第八章　触发器

触发器是许多关系数据库系统都提供的一项技术。在 ORACLE 系统里，触发器类似过程和函数，都有声明，执行和异常处理过程的 PL/SQL 块。

§8.1　触发器类型

触发器在数据库里以独立的对象存储，它与存储过程不同的是，存储过程通过其它程序来启动运行或直接启动运行，而触发器是由一个事件来启动运行。即触发器是当某个事件发生时自动地隐式运行。并且，触发器不能接收参数。所以运行触发器就叫触发或点火（firing）。ORACLE 事件指的是对数据库的表进行的 INSERT、UPDATE 及 DELETE 操作或对视图进行类似的操作。ORACLE 将触发器的功能扩展到了触发 ORACLE，如数据库的启动与关闭等。

§8.1.1 DML 触发器

ORACLE 可以在 DML 语句进行触发，可以在 DML 操作前或操作后进行触发，并且可以对每个行或语句操作上进行触发。

§8.1.2 替代触发器

由于在 ORACLE 里，不能直接对由两个以上的表建立的视图进行操作。所以给出了替代触发器。它就是 ORACLE 8 专门为进行视图操作的一种处理方法。

§8.1.3 系统触发器

ORACLE 8i 提供了第三种类型的触发器叫系统触发器。它可以在 ORACLE 数据库系统的事件中进行触发，如 ORACLE 系统的启动与关闭等。

触发器组成:

- 触发事件：即在何种情况下触发 TRIGGER; 例如：INSERT, UPDATE, DELETE。
- 触发时间：即该 TRIGGER 是在触发事件发生之前（BEFORE）还是之后(AFTER)触发，也就是触发事件和该 TRIGGER 的操作顺序。
- 触发器本身：即该 TRIGGER 被触发之后的目的和意图，正是触发器本身要做的事情。例如：PL/SQL 块。
- 触发频率：说明触发器内定义的动作被执行的次数。即语句级(STATEMENT)触发器和行级(ROW)触发器。
  语句级(STATEMENT)触发器：是指当某触发事件发生时，该触发器只执行一次；
  行级(ROW)触发器：是指当某触发事件发生时，对受到该操作影响的每一行数据，触发器都单独执行一次。

§8.2 创建触发器

创建触发器的一般语法是:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   {BEFORE | AFTER }
   {INSERT | DELETE | UPDATE [OF column [, column …]]}
   ON [schema.] table_name
   [REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]
   [FOR EACH ROW ]
   [WHEN condition]
   trigger_body;
```

其中：

BEFORE 和 AFTER 指出触发器的触发时序分别为前触发和后触发方式，前触发是在执行触发事件之前触发当前所创建的触发器，后触发是在执行触发事件之后触发当前所创建的触发器。

FOR EACH ROW 选项说明触发器为行触发器。行触发器和语句触发器的区别表现在：行触发器要求当一个 DML 语句操作影响数据库中的多行数据时，对于其中的每个数据行，只要它们符合触发约束条件，均激活一次触发器；而语句触发器将整个语句操作作为触发事件，当它符合约束条件时，激活一次触发器。当省略 FOR EACH ROW 选项时，BEFORE 和 AFTER 触发器为语句触发器，而 INSTEAD OF 触发器则为行触发器。

REFERENCING 子句说明相关名称，在行触发器的 PL/SQL 块和 WHEN 子句中可以使用相关名称参照当前的新、旧列值，默认的相关名称分别为 OLD 和 NEW。触发器的 PL/SQL 块中应用相关名称时，必须在它们之前加冒号(:)，但在 WHEN 子句中则不能加冒号。

WHEN 子句说明触发约束条件。Condition 为一个逻辑表达时，其中必须包含相关名称，而不能包含查询语句，也不能调用 PL/SQL 函数。WHEN 子句指定的触发约束条件只能用在 BEFORE 和 AFTER 行触发器中，不能用在 INSTEAD OF 行触发器和其它类型的触发器中。

当一个基表被修改(INSERT, UPDATE, DELETE)时要执行的存储过程，执行时根据其所依附的基表改动而自动触发，因此与应用程序无关，用数据库触发器可以保证数据的一致性和完整性。

每张表最多可建立 12 种类型的触发器，它们是:

BEFORE INSERT
BEFORE INSERT FOR EACH ROW
AFTER INSERT
AFTER INSERT FOR EACH ROW

BEFORE UPDATE
BEFORE UPDATE FOR EACH ROW

AFTER UPDATE
AFTER UPDATE FOR EACH ROW

BEFORE DELETE
BEFORE DELETE FOR EACH ROW
AFTER DELETE
AFTER DELETE FOR EACH ROW

§8.2.1 触发器触发次序

1. 执行 BEFORE 语句级触发器;
2. 对与受语句影响的每一行：
- 执行 BEFORE 行级触发器
- 执行 DML 语句
- 执行 AFTER 行级触发器
3. 执行 AFTER 语句级触发器

§8.2.2 创建 DML 触发器

触发器名与过程名和包的名字不一样，它是单独的名字空间，因而触发器名可以和表或过程有相同的名字，但在一个模式中触发器名不能相同。

触发器的限制

- CREATE TRIGGER 语句文本的字符长度不能超过 32KB；
- 触发器体内的 SELECT 语句只能为 SELECT … INTO …结构，或者为定义游标所使用的 SELECT 语句。
- 触发器中不能使用数据库事务控制语句 COMMIT; ROLLBACK, SVAEPOINT 语句；
- 由触发器所调用的过程或函数也不能使用数据库事务控制语句；
- 触发器中不能使用 LONG, LONG RAW 类型；
- 触发器内可以参照 LOB 类型列的列值，但不能通过 :NEW 修改 LOB 列中的数据；

问题：当触发器被触发时，要使用被插入、更新或删除的记录中的列值，有时要使用操作前、后列的值.
实现: :NEW 修饰符访问操作完成后列的值
:OLD 修饰符访问操作完成前列的值

特性	INSERT	UPDATE	DELETE
OLD	NULL	有效	有效
NEW	有效	有效	NULL

例 1: 建立一个触发器，当职工表 emp 表被删除一条记录时，把被删除记录写到职工表删除日志表中去。

