULT TABLESPACE user

3 TEMPORARY TABLESPACE temp；

SQL> GRANT CREATE SESSION TO ACD1;

SQL> CONNECT ACD1/ACD2;

（2）系统权限授权

按照不同的对象类型，Oracle数据库中设置了不同种类的对象权限。对象权限即对象之间的对应关系如下表所示：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ALTER | DELETE | EXECUTE | INDEX | INSERT | READ | REFERENCE | SELECT | UPDATE |
| DIRECTORY |  |  |  |  |  | √ |  |  |  |
| FUNCTION |  |  | √ |  |  |  |  |  |  |
| PROCEDURE |  |  | √ |  |  |  |  |  |  |
| PACKAGE |  |  | √ |  |  |  |  |  |  |
| SEQUENCE | √ |  |  |  |  |  |  | √ |  |
| TABLE | √ | √ |  | √ | √ |  | √ | √ | √ |
| VIEW |  | √ |  |  | √ |  |  | √ | √ |

其中画“√”表示某种对象所具有的对象权限，空就表示该对象没有某种权限。

对象权限只能由该对象的拥有者为其他用户授权。对象权限被授出后，获权用户才可以对对象进行相应的操作。对象权限被授出后，对象的拥有者属性和存储属性也不会改变。

使用GRANT语句可以将对象权限授予指定的用户、角色、PUBLIC公共用户组：

GRANT [对象权限| ALL[PRIVILEGES]] ON [schema.]object TO [user | role | PUBLIC]

其中“对象权限”是上面中某一类对象的相应权限，多个权限用逗号隔开，多个用户名也用逗号隔开，角色表示数据库中已经创建的角色。PUBLIC表示将该对象权限授予数据库中全体用户。

【例1.3】用户HR将EMPLOYEES表的查询、插入、更改表的对象权限授予ACD用户。

SQL> CONNECT HR/HR;

SQL> GRANT SELECT, INSERT, UPDATE ON employees TO acd；

SQL> CONNECT ACD/ACD;

授权角色：

为了简化权限管理，Oracle引入了角色概念，角色是相关权限的命名集合，使用角色的主要目的是为了简化权限管理。权限、角色不仅可以被授予用户，也可以被授予用户组（PUBLIC）。当将权限或角色授予public之后，会使所有用户都具有该权限或角色。

在创建角色时，可以为角色设置应用安全性。角色应用安全性是通过为角色设置密码进行保护的，只有提供正确的密码才能允许修改或设置角色。

通过查询数据字典**DBA\_ROLES**可以了解数据库中全部角色信息：

SQL> CONNECT SYS/密码 AS SYSDBA;

SQL> SELECT role, password FROM dba\_roles;

**DBA角色拥有所有系统级权限。**通常，角色CONNECT、RESOURCE和DBA主要用于数据库管理所以对于数据库管理员需要分别授予CONNECT、RESOURCE和DBA角色。用于数据库开房用户需要分别授予CONNECT和RESOURCE角色。

在角色创建时，它并不具有任何权限。

【例1.4】创建一个名为ACCESS\_DATABASE的角色，并为它授予一些对象权限和系统权限，并将该角色授予用户ACD。

SQL> CREATE ROLE access\_database;

SQL> GRANTCREATE SESSION, CREATE TABLE, CREATE VIEW TO access\_database;

SQL> GRANT access\_database TO acd;

**注意：在一个GRANT语句中，可以同时为用户授予系统权限和角色，但不能同时授予对象权限和角色。**

Oracle数据库中，数据库管理员通常是通过角色来管理系统权限，即将角色授予用户，而不是直接为用户授予系统权限。

将角色授予用户后，角色信息被存储在该用户的数据字典**USER\_ROLE\_PRIVS**中，通过查询该数据字典可以了解该用户所具有的角色：

SQL> SELECT username, granted\_role, admin\_option FROM user\_role\_privs;

