# ORACLE触发器详解 - IndexMan的专栏 - 博客频道 - CSDN

**Oracle PL/SQL编程之八：**

**把触发器说透**

**本篇主要内容如下：**



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，它与存储过程和函数不同的是，存储过程与函数需要用户显示调用才执行，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。所以触发器常用来完成由数据库的完整性约束难以完成的复杂业务规则的约束，或用来监视对数据库的各种操作，实现审计的功能。

**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:

1 触发事件：引起触发器被触发的事件。 例如：DML语句(INSERT, UPDATE, DELETE语句对表或视图执行数据处理操作)、DDL语句（如CREATE、ALTER、DROP语句在数据库中创建、修改、删除模式对象）、数据库系统事件（如系统启动或退出、异常错误）、用户事件（如登录或退出数据库）。

1 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。

1 触发操作：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。

1 触发对象：包括表、视图、模式、数据库。只有在这些对象上发生了符合触发条件的触发事件，才会执行触发操作。

1 触发条件：由WHEN子句指定一个逻辑表达式。只有当该表达式的值为TRUE时，遇到触发事件才会自动执行触发器，使其执行触发操作。

1 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。

**编写触发器时，需要注意以下几点：**

1 触发器不接受参数。

1 一个表上最多可有12个触发器，但同一时间、同一事件、同一类型的触发器只能有一个。并各触发器之间不能有矛盾。

1 在一个表上的触发器越多，对在该表上的DML操作的性能影响就越大。

1 触发器最大为32KB。若确实需要，可以先建立过程，然后在触发器中用CALL语句进行调用。

1 **在触发器的执行部分只能用DML语句（SELECT、INSERT、UPDATE、DELETE），不能使用DDL语句（CREATE、ALTER、DROP）。**

1 触发器中不能包含事务控制语句(COMMIT，ROLLBACK，SAVEPOINT)。因为触发器是触发语句的一部分，触发语句被提交、回退时，触发器也被提交、回退了。

1 在触发器主体中调用的任何过程、函数，都不能使用事务控制语句。

1 在触发器主体中不能申明任何Long和blob变量。新值new和旧值old也不能是表中的任何long和blob列。

1 不同类型的触发器(如DML触发器、INSTEAD OF触发器、系统触发器)的语法格式和作用有较大区别。

**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 …]]}
[OR {INSERT | DELETE | UPDATE [OF column [, column …]]}...]
ON[schema.]table_name |[schema.]view_name
[REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]
[FOR EACH ROW ]
[WHEN condition]
PL/SQL_BLOCK | CALL procedure_name;
```

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 触发器则只能为行触发器。

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：

1 执行 BEFORE行级触发器

1 执行 DML语句

1 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。

**DML触发器的限制**

1 CREATE TRIGGER语句文本的字符长度不能超过32KB;

1 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。

1 触发器中不能使用数据库事务控制语句 COMMIT; ROLLBACK, SVAEPOINT 语句;

1 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;

1 触发器中不能使用LONG, LONG RAW 类型;

1 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;

**DML触发器基本要点**

1 **触发时机：**指定触发器的触发时间。如果指定为BEFORE，则表示在执行DML操作之前触发，以便防止某些错误操作发生或实现某些业务规则；如果指定为AFTER，则表示在执行DML操作之后触发，以便记录该操作或做某些事后处理。

1 **触发事件：**引起触发器被触发的事件，即DML操作（INSERT、UPDATE、DELETE）。既可以是单个触发事件，也可以是多个触发事件的组合（只能使用OR逻辑组合，不能使用AND逻辑组合）。

1 **条件谓词：**当在触发器中包含多个触发事件（INSERT、UPDATE、DELETE）的组合时，为了分别针对不同的事件进行不同的处理，需要使用ORACLE提供的如下条件谓词。

1）。**INSERTING：**当触发事件是INSERT时，取值为TRUE，否则为FALSE。

2）。**UPDATING [（column_1,column_2,…,column_x）]：**当触发事件是UPDATE 时，如果修改了column_x列，则取值为TRUE，否则为FALSE。其中column_x是可选的。

3）。**DELETING：**当触发事件是DELETE时，则取值为TRUE，否则为FALSE。

**解发对象：**指定触发器是创建在哪个表、视图上。

1 **触发类型：**是语句级还是行级触发器。

1 **触发条件：**由WHEN子句指定一个逻辑表达式，只允许在行级触发器上指定触发条件，指定UPDATING后面的列的列表。

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 | 实际值 | 实际值 |
| NEW | 实际值 | 实际值 | 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。