```
CREATE TABLE emp_his AS SELECT * FROM EMP WHERE 1=2;

CREATE OR REPLACE TRIGGER del_emp
```

```
    BEFORE DELETE ON scott.emp FOR EACH ROW
BEGIN
    --    将修改前数据插入到日志记录表 del_emp ,以供监督使用。
    INSERT INTO emp_his(deptno , empno, ename , job ,mgr , sal , comm , hiredate )
        VALUES( :old.deptno, :old.empno, :old.ename , :old.job,
                :old.mgr, :old.sal, :old.comm, :old.hiredate );
END;

DELETE emp WHERE empno=7788;

DROP TABLE emp_his;
DROP TRIGGER del_emp;
```

§8.2.3 创建替代(INSTEAD OF)触发器

创建触发器的一般语法是:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   INSTEAD OF
   {INSERT | DELETE | UPDATE [OF column [, column …]]}
   ON [schema.] view_name
   [REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]
   [FOR EACH ROW ]
   [WHEN condition]
   trigger_body;
```

其中：

BEFORE 和 AFTER 指出触发器的触发时序分别为前触发和后触发方式，前触发是在执行触发事件之前触发当前所创建的触发器，后触发是在执行触发事件之后触发当前所创建的触发器。

INSTEAD OF 选项使 ORACLE 激活触发器，而不执行触发事件。只能对视图和对象视图建立 INSTEAD OF 触发器，而不能对表、模式和数据库建立 INSTEAD OF 触发器。

FOR EACH ROW 选项说明触发器为行触发器。行触发器和语句触发器的区别表现在：行触发器要求当一个 DML 语句操走影响数据库中的多行数据时，对于其中的每个数据行，只要它们符合触发约束条件，均激活一次触发器；而语句触发器将整个语句操作作为触发事件，当它符合约束条件时，激活一次触发器。当省略 FOR EACH ROW 选项时，BEFORE 和 AFTER 触发器为语句触发器，而 INSTEAD OF 触发器则为行触发器。

REFERENCING 子句说明相关名称，在行触发器的 PL/SQL 块和 WHEN 子句中可以使用相关名称参照当前的新、旧列值，默认的相关名称分别为 OLD 和 NEW。触发器的 PL/SQL 块中应用相关名称时，必须在它们之前加冒号(:)，但在 WHEN 子句中则不能加冒号。

WHEN 子句说明触发约束条件。Condition 为一个逻辑表达时，其中必须包含相关名称，而不能包含查询语句，也不能调用 PL/SQL 函数。WHEN 子句指定的触发约束条件只能用在 BEFORE 和 AFTER 行触发器中，不能用在 INSTEAD OF 行触发器和其它类型的

触发器中。

INSTEAD_OF 用于对视图的 DML 触发，由于视图有可能是由多个表进行联结(join)而成，因而并非是所有的联结都是可更新的。但可以按照所需的方式执行更新，例如下面情况：

```
CREATE OR REPLACE VIEW emp_view AS
    SELECT deptno, count(*) total_employeer, sum(sal) total_salary
        FROM emp GROUP BY deptno;
```

在此视图中直接删除是非法：

```
SQL>DELETE FROM emp_view WHERE deptno=10;
DELETE FROM emp_view WHERE deptno=10
            *
ERROR 位于第 1 行:
ORA-01732: 此视图的数据操纵操作非法
```

但是我们可以创建 INSTEAD_OF 触发器来为 DELETE 操作执行所需的处理，即删除 EMP 表中所有基准行：

```
CREATE OR REPLACE TRIGGER emp_view_delete
   INSTEAD OF DELETE ON emp_view FOR EACH ROW
BEGIN
   DELETE FROM emp WHERE deptno= :old.deptno;
END emp_view_delete;

DELETE FROM emp_view WHERE deptno=10;

DROP TRIGGER emp_view_delete;
DROP VIEW emp_view;
```

