المحاضرة العاشرة

التحكم بالجداول باستخدام Database Triggers

الـ **Trigger** عبارة عند وحدة نمطية **PL / SQL block** تكون مرتبطة مع جدول أو أكثر مخزن في قاعدة البيانات ويتم تنفيذه تلقائياً عندما نقوم بتنفيذ إحدى الجمل التالية على الجدول المعين:-

1. عند إدخال سجل جديد في الجدول باستخدام جملة **Insert**.
2. عند تعديل سجل بالجدول باستخدام جملة **Update**.
3. عند حذف سجل من الجدول باستخدام **Delete**.

Syntax: -

CREATE [OR REPLACE] TRIGGER *trigger_name* < BEFOR | AFTER >
< INSERT, UPDATE or DELETE > **or** أي واحدة منهم أو أكثر مع ربطهم بـ
ON *table_name*
[FOR EACH ROW]
[When *condition*] شرط إضافي يحكم التنفيذ
PL / SQL block

1. Statement level Triggers :-

هذا النوع يتم تنفيذه مرة واحدة فقط وفيه نقوم بحذف جملة **FOR EACH ROW** عند التعريف.

2. Row level Triggers :-

يتم تنفيذه في كل مرة نقوم بإجراء تعديل أو حذف أو إدخال سجل جديد وفيه يمكننا أن نشير إلى قيمة أي حقل باستخدام المعاملين **New & Old** وفيه نستخدم جملة **For Each Row**.

أنواع الـ Triggers من حيث التنفيذ :-

(1)- **Before Triggers** ويتم تنفيذه قبل إجراء التحديث على الجدول.

(2)- **After Triggers** ويتم تنفيذه بعد عمل التحديث على الجدول.

لماذا نستخدم الـ Triggers :-

كما سبق وعرفنا فإن الـ **Database Triggers** يتم تنفيذها تلقائياً عندما نقوم بتعديل أو حذف أو إضافة أي سجل للجدول لذلك يمكننا الاستفادة منها في تنفيذ المهام التالية.

(1) زيادة وتقوية مستوى السرية والأمان في قاعدة البيانات حيث يمكننا مثلاً أن نضع جدولاً يحتوى على اسم المستخدم والتاريخ يتم تعبئته تلقائياً عندما يقوم أي مستخدم بعمل حذف أو تعديل على قاعدة البيانات وبهذا نستطيع أن نعرف من قام بالتعديل ومتى.

(2) عمل شروط للتحكم في الإدخال بحيث تكون متقدمة وبها شئ من التعقيد مثل أن نضع شرط إدخال في حقل يعتمد على قيمة حقل آخر في جدول ثاني (أو في نفس الجدول).

(3) إدخال قيم تلقائية في حقل أو أكثر من الجدول عند إدخال سجل جديد حيث يمكننا مثلاً إدخال قيمة رقم متسلسل (يمكن أن يكون مفتاح أساسي).

(4) عمل توثيق **Documentation** للتغيرات التي تحدث على الجدول وذلك عند تعديل أو حذف السجلات حيث يمكننا إنشاء جدول آخر يتم فيه تسجيل البيانات القديمة التي تم تعديلها أو حذفها.

مثال: زيادة مستوى السرية والأمان :-

في هذا المثال سوف نقوم بعمل **Trigger** مهمته تسجيل اسم أي مستخدم يقوم بعمل تعديل على جدول **Emp** كم يقوم الـ **Trigger** أيضاً بتسجيل تاريخ التعديل ورقم السجل الذي تم تعديله، ويتم تسجيل هذه البيانات في جدول جديد اسمه **Security** . لتنفيذ المثال التالي يجب علينا أن ننشئ الجدول الموضح أدناه:-

```
SQL> desc  security
 Name                          Null?    Type
 ----------------------------- -------- ----
 OLD_EMPNO                              NUMBER(4)
 MY_USER                                VARCHAR2(20)
 REC_DATE                               DATE
```

بعد ذلك قم بتنفيذ المثال التالي:-

```
SQL> create or replace trigger emp_security before
  2  update on emp
  3  for each row
  4  begin
  5  insert into security
  6  (old_empno,my_user,rec_date)
  7  values
  8  (:old.empno,user,sysdate);
  9  end;
 10  /

Trigger created.
```

بعد تنفيذ المثال أعلاه نقوم بتعديل أي سجل بجدول **EMP** كالتالي :-

```
SQL> update emp
  2  set sal = 1150
  3  where
  4  ename = 'FORD';

1 row updated.
```

بعد التعديل قم باستعراض محتويات الجدول **Security** كالتالي :-

```
SQL> SELECT * FROM SECURITY;

OLD_EMPNO      MY_USER        REC_DATE
------------------------ -------------------- ---------
    7902            SCOTT          07-FEB-00
```

سوف نلاحظ أن الـ **Trigger** قام بإضافة سجل تلقائي على جدول **Security** وأضاف فيه اسم المستخدم الذي قام بتعديل الجدول والتاريخ ورقم السجل المعدل.