注意，所有用户的角色都放在了**DBA\_ROLE\_PRIVS**中，只有DBA角色的用户才能查看。

**默认角色**是当用户登录到数据库时由Oracle自动启用的一种角色。当某一角色被授予用户后，该角色即称为该用户的默认角色。可以使用ALTER USER语句来修改用户的默认角色:

ALTER USER username [defaultrolename] | ALL [EXCEPT rolename[, rolename1, rolename2] ] | NONE];

其中，defaultrolename表示默认角色；使用关键字ALL可以设置该用户的所有角色；使用EXCEPT则可以设置某些角色外其他所有角色生效；NONE则设置所有角色为失效状态。

要使用户的角色失效，可用如下语句：

ALTER USER usernameDEFAULT ROLE NONE;

使用户的角色重新生效，可用如下语句：

ALTER USER usernameDEFAULT ROLE ALL;

启用用户某个角色之外的其他所有角色，可用如下语句：

ALTER USER usernameDEFAULT ROLE ALL EXCEPT;

可以为数据库用户的会话启用或禁用角色。如果数据库管理员没有为用户取消所有默认角色，则该用户的会话将启用所有已经授予的角色。可以通过查询数据字典视图SESSION\_ROLES，查看当前数据库会话启用了哪些角色。

可以使用SET ROLE语句控制角色生效或失效。

SETROLE[role [IDENTIFIED BY password]] [, role [IDENTIFIED BY password]…] | ALL[EXCET role [, role] | NONE];

其中，使用带ALL选项的SET ROLE 语句时，将启用用户被授予的所有角色，使用ALL选项有一个前提条件是该用户的所有角色不得设置密码。EXCEPT ROLE表示除指定的角色外，启用其他全部角色。NONE表示使用户的所有角色失效。注意，该语句只会改变SESSION\_ROLES中的数据，对USER\_ROLE\_PRIVS中的数据不会造成影响。当该用户重新连接时，SESSION\_ROLES中的数据会回到原始状态。

【例1.5】演示启用/禁用角色

SQL> CONNECT SYS/密码 AS SYSDBA;

SQL> ALTER USER acd DEFAULT ROLE NONE; --使ACD用户没有默认角色

SQL> GRANT CONNECT, RESOURCE, DBA TO acd; --给ACD用户授予新的角色

SQL> ALTER USERacd DEFAULT ROLE CONNECT, RESOURCE;--设置用户默认角色为CONNECT和RESOURCE

SQL> SELECT \* FROM dba\_role\_privs WHERE GRANTEE=’MANZHIZHEN’;

SQL> CONNECT ACD/ACD;

SQL> SELECT \* FROM user\_role\_privs; --显示结果和dba\_role\_privs中该用户的角色一致

SQL> SELECT \* FROM session\_roles; --会发现只有CONNECT和RESOURCE，没有DBA，因为DBA已经不属于默认角色了

SQL> SET ROLE DBA; --在会话中强制对用户启用DBA角色（只有该用户拥有DBA角色才能这样做）。

SQL> SELECT \* FROM session\_roles;

SQL> SET ROLE NONE; --使该用户所有角色失效；

SQL> SELECT \* FROM session\_roles;

SQL> SET ROLE ALL; --使该用户所有角色生效；

SQL> SELECT \* FROM session\_roles;

可以用REVOKE语句逐一收回权限。

系统权限回收：

数据库管理员或者具备向其他用户授权的用户都可以使用REVOKE语句将授予的权限回收：

REVOKE 系统权限FROM{ PUBLIC | role | user};

【例1.6】使用SYS来收回SCOTT用户的SELECT ANY DICTIONARY系统权限。

SQL> CONNECT SYS/密码 AS SYSDBA;

SQL> REVOKE SELECT ANY DICTIONARY FROM scott;

用户的权限被回收后，相应的权限传递同时被回收。在回收系统权限时，经过传递获得的权限的用户不受影响。假设用户A将系统权限a授予了用户B，用户B又将系统权限a授予了用户C。那么当删除用户B后或者从用户B回收系统权限a后，用户C仍然保留这系统权限a。

对象权限回收：

对象的拥有者可以将授出的权限收回：

REVOKE {objectprivilege | ALL [PRIVILEGES]} ON [schema.]object FROM{ PUBLIC | role | user};