§8.2.3 创建系统事件触发器

ORACLE8i 提供的系统事件触发器可以在 DDL 或数据库系统上被触发。DDL 指的是数据定义语言，如 CREATE 、ALTER 及 DROP 等。而数据库系统事件包括数据库服务器的启动或关闭，用户的登录与退出、数据库服务错误等。创建系统触发器的语法如下：

创建触发器的一般语法是:

```
CREATE OR REPLACE TRIGGER [sachema.] trigger_name
    {BEFORE|AFTER}
    {ddl_event_list | database_event_list}
    ON { DATABASE | [schema.] SCHEMA }
    [WHEN_clause]
    trigger_body;
```

其中: ddl_event_list：一个或多个 DDL 事件，事件间用 OR 分开；
database_event_list：一个或多个数据库事件，事件间用 OR 分开；

系统事件触发器既可以建立在一个模式上，又可以建立在整个数据库上。当建立在模式(SCHEMA)之上时，只有模式所指定用户的 DDL 操作和它们所导致的错误才激活触发器，默认时为当前用户模式。当建立在数据库(DATABASE)之上时，该数据库所有用户的 DDL 操作和他们所导致的错误，以及数据库的启动和关闭均可激活触发器。要在数据库之上建立触发器时，要求用户具有 ADMINISTER DATABASE TRIGGER 权限。

下面给出系统触发器的种类和事件出现的时机（前或后）：

事件	允许的时机	说明
启动 STARTUP	之后	实例启动时激活
关闭 SHUTDOWN	之前	实例正常关闭时激活
服务器错误 SERVERERROR	之后	只要有错误就激活
登录 LOGON	之后	成功登录后激活
注销 LOGOFF	之前	开始注销时激活
创建 CREATE	之前，之后	在创建之前或之后激活
撤消 DROP	之前，之后	在撤消之前或之后激活
变更 ALTER	之前，之后	在变更之前或之后激活

§8.2.4 系统触发器事件属性

事件属性\事件	Startup/Shutdown	Servererror	Logon/Logoff	DDL	DML
事件名称	*	*	*	*	*
数据库名称	*				
数据库实例号	*				
错误号		*			
用户名			*	*	
模式对象类型				*	*
模式对象名称				*	*
列					*

除 DML 语句的列属性外，其余事件属性值可通过调用 ORACLE 定义的事件属性函数来读取。

函数名称	数据类型	说　　明
Sysevent	VARCHAR2（20）	激活触发器的事件名称
Instance_num	NUMBER	数据库实例名
Database_name	VARCHAR2（50）	数据库名称
Server_error(posi)	NUMBER	错误信息栈中 posi 指定位置中的错误号
Is_servererror(err_number)	BOOLEAN	检查 err_number 指定的错误号是否在错误信息栈中，如果在则返回 TRUE，否则返回 FALSE。在触发器内调用此函数可

		以判断是否发生指定的错误。
Login_user	VARCHAR2(30)	登陆或注销的用户名称
Dictionary_obj_type	VARCHAR2(20)	DDL 语句所操作的数据库对象类型
Dictionary_obj_name	VARCHAR2(30)	DDL 语句所操作的数据库对象名称
Dictionary_obj_owner	VARCHAR2(30)	DDL 语句所操作的数据库对象所有者名称
Des_encrypted_password	VARCHAR2(2)	正在创建或修改的经过 DES 算法加密的用户口令

§8.2.5 使用触发器谓词

ORACLE 提供三个参数 INSERTING, UPDATING, DELETING 用于判断触发了哪些操作。

谓词	行为
INSERTING	如果触发语句是 INSERT 语句，则为 TRUE,否则为 FALSE
UPDATING	如果触发语句是 UPDATE 语句，则为 TRUE,否则为 FALSE
DELETING	如果触发语句是 DELETE 语句，则为 TRUE,否则为 FALSE

§8.2.6 重新编译触发器

如果在触发器内调用其它函数或过程，当这些函数或过程被删除或修改后，触发器的状态将被标识为无效。当 DML 语句激活一个无效触发器时，ORACLE 将重新编译触发器代码，如果编译时发现错误，这将导致 DML 语句执行失败。

在 PL/SQL 程序中可以调用 ALTER TRIGGER 语句重新编译已经创建的触发器，格式为：

ALTER TRIGGER [schema.] trigger_name COMPILE [DEBUG]

其中：DEBUG 选项要器编译器生成 PL/SQL 程序条使其所使用的调试代码。

§8.3 删除和使能触发器

- **删除触发器：**

  DROP TRIGGER trigger_name;

当删除其他用户模式中的触发器名称，需要具有 DROP ANY TRIGGER 系统权限，当删除建立在数据库上的触发器时，用户需要具有 ADMINISTER DATABASE TRIGGER 系统权限。