المعاملين NEW و OLD:-

لاحظنا في المثال السابق أننا استخدمنا المعامل **OLD** وهو هنا يشير إلى القيمة القديمة (أي قيمة الحقل قبل التحديث أو الحذف) أما المعامل **NEW** فهو يشير إلى القيمة الجديدة (أي قيمة الحقل قبل التحديث أو القيمة الجديدة المراد إدخالها على الحقل)

صيغة استخدامهما في جملة شرط الـ Trigger :-

إذا أردنا استخدام المعامل **New** أو المعامل **Old** في جملة الشرط **Where** للازمة لل **Trigger** فيمكننا استخدامهما كالتالي :-

NEW **.** *field_name*
OLD **.** *field_name*

صيغة استخدامهما داخل جسم الـ Trigger :-

إذا أردنا استخدام المعامل **New** أو المعامل **Old** في داخل الـ **Trigger** مثل أن نضع عليهم شرط لمعرفة قيمة الحقل أو نريد أن نستخدمهم مع إحدى جمل **Insert or Update or Delete** فيجب أن نستخدم النقطتين فوق بعضهما قبل أن نكتب أي من المعاملين كالتالي :-

if **:**NEW **.** *field_name* = *value* then
if **:**OLD **.** *field_name* = *value* then

مثال: استخدام الـ Trigger في عمل تحكمات متقدمة على الحقول :-

في هذا المثال سوف ننشئ **Trigger** مهمته يحكم عدد محللي النظم **ANALYST** بالشركة بحيث لا يزيد عن (**2**) ، فعندما ندخل الوظيفة **Job = 'ANALYST'** يقوم الـ **Trigger** بحساب عدد محللي النظم الموجودين أصلاً بالجدول فإذا كان (**2**) فسوف يعطي رسالة خطاء.

```
SQL> create or replace trigger just_two_analist before
  2  insert or update on emp
  3  for each row
  4  when (new.job = 'ANALYST')
  5  declare
  6  no_of_analyst number(2);
  7  begin
  8  select count(*) into no_of_analyst from emp
  9  where
 10  job = 'ANALYST';
 11  if no_of_analyst >1 then
 12  raise_application_error(-20000,'you can not enter more than 2 analyst employee');
 13  end if;
 14  end;
 15  /

Trigger created.
```

بعد إنشاء الـ **Trigger** نقوم بمحاولة إدخال سجل جديد بحيث تكون الوظيفة محلل نظم **Analyst** كالتالي:-

```
SQL> insert into emp
  2  (empno,ename,job,sal)
  3  values
  4  (1111,'RXR','ANALYST',4500);
insert into emp
*
ERROR at line 1:
```

ORA-20000: you can not enter more than 2 analyst employee
ORA-06512: at "SCOTT.JUST_TWO_ANALIST", line 8
ORA-04088: error during execution of trigger 'SCOTT.JUST_TWO_ANALIST'

نلاحظ أن الأوراكل قام بإعطاء رسالة الخطاء والتي كتبناها في الـ **Trigger** . ولم يقوم بإضافة السجل إلى الجدول ويمكننا التأكد من ذلك باستعراض محتويات الجدول.