回收对象权限时，授权者只能从自己授权的那些用户那里回收对象权限。如果被授权用户基于一个对象权限创建了过程、视图，那么当回收该对象权限后，这些过程、视图将变为无效。

【例1.7】HR用户回收acd对employees表的SELECT对象权限。

SQL> CONNECT HR/HR;

SQL> REVOKE SELECT ON employees FROM acd;

【例1.8】HR用户将基表employees的所有权限从PUBLIC用户回收。

SQL> CONNECT HR/HR;

SQL> REVOKE ALL ON employees FROM public;

在回收对象权限时，经过传递获得权限的用户会受到影响。假设用户A将对象权限a授予了用户B，用户B又将对象权限a授予了用户C。那么当删除用户B后或者从用户B回收对象权限a后，用户C将不再具有该对象权限a，并且用户B和C中与该对象权限有关的对象都变成无效。

可以使用DROP ROLE来删除角色，使用该角色的用户的权限同时也会被回收。

【例1.9】删除角色ACCESS\_DATABASE。

SQL> CONNECT SYSTEM/密码 AS SYSDBA;

SQL> DROP ROLE access\_database;

删除数据库对象，Oracle数据库对象中最基本的是表和视图，其他还有约束、序列、函数、存储过程、包、触发器、索引等。删除数据库对象后，也就删除了用户或角色对该数据库对象的访问权限。

删除表：DRPP TABLE [schema.]tablename [CASCADE CONSTRAINTS];

删除视图：DRPP VIEW viewname;

需要注意的是，将视图定义从数据字典中删除，基于视图的权限也同时被删除，其他涉及到该视图的函数、视图、程序等都将被视为非法。

当删除一个用户时，系统会将该用户帐号以及用户模式的信息从数据字典中删除。用户被删除后，用户创建的所有数据库对象也被全部删除。删除用户可以使用DROP USER语句。如果用户当前正在连接到数据库，则不能删除该用户，必须等到该用户退出再删除。如果要删除的用户模式中包含有模式对象，则必须在DROP USER 语句中带上CASCADE关键字，将该用户创建的模式对象全部删除。

## 数据库用户管理

可以使用企业管理器创建用户，也可以使用SQL语句创建用户：

**CREATE USER** username **IDENTIFIED BY** password

**DEFAULT TABLESPACE** tablespacename --默认表空间

**TEMPORARY TABLESPACE** tablespacename; --默认的临时的表空间

**CREATE USER** username **IDENTIFIED BY** password

**DEFAULT TABLESPACE** tablespacename --默认表空间

**TEMPORARY TABLESPACE** tablespacename --默认的临时的表空间

**QUOTA** num **ON** tablespacename; --用户可以使用表空间的字节数，如果设置num为0，表示不能使用该表空间

**CREATE USER** username **IDENTIFIED BY** password

**DEFAULT TABLESPACE** tablespacename --默认表空间

**TEMPORARY TABLESPACE** tablespacename --默认的临时的表空间

**QUOTA UNLIMITED ON** tablespacename; --用户可以使用tablespacename表空间不受限制

**CREATE USER** pcname **IDENTIFIEDEXTERNALLY** --表示用户名在操作系统下验证，这个用户名必须于操作系统中定义的相同。

**DEFAULT TABLESPACE** tablespacename --默认表空间

**TEMPORARY TABLESPACE** tablespacename --默认的临时的表空间

**PASSWORD EXPIRE** --设置口令过期状态，用户再次登录进入前必须输入密码

**ACCOUNT LOCK;** --用户是否被加锁，默认是不加锁的

## 数据库空间管理

在数据库中，除了用户可以直接使用的数据外，还有另一种数据，它们是有关数据库的定义信息，如数据库的名称、数据表的定义、数据库账户、权限等，这些数据通常存放在一个“**数据字典**（data dictionary）”中。数据字典是数据库管理系统工作的依据，它是有数据库管理系统自动生成并维护的一组表和视图。在Oracle数据库中，数据字典存放在SYSTEM表空间中。