此外，当删除表或视图时，建立在这些对象上的触发器也随之删除。

- **使能触发器**

  数据库 TRIGGER 的状态：

  有效状态(ENABLE)：当触发事件发生时，处于有效状态的数据库触发器 TRIGGER 将被触发。

  无效状态(DISABLE)：当触发事件发生时，处于无效状态的数据库触发器 TRIGGER 将不会被触发，此时就跟没有这个数据库触发器(TRIGGER) 一样。

  数据库 TRIGGER 的这两种状态可以互相转换。格式为：

ALTER TIGGER trigger_name [DISABLE | ENABLE];
例：ALTER TRIGGER emp_view_delete DISABLE;

ALTER TRIGGER 语句一次只能改变一个触发器的状态，而 ALTER TABLE 语句则一次能够改变与指定表相关的所有触发器的使用状态。格式为：

ALTER TABLE [schema.]table_name {ENABLE|DISABLE} ALL TRIGGERS;

例：使表 EMP 上的所有 TRIGGER 失效：

ALTER TABLE emp DISABLE ALL TRIGGERS;

§8.4 触发器和数据字典

相关数据字典：USER_TRIGGERS、ALL_TRIGGERS、DBA_TRIGGERS

```
COL trigger_name FORMAT A10
COL trigger_type FORMAT A10
COL triggering_event FORMAT A10
COL table_owner FORMAT A10
COL base_object_type FORMAT A10
COL referencing_names FORMAT A10
COL status FORMAT A10
COL action_type FORMAT A10

SELECT TRIGGER_NAME, TRIGGER_TYPE, TRIGGERING_EVENT,
    TABLE_OWNER, BASE_OBJECT_TYPE, REFERENCING_NAMES,
    STATUS, ACTION_TYPE
    FROM user_triggers;
```

§8.5 数据库触发器的应用举例

例 1：创建一个 DML 语句级触发器，当对 emp 表执行 INSERT, UPDATE, DELETE 操作时，它自动更新 dept_summary 表中的数据。由于在 PL/SQL 块中不能直接调用 DDL 语句，所以，利用 ORACLE 内置包 DBMS_UTILITY 中的 EXEC_DDL_STATEMENT 过程，由它执行 DDL 语句创建触发器。

```
CREATE TABLE dept_summary(
    Deptno NUMBER(2),
    Sal_sum NUMBER(9, 2),
    Emp_count NUMBER);

INSERT INTO dept_summary(deptno, sal_sum, emp_count)
    SELECT deptno, SUM(sal), COUNT(*)
    FROM emp
    GROUP BY deptno;
```

```
--创建一个 PL/SQL 过程 disp_dept_summary
--在触发器中调用该过程显示 dept_summary 标中的数据。

CREATE OR REPLACE PROCEDURE disp_dept_summary
IS
    Rec dept_summary%ROWTYPE;
    CURSOR c1 IS SELECT * FROM dept_summary;
BEGIN
    OPEN c1;
    FETCH c1 INTO REC;
    DBMS_OUTPUT.PUT_LINE('deptno      sal_sum      emp_count');
    DBMS_OUTPUT.PUT_LINE('---------      ------------      ----------------');
    WHILE c1%FOUND LOOP
        DBMS_OUTPUT.PUT_LINE(RPAD(rec.deptno, 6)||
            To_char(rec.sal_sum, '$999,999.99')||
            LPAD(rec.emp_count, 13));
        FETCH c1 INTO rec;
    END LOOP;
    CLOSE c1;
END;

BEGIN
    DBMS_OUTPUT.PUT_LINE('插入前');
    Disp_dept_summary();
    DBMS_UTILITY.EXEC_DDL_STATEMENT('
        CREATE OR REPLACE TRIGGER trig1
            AFTER INSERT OR DELETE OR UPDATE OF sal ON emp
        BEGIN
            DBMS_OUTPUT.PUT_LINE(''正在执行 trig1  触发器…");
            DELETE FROM dept_summary;
            INSERT INTO dept_summary(deptno, sal_sum, emp_count)
                SELECT deptno, SUM(sal), COUNT(*)
                    FROM emp GROUP BY deptno;
        END;
    ');

    INSERT INTO dept(deptno, dname, loc)
        VALUES(90, 'demo_dept', 'none_loc');
    INSERT INTO emp(ename, deptno, empno, sal)
        VALUES(USER, 90, 9999, 3000);
    DBMS_OUTPUT.PUT_LINE('插入后');
    Disp_dept_summary();
```

```
    UPDATE emp SET sal=1000 WHERE empno=9999;
    DBMS_OUTPUT.PUT_LINE('修改后');
    Disp_dept_summary();

    DELETE FROM emp WHERE empno=9999;
    DELETE FROM dept WHERE deptno=90;
    DBMS_OUTPUT.PUT_LINE('删除后');
    Disp_dept_summary();

    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig1');
EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
```

例 2：创建 DML 语句行级触发器。当对 emp 表执行 INSERT, UPDATE, DELETE 操作时，它自动更新 dept_summary 表中的数据。由于在 PL/SQL 块中不能直接调用 DDL 语句，所以，利用 ORACLE 内置包 DBMS_UTILITY 中的 EXEC_DDL_STATEMENT 过程，由它执行 DDL 语句创建触发器。