مثال: استخدام الـ Trigger في إضافة قيم تلقائية للجدول :-

في هذا المثال سوف نقوم بعمل **Trigger** يقوم بإدخال ترقيم تلقائي لرقم الموظف ، حيث يقوم بإيجاد قيمة آخر رقم لموظف تم إدخاله وإضافة واحد عليه كالتالي :-

```
SQL> create or replace trigger empno_counter before
  2  insert on emp
  3  for each row
  4  declare
  5  new_no number(4);
  6  last_no number(4);
  7  begin
  8  select max(empno) into last_no from emp;
  9  new_no := last_no + 1;
 10  :new.empno := new_no ;        ( هذا السطر يكفي لإدخال قيمة رقم الموظف في السجل الجديد )
 11  end;
 12  /

Trigger created.
```

بعد إنشاء الـ **Trigger** نقوم بإضافة سجل جديد دون أن ندخل رقم للموظف كالتالي:-

```
SQL> insert into emp
  2  (ename,job,sal)
  3  values
  4  ('AAAAAAA','AAAAAA',1000);

1 row created.
```

ثم بعد ذلك نقوم بإضافة سجل جديد آخر مع إضافة رقم الموظف وإعطائه أي رقم عشوائي كالتالي :-

```
SQL> insert into emp
  2  (empno,ename,job,sal)
  3  values
  4  (1111,'AAAAAAA','AAAAAA',1000);

1 row created.
```

بعد ذلك نقوم باستعراض محتويات جدول **Emp** كالتالي:-

```
SQL> select * from emp order by empno;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	1150		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10
7935	**AAAAAAA**	**AAAAAA**			**1000**		
7936	**AAAAAAA**	**AAAAAA**			**1000**		

16 rows selected.

من النتيجة أعلاه نلاحظ أنه تم إضافة السجلين بأرقام تسلسلية من آخر رقم مدخل (أكبر رقم) كما نلاحظ أنه تم إهمال قيمة رقم الموظف التي أدخلناها في المرة الثانية. ومن هنا يتضح أنه سواء أن أدخلنا قيمة لرقم الموظف

أو لم ندخل قيمة فسوف يقوم الـ **Trigger** بإدخال قيم تسلسلية لأرقام الموظفين.

مثال: استخدام الـ Trigger في إدخال قيم محسوبة :-

في هذا المثال سوف نقوم بعمل **Trigger** يتم تنفيذه إذا كانت وظيفة الموظف الجديد مندوب مبيعات ويقوم بحساب العمولة له بأن تكون **2.5**% من قيمة الراتب الأساسي له.

```
SQL> create or replace trigger sales_comm before
  2  insert or update on emp
  3  for each row
  4  when (new.job = 'SALESMAN')
  5  declare
  6  mem_comm number(8,2);
  7  begin
  8  if :new.sal is not null or :new.sal != 0 then
  9  mem_comm := (:new.sal * 2.5 / 100);
 10  :new.comm := mem_comm;
 11  end if;
 12  end;
 13  /
Trigger created.
```

بعد ذلك نقوم بإدخال سجلين جديدين بحيث لا ندخل لأحدهما قيمة الراتب **Sal** كما هو موضح أدناه

```
SQL> insert into emp
  2  (ename,job,sal)
  3  values
  4  ('BBBB','SALESMAN',3000);
1 row created.

SQL> insert into emp
  2  (ename,job)
  3  values
  4  ('BBBB','SALESMAN');

1 row created.
```

(يتبع)

بعد إدخال السجلين نقوم باستعراض محتويات جدول **Emp** كالتالي :-

```
SQL> select empno,ename,job,sal,comm from emp
  2  where job = 'SALESMAN';
   EMPNO     ENAME          JOB            SAL        COMM
---------  ----------    ---------      ---------  ---------
    7499     ALLEN       SALESMAN       1600        300
    7521     WARD        SALESMAN       1250        500
    7654     MARTIN      SALESMAN       1250        1400
    7844     TURNER      SALESMAN       1500        0
    7937     BBBB        SALESMAN       3000        75
    7938     BBBB        SALESMAN
6 rows selected.
```

نلاحظ أنه تم حساب قيمة العمولة **Comm** للموظف الذي أدخلنا له قيمة للراتب بينما تم تجاهل حساب العمولة للموظف الثاني الذي لم ندخل له قيمة الراتب. كما نلاحظ أنه تم إدخال رقم الموظف كترقيم تلقائي وذلك لأننا كنا قد أنشأنا **Trigger** في المثال السابق ليدخل الترقيم التلقائي ، ومن هنا نستخلص أنه يمكن لأكثر من **Trigger** أن يؤثر على الجدول.

بعد ذلك قم بتعديل سجل الموظف الذي لم ندخل له قيمة الراتب وندخل له قيمة للراتب كالتالي :-

```
SQL> update emp
  2  set sal = 4000
  3  where empno = 7938;
1 row updated.
```

ثم نستعرض محتويات الجدول **Emp** فنلاحظ أنه تم حساب قيمة العمولة **Comm** وذلك لأن الـ **Trigger** يتم تنفيذه عند الإدخال أو التعديل.