```
CREATETABLE emp_his ASSELECT*FROM EMP WHERE1=2;
CREATEORREPLACETRIGGER tr_del_emp
   BEFORE DELETE--指定触发时机为删除操作前触发
ON scott.emp
FOR EACH ROW   --说明创建的是行级触发器
BEGIN
--将修改前数据插入到日志记录表 del_emp ，以供监督使用。
INSERTINTO 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;
DROPTABLE emp_his;
DROPTRIGGER del_emp;
```

**例2**：限制对Departments表修改（包括INSERT,DELETE,UPDATE）的时间范围，即不允许在非工作时间修改departments表。

```
CREATEORREPLACETRIGGER tr_dept_time
BEFORE INSERTORDELETEORUPDATE
ON departments
BEGIN
IF (TO_CHAR(sysdate,'DAY') IN ('星期六', '星期日')) OR (TO_CHAR(sysdate, 'HH24:MI')
NOTBETWEEN'08:30'AND'18:00') THEN
     RAISE_APPLICATION_ERROR(-20001, '不是上班时间，不能修改departments表');
ENDIF;
END;
```

**例3**：限定只对部门号为80的记录进行行触发器操作。

```
CREATEORREPLACETRIGGER tr_emp_sal_comm
BEFORE UPDATEOF salary, commission_pct
ORDELETE
ON HR.employees
FOR EACH ROW
WHEN (old.department_id =80)
BEGIN
CASE
WHEN UPDATING ('salary') THEN
IF :NEW.salary < :old.salary THEN

          RAISE_APPLICATION_ERROR(-20001, '部门80的人员的工资不能降');
ENDIF;
WHEN UPDATING ('commission_pct') THEN

IF :NEW.commission_pct < :old.commission_pct THEN
          RAISE_APPLICATION_ERROR(-20002, '部门80的人员的奖金不能降');
ENDIF;
WHEN DELETING THEN
         RAISE_APPLICATION_ERROR(-20003, '不能删除部门80的人员记录');
ENDCASE;
END;
```

```
/*
实例：
UPDATE employees SET salary = 8000 WHERE employee_id = 177;
DELETE FROM employees WHERE employee_id in (177,170);
*/
```

**例4：**利用行触发器实现级联更新。在修改了主表regions中的region_id之后（AFTER），级联的、自动的更新子表countries表中原来在该地区的国家的region_id。

```
CREATEORREPLACETRIGGER tr_reg_cou
AFTER updateOF region_id
ON regions
FOR EACH ROW
BEGIN
 DBMS_OUTPUT.PUT_LINE('旧的region_id值是'||:old.region_id
||'、新的region_id值是'||:new.region_id);
UPDATE countries SET region_id = :new.region_id
WHERE region_id = :old.region_id;
END;
```

**例5：**在触发器中调用过程。

```
CREATEORREPLACEPROCEDURE add_job_history
 ( p_emp_id          job_history.employee_id%type
   , p_start_date      job_history.start_date%type
  , p_end_date        job_history.end_date%type
   , p_job_id          job_history.job_id%type
   , p_department_id   job_history.department_id%type
   )
IS
BEGIN
INSERTINTO job_history (employee_id, start_date, end_date,
                        job_id, department_id)
VALUES(p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id);
END add_job_history;

--创建触发器调用存储过程...
CREATEORREPLACETRIGGER update_job_history
 AFTER UPDATEOF job_id, department_id ON employees
FOR EACH ROW
BEGIN
 add_job_history(:old.employee_id, :old.hire_date, sysdate,
                 :old.job_id, :old.department_id);
END;
```

### 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 …]]}
[OR {INSERT | DELETE | UPDATE [OF column [, column …]]}...]
ON[schema.] view_name --只能定义在视图上
```

[REFERENCING {OLD [AS] old | NEW [AS] new| PARENT as parent}]