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('插入前');
    Disp_dept_summary();
    DBMS_UTILITY.EXEC_DDL_STATEMENT(
        'CREATE OR REPLACE TRIGGER trig2_update
            AFTER UPDATE OF sal ON emp
            REFERENCING OLD AS old_emp NEW AS new_emp
            FOR EACH ROW
            WHEN (old_emp.sal != new_emp.sal)
        BEGIN
            DBMS_OUTPUT.PUT_LINE(''正在执行 trig2_update 触发器…'');
            DBMS_OUTPUT.PUT_LINE(''sal 旧值：''|| :old_emp.sal);
            DBMS_OUTPUT.PUT_LINE(''sal 新值：''|| :new_emp.sal);
            UPDATE dept_summary
                SET sal_sum=sal_sum + :new_emp.sal - :old_emp.sal
                WHERE deptno = :new_emp.deptno;
        END;'
    );

    DBMS_UTILITY.EXEC_DDL_STATEMENT(
        'CREATE OR REPLACE TRIGGER trig2_insert
            AFTER INSERT ON emp
            REFERENCING NEW AS new_emp
            FOR EACH ROW
        DECLARE
```

```
        I NUMBER;
    BEGIN
        DBMS_OUTPUT.PUT_LINE(''正在执行 trig2_insert 触发器…");
        SELECT COUNT(*) INTO I
            FROM dept_summary WHERE deptno = :new_emp.deptno;
        IF I > 0 THEN
            UPDATE dept_summary
                SET sal_sum=sal_sum+:new_emp.sal,
                    Emp_count=emp_count+1
                WHERE deptno = :new_emp.deptno;
        ELSE
            INSERT INTO dept_summary
                VALUES (:new_emp.deptno, :new_emp.sal, 1);
        END IF;
    END;'
);

DBMS_UTILITY.EXEC_DDL_STATEMENT(
    'CREATE OR REPLACE TRIGGER trig2_delete
        AFTER DELETE ON emp
        REFERENCING OLD AS old_emp
        FOR EACH ROW
    DECLARE
        I NUMBER;
    BEGIN
        DBMS_OUTPUT.PUT_LINE(''正在执行 trig2_delete 触发器…");
        SELECT emp_count INTO I
            FROM dept_summary WHERE deptno = :old_emp.deptno;
        IF I >1 THEN
            UPDATE dept_summary
                SET sal_sum=sal_sum - :old_emp.sal,
                    Emp_count=emp_count - 1
                WHERE deptno = :old_emp.deptno;
        ELSE
            DELETE FROM dept_summary WHERE deptno = :old_emp.deptno;
        END IF;
    END;'
);

INSERT INTO dept(deptno, dname, loc)
    VALUES(90, 'demo_dept', 'none_loc');
INSERT INTO emp(ename, deptno, empno, sal)
    VALUES(USER, 90, 9999, 3000);
INSERT INTO emp(ename, deptno, empno, sal)
```

```
        VALUES(USER, 90, 9998, 2000);
    DBMS_OUTPUT.PUT_LINE('插入后');
    Disp_dept_summary();

    UPDATE emp SET sal = sal*1.1 WHERE deptno=90;
    DBMS_OUTPUT.PUT_LINE('修改后');
    Disp_dept_summary();

    DELETE FROM emp WHERE deptno=90;
    DELETE FROM dept WHERE deptno=90;
    DBMS_OUTPUT.PUT_LINE('删除后');
    Disp_dept_summary();