```
SQL> select empno,ename,job,sal,comm from emp
  2  where job = 'SALESMAN';

   EMPNO   ENAME        JOB            SAL        COMM
---------  ----------   ---------      ---------  ---------
    7499   ALLEN        SALESMAN       1600        300
    7521   WARD         SALESMAN       1250        500
    7654   MARTIN       SALESMAN       1250        1400
    7844   TURNER       SALESMAN       1500        0
    7937   BBBB         SALESMAN       3000        75
    7938   BBBB         SALESMAN       4000        100
6 rows selected.
```

مثال: استخدام الدوال أو الإجراءات المحفوظة مع الـ Trigger :-

في هذا المثال سوف نقوم بإنشاء نفس الـ **Trigger** السابق مع حساب قيمة العمولة وتحويلها بالدولار بدلاً من الريال ، ولإتمام هذه العملية سوف نقوم باستخدام الدالة **To_Dollar** والتي كنا قد أنشأناها سابقاً (راجع الدوال)

```
SQL> create or replace trigger sales_comm before
  2  insert or update on emp
  3  for each row
  4  when (new.job = 'SALESMAN')
  5  declare
  6  mem_comm number(8,2);
  7  begin
  8  if :new.sal is not null or :new.sal != 0 then
  9  mem_comm := (:new.sal * 2.5 / 100);
 10  :new.comm := to_dolar(mem_comm,3.75);
 11  end if;
 12  end;
 13  /
Trigger created.
```

بعد ذلك ندخل بيانات موظف جديد كالتالي :-

```
SQL> insert into emp
  2  (ename,job,sal)
  3  values
  4  ('CCCC','SALESMAN',3000);
1 row created.
```

ثم نستعرض محتويات الجدول **Emp** للتأكد من عمل الـ **Trigger** كالتالي :-

```
SQL> select empno,ename,job,sal,comm from emp
  2  where job = 'SALESMAN';
   EMPNO        ENAME      JOB          SAL        COMM
---------      ----------  ---------    ---------  ---------
    7499        ALLEN      SALESMAN      1600       300
    7521        WARD       SALESMAN      1250       500
    7654        MARTIN     SALESMAN      1250       1400
    7844        TURNER     SALESMAN      1500       0
    7937        BBBB       SALESMAN      3000       75
    7938        BBBB       SALESMAN      4000       100
    7939        CCCC       SALESMAN      3000       20
```

☑	لا يمكننا استخدام الدوال والإجراءات مع الـ Trigger إذا كانت تحتوي على الاوامر Commit أو Rollback

مثال: استخدام الـ Trigger في التوثيق :-

المقصود بالتوثيق هو متابعة التعديل والحذف الذي يحدث على الجدول وذلك بكتابة رقم السجل والقيم التي تم تعديلها (أي القيم القديمة قبل الحذف أو التعديل) إلى جدول جديد مع كتابة تاريخ التعديل أو الحذف. وفي المثال التالي سوف ننشئ جدول **Emp_history** ليتم فيه حفظ التعديلات التي تحدث على جدول **Emp** ويتم إنشاء الجدول كالتالي:-

```
Name                                      Type
-----------------------------------------  -------------------
EMPNO                                     NUMBER(4)
OLD_SAL                                   NUMBER(8,2)
OLD_COMM                                  NUMBER(8,2)
CHANGED_IN                                DATE
```

بعد ذلك ننشئ الـ **Trigger** الذي يقوم بإدخال البيانات القديمة في جدول **Emp_history** بحيث لا يتم تنفيذ الـ **Trigger** إلا عند حدوث تغيير على راتب الموظف أو العمولة. وهو موضح كالتالي:-

```
SQL> create or replace trigger emp_historical_log after
  2  update or delete on emp
  3  for each row
  4  when ((new.comm <> old.comm)
  5  or (new.comm <> old.comm)
  6  or (new.empno is null and old.empno is not null))
  7  begin
  8  insert into emp_history
  9  (empno,old_sal,old_comm,changed_in)
 10  values
 11  (:old.empno,:old.sal,:old.comm,sysdate);
 12  end;
 13  /

Trigger created.
```

(يعني إذا تم حذف السجل)

لتجربة الـ **Trigger** نقوم بحذف السجلات التالية ثم نستعرض محتويات جدول **Emp_history** كالتالي:-

```
SQL> delete from emp
  2  where
  3  ename = 'BBBB';
```

2 rows deleted.