[FOR EACH ROW ]--因为INSTEAD OF触发器只能在行级上触发,所以没有必要指定

[WHEN condition]

PL/SQL_block | CALL procedure_name;

其中：

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：

**例1：**

```
CREATEORREPLACEVIEW emp_view AS
SELECT deptno, count(*) total_employeer, sum(sal) total_salary
FROM emp GROUPBY deptno;
```

在此视图中直接删除是非法：

```
SQL>DELETEFROM emp_view WHERE deptno=10;
DELETEFROM 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：

```
CREATEORREPLACETRIGGER emp_view_delete
   INSTEAD OFDELETEON emp_view FOR EACH ROW
BEGIN
DELETEFROM emp WHERE deptno= :old.deptno;
END emp_view_delete;

DELETEFROM emp_view WHERE deptno=10;

DROPTRIGGER emp_view_delete;

DROPVIEW emp_view;
```

**例2：**创建复杂视图，针对INSERT操作创建INSTEAD OF触发器，向复杂视图插入数据。

1　　创建视图：

```
CREATEORREPLACE FORCE VIEW "HR"."V_REG_COU" ("R_ID", "R_NAME", "C_ID", "C_NAME")
AS
SELECT r.region_id,
    r.region_name,
    c.country_id,
    c.country_name
```

```
FROM regions r,
    countries c
WHERE r.region_id = c.region_id;
```

1　　创建触发器：

```
CREATEORREPLACETRIGGER "HR"."TR_I_O_REG_COU" INSTEAD OF
INSERTON v_reg_cou FOR EACH ROW DECLARE v_count NUMBER;
BEGIN
SELECTCOUNT(*) INTO v_count FROM regions WHERE region_id = :new.r_id;
IF v_count =0THEN
INSERTINTO regions
      (region_id, region_name
      ) VALUES
      (:new.r_id, :new.r_name
      );
ENDIF;

SELECTCOUNT(*) INTO v_count FROM countries WHERE country_id = :new.c_id;
IF v_count =0THEN
INSERT
INTO countries
      (
        country_id,
        country_name,
        region_id
      )
VALUES
      (
        :new.c_id,
        :new.c_name,
        :new.r_id
      );
ENDIF;
END;
```

**创建INSTEAD OF触发器需要注意以下几点：**

1　　只能被创建在视图上，并且该视图没有指定WITH CHECK OPTION选项。

1　　不能指定BEFORE 或 AFTER选项。

1　　FOR EACH ROW子可是可选的，即INSTEAD OF触发器只能在行级上触发、或只能是行级触发器，没有必要指定。

1　　没有必要在针对一个表的视图上创建INSTEAD OF触发器，只要创建DML触发器就可以了。

### 8.2.3 创建系统事件触发器

ORACLE10G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：**

```
CREATEORREPLACETRIGGER[sachema.]trigger_name
{BEFORE|AFTER}
{ddl_event_list | database_event_list}
ON { DATABASE|[schema.]SCHEMA }
```

[WHEN condition]

PL/SQL_block | CALL procedure_name;

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 |
|---|---|---|
| STARTUP | AFTER | 启动数据库实例之后触发 |
| SHUTDOWN | BEFORE | 关闭数据库实例之前触发（非正常关闭不触发） |
| SERVERERROR | AFTER | 数据库服务器发生错误之后触发 |
| LOGON | AFTER | 成功登录连接到数据库后触发 |
| LOGOFF | BEFORE | 开始断开数据库连接之前触发 |
| CREATE | BEFORE，AFTER | 在执行CREATE语句创建数据库对象之前、之后触发 |
| DROP | BEFORE，AFTER | 在执行DROP语句删除数据库对象之前、之后触发 |
| ALTER | BEFORE，AFTER | 在执行ALTER语句更新数据库对象之前、之后触发 |
| DDL | BEFORE，AFTER | 在执行大多数DDL语句之前、之后触发 |

| | | |
|---|---|---|
| GRANT | BEFORE，AFTER | 执行GRANT语句授予权限之前、之后触发 |
| REVOKE | BEFORE，AFTER | 执行REVOKE语句收权限之前、之后触犯发 |
| RENAME | BEFORE，AFTER | 执行RENAME语句更改数据库对象名称之前、之后触犯发 |
| AUDIT / NOAUDIT | BEFORE，AFTER | 执行AUDIT或NOAUDIT进行审计或停止审计之前、之后触发 |

### 8.2.4 系统触发器事件属性

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

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 |
| --- | --- | --- |
| Ora_sysevent | VARCHAR2（20） | 激活触发器的事件名称 |
| Instance_num | NUMBER | 数据库实例名 |
| Ora_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_serverror(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 |

**例1：**创建触发器，存放有关事件信息。