    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig2_update');
    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig2_insert');
    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig2_delete');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
```

例 3：利用 ORACLE 提供的条件谓词 INSERTING、UPDATING 和 DELETING 创建与例 2 具有相同功能的触发器。

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('插入前');
    Disp_dept_summary();
    DBMS_UTILITY.EXEC_DDL_STATEMENT(
        'CREATE OR REPLACE TRIGGER trig2
            AFTER INSERT OR DELETE OR UPDATE OF sal
            ON emp
            REFERENCING OLD AS old_emp NEW AS new_emp
            FOR EACH ROW
        DECLARE
            I NUMBER;
        BEGIN
            IF UPDATING AND :old_emp.sal != :new_emp.sal THEN
                DBMS_OUTPUT.PUT_LINE(''正在执行 trig2 触发器…'');
                DBMS_OUTPUT.PUT_LINE(''sal 旧值：''|| :old_emp.sal);
                DBMS_OUTPUT.PUT_LINE(''sal 新值：''|| :new_emp.sal);
                UPDATE dept_summary
                    SET sal_sum=sal_sum + :new_emp.sal - :old_emp.sal
                    WHERE deptno = :new_emp.deptno;
            ELSIF INSERTING THEN
                DBMS_OUTPUT.PUT_LINE(''正在执行 trig2 触发器…'');
```

```
            SELECT COUNT(*) INTO I
                FROM dept_summary
                WHERE deptno = :new_emp.deptno;
            IF I > 0 THEN
                UPDATE dept_summary
                    SET sal_sum=sal_sum+:new_emp.sal,
                        Emp_count=emp_count+1
                    WHERE deptno = :new_emp.deptno;
            ELSE
                INSERT INTO dept_summary
                    VALUES (:new_emp.deptno, :new_emp.sal, 1);
            END IF;
        ELSE
            DBMS_OUTPUT.PUT_LINE(''正在执行 trig2 触发器…'');
            SELECT emp_count INTO I
                FROM dept_summary WHERE deptno = :old_emp.deptno;
            IF I > 1 THEN
            UPDATE dept_summary
                SET sal_sum=sal_sum - :old_emp.sal,
                    Emp_count=emp_count - 1
                WHERE deptno = :old_emp.deptno;
            ELSE
                DELETE FROM dept_summary
                    WHERE deptno = :old_emp.deptno;
            END IF;
        END IF;
    END;'
);

INSERT INTO dept(deptno, dname, loc)
    VALUES(90, 'demo_dept', 'none_loc');
INSERT INTO emp(ename, deptno, empno, sal)
    VALUES(USER, 90, 9999, 3000);
INSERT INTO emp(ename, deptno, empno, sal)
    VALUES(USER, 90, 9998, 2000);
DBMS_OUTPUT.PUT_LINE('插入后');
Disp_dept_summary();

UPDATE emp SET sal = sal*1.1 WHERE deptno=90;
DBMS_OUTPUT.PUT_LINE('修改后');
Disp_dept_summary();

DELETE FROM emp WHERE deptno=90;
DELETE FROM dept WHERE deptno=90;
```

```
    DBMS_OUTPUT.PUT_LINE('删除后');
    Disp_dept_summary();

    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig2');
EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
```

例 4：创建 INSTEAD OF 触发器。首先创建一个视图 myview，由于该视图是复合查询所产生的视图，所以不能执行 DML 语句。根据用户对视图所插入的数据判断需要将数据插入到哪个视图基表中，然后对该基表执行插入操作。

```
DECLARE
    No NUMBER;
    Name VARCHAR2(20);
BEGIN
    DBMS_UTILITY.EXEC_DDL_STATEMENT('
        CREATE OR REPLACE VIEW myview AS
            SELECT empno, ename, ''E'' type FROM emp
            UNION
            SELECT dept.deptno, dname, ''D'' FROM dept
    ');
    -- 创建 INSTEAD OF 触发器 trigger3;
    DBMS_UTILITY.EXEC_DDL_STATEMENT('
        CREATE OR REPLACE TRIGGER trig3
            INSTEAD OF INSERT ON myview
            REFERENCING NEW n
            FOR EACH ROW
        DECLARE
            Rows INTEGER;
        BEGIN
            DBMS_OUTPUT.PUT_LINE(''正在执行 trig3 触发器…'');
            IF :n.type = ''D'' THEN
                SELECT COUNT(*) INTO rows
                    FROM dept WHERE deptno = :n.empno;
                IF rows = 0 THEN
                    DBMS_OUTPUT.PUT_LINE(''向 dept 表中插入数据…'');
                    INSERT INTO dept(deptno, dname, loc)
                        VALUES (:n.empno, :n.ename, ''none'');
                ELSE
                    DBMS_OUTPUT.PUT_LINE(''编号为''|| :n.empno||
                        ''的部门已存在，插入操作失败！'');
                END IF;
            ELSE
```

```
                SELECT COUNT(*) INTO rows
                    FROM emp WHERE empno = :n.empno;
                IF rows = 0 THEN
                    DBMS_OUTPUT.PUT_LINE(''向 emp 表中插入数据…'');
                    INSERT INTO emp(empno, ename)
                        VALUES(:n.empno, :n.ename);
                ELSE
                    DBMS_OUTPUT.PUT_LINE(''编号为''|| :n.empno||
                        ''的人员已存在，插入操作失败！'');
                END IF;
            END IF;
        END;
        ');

    INSERT INTO myview VALUES (70, 'demo', 'D');
    INSERT INTO myview VALUES (9999, USER, 'E');
    SELECT deptno, dname INTO no, name FROM dept WHERE deptno=70;
    DBMS_OUTPUT.PUT_LINE('员工编号：'||TO_CHAR(no)||'姓名：'||name);
    SELECT empno, ename INTO no, name FROM emp WHERE empno=9999;
    DBMS_OUTPUT.PUT_LINE('部门编号：'||TO_CHAR(no)||'姓名：'||name);
    DELETE FROM emp WHERE empno=9999;
    DELETE FROM dept WHERE deptno=70;
    DBMS_UTILITY.EXEC_DDL_STATEMENT('DROP TRIGGER trig3');
END;
```