## 索引

在数据库中，**索引**是除表之外最重要的数据对象，其功能是提高对数据表的检索效率。数据库的索引类似书的目录。索引使DML操作能迅速的找到表中的数据，而不必扫描整个表。索引是数据和存储位置的列表。

索引是建立在表上的、可选的数据对象。索引通过事先保存的、排序后的索引键取代默认的全表扫描检索方式。在一个表上是否创建索引、创建多少个索引、创建什么类型的索引，都不会对表的使用方式产生任何影响。但是，通过在表中的一个列或多个列上创建索引，却能够为数据的检索提供快捷的存取路径、减少查询时的硬盘I/O操作、加快数据的检索速度。

索引是将创建列的键值和对应记录的物理记录号（ROWID）排序后存储起来，需要占用额外的存储空间来存放。它可以被存储在与表不同的磁盘或表空间中，有单独为其设立命名的存储结构，即索引段。

索引一旦被创建，在表上执行DML操作时Oracle就会自动对索引进行维护，并且由Oracle决定何时使用索引。用户完全不需要在SQL语句中指定使用哪个索引、如何使用索引等。

假设有如下电影关系表MOVES(ID, TITLE, STAR\_NAME, YEAR)，通常，如果多属性所以中的键是某些属性按特定顺序的组合，那么可以使用这个索引找出匹配属性列表中前面的任何属性子集的全部元祖。这样多属性索引的设计即是属性列表顺序的选择。例如，若查找时对电影名TITLE的查询比对年份YEAR的查询要多，就使用（TITLE,YEAR）来定义的多属性索引，若对电影年份的查询比対电影名的查询多，那么最好创建一个（YEAR,TITLE）上的索引。

在有关数据库的讨论中，磁盘页就是通常所说的磁盘块。

通常，关系(表)上最有用的索引时其主键上的索引，原因有两个：1.在查询中为主键指定值是比较普遍的。因此，主键上的索引通常会被频繁的使用。2.因为键值是唯一的，故与给定键值匹配的元组最多只有一个，因此索引返回的要么是这个元组的位置，要么什么也不返回。也就是说，为了取得这个元组，最多只有一个磁盘页需要被读入到主存（尽管有时为了使用索引本身还需要读入储存索引的相关磁盘页）。

当索引不是建立在主键上时，则在执行查询时，它可能会也可能不会加速元组的检索速度。存在两种情况，即使不是建立在主键属性上的索引也仍然有效：1.该属性几乎可以看成是一个主键，即相对来说基本上没有多少元组在该属性上具有给定值（相同值）。所以，即使每个具有给定值（相同值）的元组分别位于不同磁盘页上，也不需要检索大量的磁盘页。2.元组在该属性上市“聚集”的，即通过将具有该属性上公共值的元组分组到尽可能少的磁盘页里来将一个关系聚合到一个属性上。在这种情况下，即使符合要求的元组可能有很多，但是却不必检索与符合要求的元组数目相同的磁盘页。

如果更新是最频繁发生的操作，则对于索引的创建应该采取非常保守的策略，因为对于关系表R的更新操作都会迫使同时更新R所修改过的属性或者索引集上的任何索引。这样，不仅要读取和回写修改的R的页，还必须花费额外代价读取和回写储存索引的页。尽管有时更新操作是对数据库才去的主要操作，在频繁访问的属性上建立索引仍然可以获得性能改进。因为，实际上一些更新操作本身也包含了对数据库的查询操作（比如，带有select-from-where子查询的插入或者待条件的删除等操作）。

必须牢记，典型的关系被储存在很多磁盘块（页）上，而查询或更新操作的主要代价通常来自于将所需的磁盘页读入到主存的数目。因此，能够快速找到所需元组而不需要对整个关系表进行遍历的索引能节省大量的时间。然而，不幸的是，索引本身也需要（至少是部分地）被储存在磁盘上，访问索引和修改索引本身也需要进行磁盘访问。实际上，由于修改操作需要一次磁盘访问以读取磁盘页，而另一次磁盘访问用于将修改后的页写回磁盘，所以它的开销是查询中访问索引或数据的两倍。