```
DESC ora_sysevent
DESC ora_login_user
```

```
--创建用于记录事件用的表

CREATETABLE ddl_event
(crt_date timestampPRIMARYKEY,
 event_name VARCHAR2(20),
user_nameVARCHAR2(10),
 obj_type VARCHAR2(20),
 obj_name VARCHAR2(20));

--创建触犯发器
CREATEORREPLACETRIGGER tr_ddl
AFTER DDL ONSCHEMA
BEGIN
INSERTINTO ddl_event VALUES
   (systimestamp,ora_sysevent, ora_login_user,
    ora_dict_obj_type, ora_dict_obj_name);
END tr_ddl;
```

**例2：** 创建登录、退出触发器。

```
CREATETABLE log_event
(user_nameVARCHAR2(10),
 address VARCHAR2(20),
 logon_date timestamp,
 logoff_date timestamp);

--创建登录触发器
CREATEORREPLACETRIGGER tr_logon
AFTER LOGON ONDATABASE
BEGIN
INSERTINTO log_event (user_name, address, logon_date)
VALUES (ora_login_user, ora_client_ip_address, systimestamp);
END tr_logon;
--创建退出触发器
CREATEORREPLACETRIGGER tr_logoff
BEFORE LOGOFF ONDATABASE
BEGIN
INSERTINTO log_event (user_name, address, logoff_date)
VALUES (ora_login_user, ora_client_ip_address, systimestamp);
END tr_logoff;
```

### 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：

```
ALTERTRIGGER[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

1 **删除触发器：**

```
DROPTRIGGER 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。

1 **禁用或启用触发器**

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：

```
ALTERTABLE[schema.]table_name {ENABLE|DISABLE} ALL TRIGGERS;

--例：使表EMP 上的所有TRIGGER 失效：
ALTERTABLE 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

```
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。

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

INSERTINTO dept_summary(deptno, sal_sum, emp_count)
SELECT deptno, SUM(sal), COUNT(*)
FROM emp
GROUPBY 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。
CREATEORREPLACEPROCEDURE disp_dept_summary
IS
 Rec dept_summary%ROWTYPE;
CURSOR c1 ISSELECT*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;
');


INSERTINTO dept(deptno, dname, loc)
VALUES(90, ‘demo_dept’, ‘none_loc’);
INSERTINTO emp(ename, deptno, empno, sal)
VALUES(USER, 90, 9999, 3000);

 DBMS_OUTPUT.PUT_LINE('插入后');
 Disp_dept_summary();

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

DELETEFROM emp WHERE empno=9999;
DELETEFROM dept WHERE deptno=90;

 DBMS_OUTPUT.PUT_LINE('删除后');
 Disp_dept_summary();
 DBMS_UTILITY.EXEC_DDL_STATEMENT( ‘DROPTRIGGER 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;'
  );

INSERTINTO dept(deptno, dname, loc)
VALUES(90, 'demo_dept', 'none_loc');
```

```
INSERTINTO emp(ename, deptno, empno, sal)
VALUES(USER, 90, 9999, 3000);
INSERTINTO 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*1.1WHERE deptno=90;
  DBMS_OUTPUT.PUT_LINE('修改后');
  Disp_dept_summary();

DELETEFROM emp WHERE deptno=90;
DELETEFROM 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;'
  );

INSERTINTO dept(deptno, dname, loc)
VALUES(90, 'demo_dept', 'none_loc');
INSERTINTO emp(ename, deptno, empno, sal)
VALUES(USER, 90, 9999, 3000);
INSERTINTO 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*1.1WHERE deptno=90;
  DBMS_OUTPUT.PUT_LINE('修改后');
  Disp_dept_summary();

DELETEFROM emp WHERE deptno=90;
DELETEFROM 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…' );
INSERTINTO 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!'');
ENDIF;
ENDIF;
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('DROPTRIGGER 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;

CREATETABLE 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;

DROPTRIGGER trig4_ddl;
DROPTRIGGER trig4_before;
DROPTRIGGER trig4_after;
DROPTABLE eventlog;
DROPTABLE 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：

1 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。