例 5：利用 ORACLE 事件属性函数，创建一个系统事件触发器。首先创建一个事件日志表 eventlog，由它存储用户在当前数据库中所创建的数据库对象，以及用户的登陆和注销、数据库的启动和关闭等事件，之后创建 trig4_ddl、trig4_before 和 trig4_after 触发器，它们调用事件属性函数将各个事件记录到 eventlog 数据表中。

```
BEGIN
    -- 创建用于记录事件日志的数据表
    DBMS_UTILITY.EXEC_DDL_STATEMENT('
        CREATE TABLE eventlog(
            Eventname VARCHAR2(20) NOT NULL,
            Eventdate date default sysdate,
            Inst_num NUMBER NULL,
            Db_name VARCHAR2(50) NULL,
            Srv_error NUMBER NULL,
            Username VARCHAR2(30) NULL,
            Obj_type VARCHAR2(20) NULL,
            Obj_name VARCHAR2(30) NULL,
            Obj_owner VARCHAR2(30) NULL
        )
    ');
```

```
-- 创建 DDL 触发器 trig4_ddl
DBMS_UTILITY.EXEC_DDL_STATEMENT('
    CREATE OR REPLACE TRIGGER trig4_ddl
        AFTER CREATE OR ALTER OR DROP
        ON DATABASE
    DECLARE
        Event VARCHAR2(20);
        Typ VARCHAR2(20);
        Name VARCHAR2(30);
        Owner VARCHAR2(30);
    BEGIN
        -- 读取 DDL 事件属性
        Event := SYSEVENT;
        Typ := DICTIONARY_OBJ_TYPE;
        Name := DICTIONARY_OBJ_NAME;
        Owner := DICTIONARY_OBJ_OWNER;
        -- 将事件属性插入到事件日志表中
        INSERT INTO scott.eventlog(eventname, obj_type, obj_name, obj_owner)
            VALUES(event, typ, name, owner);
    END;
');

-- 创建 LOGON、STARTUP 和 SERVERERROR 事件触发器
DBMS_UTILITY.EXEC_DDL_STATEMENT('
    CREATE OR REPLACE TRIGGER trig4_after
        AFTER LOGON OR STARTUP OR SERVERERROR
        ON DATABASE
    DECLARE
        Event VARCHAR2(20);
        Instance NUMBER;
        Err_num NUMBER;
        Dbname VARCHAR2(50);
        User VARCHAR2(30);
    BEGIN
        Event := SYSEVENT;
        IF event = ''LOGON'' THEN
            User := LOGIN_USER;
            INSERT INTO eventlog(eventname, username)
                VALUES(event, user);
        ELSIF event = ''SERVERERROR'' THEN
            Err_num := SERVER_ERROR(1);
            INSERT INTO eventlog(eventname, srv_error)
                VALUES(event, err_num);
```

```
            ELSE
                Instance := INSTANCE_NUM;
                Dbname := DATABASE_NAME;
                INSERT INTO eventlog(eventname, inst_num, db_name)
                    VALUES(event, instance, dbname);
            END IF;
        END;
    ');

    -- 创建 LOGOFF 和 SHUTDOWN 事件触发器
    DBMS_UTILITY.EXEC_DDL_STATEMENT('
        CREATE OR REPLACE TRIGGER trig4_before
            BEFORE LOGOFF OR SHUTDOWN
            ON DATABASE
        DECLARE
            Event VARCHAR2(20);
            Instance NUMBER;
            Dbname VARCHAR2(50);
            User VARCHAR2(30);
        BEGIN
            Event := SYSEVENT;
            IF event = ''LOGOFF'' THEN
                User := LOGIN_USER;
                INSERT INTO eventlog(eventname, username)
                    VALUES(event, user);
            ELSE
                Instance := INSTANCE_NUM;
                Dbname := DATABASE_NAME;
                INSERT INTO eventlog(eventname, inst_num, db_name)
                    VALUES(event, instance, dbname);
            END IF;
        END;
    ');
END;

CREATE TABLE mydata(mydate NUMBER);
CONNECT SCOTT/TIGER

COL eventname FORMAT A10
COL eventdate FORMAT A12
COL username FORMAT A10
COL obj_type FORMAT A15
COL obj_name FORMAT A15
COL obj_owner FORMAT A10
```

```
SELECT eventname, eventdate, obj_type, obj_name, obj_owner, username, Srv_error
    FROM eventlog;

DROP TRIGGER trig4_ddl;
DROP TRIGGER trig4_before;
DROP TRIGGER trig4_after;
DROP TABLE eventlog;
DROP TABLE mydata;
```