由于Oracle有时也会利用索引，如唯一索引来实现一些完整性约束，因此在创建主键约束时会自动创建主键索引。

若要在自己的方案中创建索引，则需要具有CREATE INDEX系统权限；若要在其他用户的方案创建索引，则需要具有CREATE ANY INDEX系统权限。同理，如果要在自己的方案中删除索引，需要具有DROP INDEX 系统权限；如果要删除其他用户的方案中的索引，需要具有DROP ANY INDEX系统权限。

创建索引的语法：

**CREATE [UNIQUE] | [BITMAP] INDEX** [schema.]index\_name--不加BITMAP则默认是B树索引。

**ON** [schema.]table\_name([column1**[ASC | DESC]**, column2**[ASC | DESC]**, …] | [express])

**[TABLESPACE** tablespace\_name**]** --指定索引段所在的表空间。

**[PCTFREDD** n1**]** --设置将来的INSERT操作所预留的空间百分比。

**[STORAGE(INITIAL** n2**)]**

**[COMPRESS** n3**] | [NOCOMPRESS]**

**[ONLINE]**

**[COMPUTE STATISTICS]**

**[REVERSE] | [NOSORT];**

删除索引的语法：

**DROP INDEX** index\_name**;**

【例4.1】给EMPLOYEES表的列EMPLOYEE\_ID创建一个索引employee\_id\_index。

SQL> CONNECT ACD/ACD;

SQL> CREATE INDEX employee\_id\_index ON employees(employee\_id);

如果索引是通过CREATE INDEX 语句创建的，可以使用DROP INDEX 语句删除索引；如果索引是在定义约束时由Oracle自动建立的，则可以通过金庸约束（DISABLE）或删除约束的方式来删除定义的索引。

**注意，删除一个表时，基于该表的索引也会被自动删除。**

## 存储过程

## PL/SQL编程

## 游标

在PL/SQL块中执行SELECT、INSERT、UPDATE和DELETE语句时，Oracle会在内存中为

其分配上下文区（Context Area），即一个缓冲区。游标是指向该区的一个指针，或是命名一个工作区（Work Area），或是一种结构化数据类型。它为应用程序提供了一种对具有多行数据查询结果集中的每一行数据分别进行单独处理的方法，是设计嵌入式SQL语句的应用程序的常用编程方式。

游标分为**显式游标**和**隐式游标**两种。显式游标是由用户声明和操作的一种游标；隐式游标是Oracle为所有数据操纵语句（包括只返回单行数据的查询语句）自动声明和操作一种游标。

在每个用户会话中，可以同时打开多个游标，其数量由数据库初始化参数文件中的OPEN CURSORS参数定义。

显式游标的处理包括声明游标、打开游标、提取游标、关闭游标4个步骤。游标声明需要在块的声明部分进行，其他的3个步骤都在执行部分或异常处理中进行。

1. **声明游标**

对游标的声明定义了游标的名称并将该游标和一个SELECT语相关联。游标声明可以在WHERE子句中引用PL/SQL变量。这些变量被认为是**联编变量**bindVARIABLE，即已经被分配空间并映射到绝对地址的变量。

语法为：

**CURSOR** cursor\_name **IS SELECT** select\_sql;

需要注意的是，select\_sql语句不能包含INTO子句。INTO子句是FETCH语句（提取游标）的一部分。

【例7.1】声明游标举例。

DECLARE

employee\_id NUMBER(10); --定义4个变量来存放EMPLOYEES表中的内容

employee\_name VARCHAR2(50);

employee\_age NUMBER(3);

employee\_sex CHAR(1);

CURSOR employee\_cur IS --定义游标employee\_cur

SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, EMPLOYEE\_AGE, EMPLOYEE\_SEX

FROM EMPLOYEES WHERE EMPLOYEE\_AGE < 30;