```
CREATETABLE audit_table(
    Audit_id     NUMBER,
User_nameVARCHAR2(20),
    Now_time DATE,
    Terminal_name VARCHAR2(10),
    Table_name VARCHAR2(10),
    Action_name VARCHAR2(10),
    Emp_id NUMBER(4));

CREATETABLE 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 WITH1000
    INCREMENT BY1
    NOMAXVALUE
    NOCYCLE NOCACHE;

CREATEORREPLACETRIGGER audit_emp
    AFTER INSERTORUPDATEORDELETEON emp
FOR EACH ROW
DECLARE
    Time_now DATE;
    Terminal CHAR(10);
BEGIN
    Time_now:=sysdate;
    Terminal:=USERENV('TERMINAL');
IF INSERTING THEN
INSERTINTO audit_table
VALUES(audit_seq.NEXTVAL, user, time_now,
           terminal, 'EMP', 'INSERT', :new.empno);
    ELSIF DELETING THEN
INSERTINTO audit_table
VALUES(audit_seq.NEXTVAL, user, time_now,
           terminal, 'EMP', 'DELETE', :old.empno);
ELSE
INSERTINTO audit_table
VALUES(audit_seq.NEXTVAL, user, time_now,
           terminal, 'EMP', 'UPDATE', :old.empno);
IF UPDATING('SAL') THEN
INSERTINTO audit_table_val
VALUES(audit_seq.CURRVAL, 'SAL', :old.sal, :new.sal);
ELSE UPDATING('DEPTNO')
INSERTINTO audit_table_val
VALUES(audit_seq.CURRVAL, 'DEPTNO', :old.deptno, :new.deptno);
ENDIF;
ENDIF;
END;
```

1　　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 BY1
    START WITH1000
    MAXVALUE 5000 CYCLE;

ALTERTABLE emp
ADD update_id NUMBER;

CREATEORREPLACE PACKAGE integritypackage AS
    Updateseq NUMBER;
```

```
END integritypackage;

CREATEORREPLACE PACKAGE BODY integritypackage AS
END integritypackage;

CREATEORREPLACETRIGGER dept_cascade1
    BEFORE UPDATEOF deptno ON dept
DECLARE
DummyNUMBER;
BEGIN
SELECT update_sequence.NEXTVAL INTOdummyFROM dual;
    Integritypackage.updateseq:=dummy;
END;

CREATEORREPLACETRIGGER dept_cascade2
    AFTER DELETEORUPDATEOF 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 ISNULL;
ENDIF;
IF DELETING THEN
DELETEFROM emp
WHERE emp.deptno=:old.deptno;
ENDIF;
END;

CREATEORREPLACETRIGGER dept_cascade3
    AFTER UPDATEOF deptno ON dept
BEGIN
UPDATE emp SET update_id=NULL
WHERE update_id=integritypackage.updateseq;
END;

SELECT*FROM EMP ORDERBY DEPTNO;
UPDATE dept SET deptno=25WHERE deptno=20;
```

1　　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；

```
CREATETABLE company_holidays(day DATE);

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

CREATEORREPLACETRIGGER emp_permit_change
```

```
    BEFORE INSERTORDELETEORUPDATEON emp
DECLARE
DummyNUMBER;
    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;
ENDIF;
/* check for company holidays */
SELECTCOUNT(*) INTOdummyFROM company_holidays
WHERE TRUNC(day)=TRUNC(SYSDATE);
IFdummy>0THEN
    RAISE not_on_holidays;
ENDIF;
/* check for work hours(8:00 AM to 18:00 PM */
IF (TO_CHAR(SYSDATE,'HH24')<8OR TO_CHAR(SYSDATE, 'HH24')>18) THEN
  RAISE not_working_hours;
ENDIF;
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;
```