```
SQL> select * from emp_history;

   EMPNO   OLD_SAL  OLD_COMM       CHANGED_IN
------- ---------- ---------- ---------- ----------
    7937       3000         75                 08-FEB-00
    7938       4000        100                 08-FEB-00
```

إدارة الـ Triggers :-

عندما نقوم بكتابة الـ **Database Triggers** حتماً سنحتاج لعمل الأشياء التالية :-

1. معرفة الـ **Triggers** الموجودة حالياً.
2. معرفة الكود لـ **Trigger** معين.
3. عدم تمكين أو تمكين **Trigger** من العمل.
4. حذف **Trigger**.

وللتعامل مع كل هذه الأشياء يوفر لنا الأوراكل جدول عروض **View** يتم فيه حفظ كل البيانات والمعلومات عن كل الـ **Triggers** للمستخدم. وهذا الـ **View** يعرف باسم **User_triggers** وتوصيفه كالتالي :-

```
SQL> desc user_triggers
 Name                                      Null?            Type
 ----------------------------------------- --------         ----------
 TRIGGER_NAME                              NOT NULL         VARCHAR2(30)
 TRIGGER_TYPE                                               VARCHAR2(16)
 TRIGGERING_EVENT                                           VARCHAR2(26)
 TABLE_OWNER                               NOT NULL         VARCHAR2(30)
 TABLE_NAME                                NOT NULL         VARCHAR2(30)
 REFERENCING_NAMES                                          VARCHAR2(87)
 WHEN_CLAUSE                                                VARCHAR2(2000)
 STATUS                                                     VARCHAR2(8)
 DESCRIPTION                                                VARCHAR2(2000)
 TRIGGER_BODY                                               LONG
```

مثال: معرفة الـ Triggers الموجودة حالياً والمرتبطة بجدول Emp :-

```
SQL> select trigger_name,triggering_event,trigger_type
  2  from user_triggers
  3  where table_name = 'EMP';

TRIGGER_NAME                   TRIGGERING_EVENT              TRIGGER_TYPE
------------------------------ ------------------------------ ----------------------------
EMPNO_COUNTER                  INSERT                         BEFORE EACH ROW
EMP_HISTORICAL_LOG             UPDATE OR DELETE               AFTER EACH ROW
EMP_SECURITY                   UPDATE                         BEFORE EACH ROW
JUST_TWO_ANALIST               INSERT OR UPDATE               BEFORE EACH ROW
SALES_COMM                     INSERT OR UPDATE               BEFORE EACH ROW
```

مثال: معرفة الكود لـ Trigger معين :-

قبل أن نستعرض الكود يفضل أن نستخدم الأوامر التالية :-

```
SET  LONG  50000
SET  HEADING  OFF

SQL> select when_clause,trigger_body from user_triggers
  2  where trigger_name = 'SALES_COMM';
```

عند تنفيذ الأمر أعلاه سوف تظهر النتيجة كالتالي :-

```
new.job = 'SALESMAN'
declare
mem_comm number(8,2);
begin
if :new.sal is not null or :new.sal != 0 then
mem_comm := (:new.sal * 2.5 / 100);
:new.comm := to_dolar(mem_comm,3.75);
end if;
end;
```

أما لمعرفة نوع الـ Trigger فنقوم بتنفيذ الأمر التالي :-

```
SQL> select description from user_triggers
  2  where trigger_name = 'SALES_COMM';
```

عند تنفيذ الأمر أعلاه سوف تظهر النتيجة كالتالي :-

```
sales_comm before
insert or update on emp
for each row
```

تمكين وعدم تمكين الـ Triggers :-

أحياناً كثيرة قد نحتاج لعدم تمكين الـ **Trigger** وذلك بجعله **Not Enable** ، وعادةً ما نحتاج هذا الأمر عندما يحدث أمر طارئ ونريد أن نتخطى عمل الـ **Trigger** . مع ملاحظة أن نرجعه إلى حالته الأولي فور الانتهاء من هذا الأمر الطارئ.

Syntax: -

ALTER TRIGGER *trigger_name* < ENABLED / DISABLED > ;

حذف الـ Triggers :-

Syntax: -

DROP TRIGGER *trigger_name* ;