§8.6 数据库触发器的应用实例

用户可以使用数据库触发器实现各种功能：

- 复杂的审计功能；

例：将 EMP 表的变化情况记录到 AUDIT_TABLE 和 AUDIT_TABLE_VALUES 中。

```
CREATE TABLE audit_table(
    Audit_id        NUMBER,
    User_name VARCHAR2(20),
    Now_time DATE,
    Terminal_name VARCHAR2(10),
    Table_name VARCHAR2(10),
    Action_name VARCHAR2(10),
    Emp_id NUMBER(4));

CREATE TABLE audit_table_val(
    Audit_id NUMBER,
    Column_name VARCHAR2(10),
    Old_val NUMBER(7,2),
    New_val NUMBER(7,2));

CREATE SEQUENCE audit_seq
    START WITH 1000
    INCREMENT BY 1
    NOMAXVALUE
    NOCYCLE NOCACHE;

CREATE OR REPLACE TRIGGER audit_emp
    AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH ROW
DECLARE
    Time_now DATE;
    Terminal CHAR(10);
BEGIN
    Time_now:=sysdate;
    Terminal:=USERENV('TERMINAL');
```

```
    IF INSERTING THEN
        INSERT INTO audit_table
            VALUES(audit_seq.NEXTVAL, user, time_now,
                terminal, 'EMP', 'INSERT', :new.empno);
    ELSIF DELETING THEN
        INSERT INTO audit_table
            VALUES(audit_seq.NEXTVAL, user, time_now,
                terminal, 'EMP', 'DELETE', :old.empno);
    ELSE
        INSERT INTO audit_table
            VALUES(audit_seq.NEXTVAL, user, time_now,
                terminal, 'EMP', 'UPDATE', :old.empno);
        IF UPDATING('SAL') THEN
            INSERT INTO audit_table_val
                VALUES(audit_seq.CURRVAL, 'SAL', :old.sal, :new.sal);
        ELSE UPDATING('DEPTNO')
            INSERT INTO audit_table_val
                VALUES(audit_seq.CURRVAL, 'DEPTNO', :old.deptno, :new.deptno);
        END IF;
    END IF;
END;
```

- 增强数据的完整性管理；

例：修改 DEPT 表的 DEPTNO 列时，同时把 EMP 表中相应的 DEPTNO 也作相应的修改；

```
CREATE SEQUENCE update_sequence
    INCREMENT BY 1
    START WITH 1000
    MAXVALUE 5000 CYCLE;

ALTER TABLE emp
    ADD update_id NUMBER;

CREATE OR REPLACE PACKAGE integritypackage AS
    Updateseq NUMBER;
END integritypackage;

CREATE OR REPLACE PACKAGE BODY integritypackage AS
END integritypackage;

CREATE OR REPLACE TRIGGER dept_cascade1
    BEFORE UPDATE OF deptno ON dept
DECLARE
    Dummy NUMBER;
```

```
BEGIN
    SELECT update_sequence.NEXTVAL INTO dummy FROM dual;
    Integritypackage.updateseq:=dummy;
END;

CREATE OR REPLACE TRIGGER dept_cascade2
    AFTER DELETE OR UPDATE OF deptno ON dept
    FOR EACH ROW
BEGIN
    IF UPDATING THEN
        UPDATE emp SET deptno=:new.deptno,
            update_id=integritypackage.updateseq
            WHERE emp.deptno=:old.deptno AND update_id IS NULL;
    END IF;
    IF DELETING THEN
        DELETE FROM emp
            WHERE emp.deptno=:old.deptno;
    END IF;
END;

CREATE OR REPLACE TRIGGER dept_cascade3
    AFTER UPDATE OF deptno ON dept
BEGIN
    UPDATE emp SET update_id=NULL
        WHERE update_id=integritypackage.updateseq;
END;

SELECT * FROM EMP ORDER BY DEPTNO;

UPDATE dept SET deptno=25 WHERE deptno=20;
```

- 帮助实现安全控制；

例：保证对 EMP 表的修改仅在工作日的工作时间；

```
CREATE TABLE company_holidays(day DATE);

INSERT INTO company_holidays
    VALUES(sysdate);
INSERT INTO company_holidays
    VALUES(TO_DATE('21-10 月-01', 'DD-MON-YY'));

CREATE OR REPLACE TRIGGER emp_permit_change
    BEFORE INSERT OR DELETE OR UPDATE ON emp
DECLARE
```

```
    Dummy NUMBER;
    Not_on_weekends EXCEPTION;
    Not_on_holidays EXCEPTION;
    Not_working_hours EXCEPTION;
BEGIN
    /* check for weekends */
    IF TO_CHAR(SYSDATE, 'DAY') IN ('星期六', '星期日') THEN
        RAISE not_on_weekends;
    END IF;
    /* check for company holidays */
    SELECT COUNT(*) INTO dummy FROM company_holidays
        WHERE TRUNC(day)=TRUNC(SYSDATE);
    IF dummy >0 THEN
        RAISE not_on_holidays;
    END IF;
    /* check for work hours(8:00 AM to 18:00 PM */
    IF (TO_CHAR(SYSDATE,'HH24')<8 OR TO_CHAR(SYSDATE, 'HH24')>18) THEN
        RAISE not_working_hours;
    END IF;
EXCEPTION
    WHEN not_on_weekends THEN
        RAISE_APPLICATION_ERROR(-20324,
            'May not change employee table during the weekends');
    WHEN not_on_holidays THEN
        RAISE_APPLICATION_ERROR(-20325,
            'May not change employee table during a holiday');
    WHEN not_working_hours THEN
        RAISE_APPLICATION_ERROR(-20326,
            'May not change employee table during no_working hours');
END;
```

- 管理复杂的表复制；
- 防止非法的事务发生；
- 自动生成派生的列值；
- 帮助式显复杂的商业管理。