1. **打开游标**

打开游标就是执行定义的SELECT语句。执行完毕，查询结果装入内存，游标停在查询结果的首部，注意，并不是第一行。当打开一个游标时，会完成以下几件事情：a.检查联编变量的取值。b.根据联编变量的取值，确定活动集。c.活动集的指针指向第一行。

打开游标语句：

**OPEN** cursor\_name;

【例7.2】打开游标举例。

。。。

BEGIN

OPEN employee\_cur; --打开游标

注意：打开一个已经被打开的游标是合法的。在第二次执行OPEN语句以前，PL/SQL将在重新打开该游标之前隐式地执行一条CLOSE语句。一次也可以同时打开多个游标。

1. **提取游标**

打开游标后的工作就是取值了。取值语句时FETCH。它的用法有两种形式，如下：

**FETCH** cursor\_name **INTO** var\_list;

或 **FETCH** cursor\_name **INTO** plsql\_record;

其中var\_list是已经声明的PL/SQL变量列表（变量之间用逗号隔开），plsql\_record是已经声明的PL/SQL记录。在这两种情形中，INTO子句中的变量的类型都必须与查询的选择列表类型相兼容，否则将拒绝执行。

【例7.3】提取游标举例。

。。。

BEGIN

OPEN employee\_cur; --打开游标

FETCH employee\_cur INTO employee\_id, employee\_name, employee\_age, employee\_sex;--将第一行数据放入变量中，游标后移

FETCH语句每执行一次，游标向后移动一行，直到结束（游标只能逐个向后移动，不能跳跃移动或向前移动）。

1. **关闭游标**

当所有的活动集都被检索以后，游标就应该被关闭。PL/SQL程序将被告知对于游标的处理已经结束，与游标相关联的资源可以被释放了。这些资源包括用来存储活动集的存储空间，以及用来存储活动集的临时空间。关闭游标的语法：

**CLOSE** cursor\_name;

注意，关闭已经关闭的游标将会报错。

【例7.4】对游标各种操作的完整示例。

DECLARE

employee\_id NUMBER(10); --定义4个变量来存放EMPLOYEES表中的内容

employee\_name VARCHAR2(50);

employee\_age NUMBER(3);

employee\_sex CHAR(1);

CURSOR employee\_cur IS --定义游标employee\_cur

SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, EMPLOYEE\_AGE, EMPLOYEE\_SEX

FROM EMPLOYEES WHERE EMPLOYEE\_AGE < 30;

BEGIN

OPEN employee\_cur; --打开游标

FETCH employee\_cur INTO employee\_id, employee\_name, employee\_age, employee\_sex;--将第一行数据放入变量中，游标后移

LOOP

EXIT WHEN NOT employee\_cur%FOUND; --如果游标到尾则结束

IF employee\_sex=’M’ THEN

INSERT INTO MAIL\_EMPLOYEE(EMPLOYEE\_ID, EMPLOYEE\_NAME)

VALUES(employee\_id, employee\_name);

ELSE

INSERT INTO FEMAIL\_EMPLOYEE(EMPLOYEE\_ID, EMPLOYEE\_NAME)

VALUES(employee\_id, employee\_name);

END IF;

FETCH employee\_cur INTO employee\_id, employee\_name, employee\_age, employee\_sex;

END LOOP;

CLOSE employee\_cur; --关闭游标

END;

上述执行的PL/SQL过程已经把数据分别插入到男员工和女员工表中。

使用显式游标时，需注意以下事项：

a.使用前必须用%ISOPNE检查其打开状态，只有此值为TRUE的游标才可以使用，否则要先将游标打开。

b.在使用游标过程中，每次都要用%FOUND或%NOTFOUND属性检查是否返回成功，即是否还有要操作的行。

使用游标也能进行修改和删除操作，但必须在游标定义时制定FOR子句后面的编辑类，如DELETE或UPDATE.

## 触发器

触发器是一种特殊的存储过程。一般的存储过程是通过存储过程名直接调用，而触发器主要是通过事件(增、删、改)进行触发而被执行的。其在表中数据发生变化时自动强制执行。主要实现对于数据库增、删、改引起事件的关联操作。

## 常用SQL语句