تحدثنا في المحاضرات السابقة عن الـ oracle instance وكيف يقوم بعمليتي startup و shutdown وعن الـ parameters الخاصة به ، بعد ذلك انتقلنا للحديث عن الـ Tablespace والـ User حيث قمنا ببناء Tablespace ، Profile ، Role وأخيراً حساب user ثم القيام بـ connect عن طريقه .

أما في المحاضرة السابقة تحدثنا عن الـArchive Mode والـ NON-Archive Mode والفرق بينهما وكيفية عمل كل منهما وأخذ Backup عن الـ DB.

نستطيع في Oracle كما هو الحال في SQL بناء Object مثل Procedures , Functions , Triggers , Views ويتم ذلك باستخدام لغة PL/SQL (procedural language/SQL) وهي شبيهة جداً بـ SQL ولكن يضاف إليها بعض التعليمات البسيطة لتساعدنا في كتابة الكود البرمجي كـ if ، while ، ...

Procedures:

تعريف إجرائية يتطلب تحديد اسمها والـ input parameters و الـ out parameters و الـ body الخاص بها :

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

يمكن تعريف cursor (يحمل مجموعة من الـ rows اعيدت من قبل statement ما) ونقوم بمعالجة هذه الـ rows خلال الـ code .

```
DECLARE
   c_id customers.id%type;
   c_name customerS.No.ame%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

Functions:

الفرق بين التابع والإجرائية أن التابع يمكن استدعاءه ضمن SQL statement أما الإجرائية فلا يمكن ذلك

مثلاً يمكن القيام بـ .... Select function بينما لا يمكن ذلك في الإجرائية .

والـ function له return value.

.

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   < function_body >
END [function_name];
```

Packages:

خاصية غير موجودة في الـ SQL server وهي تحزيم مجموعة توابع وإجرائيات ضمن حزمة واحدة وهذه الحزمة لها header و body حيث تعرف ضمن الـ header مجموعة التوابع والإجرائيات ونكتب الـ code الخاص بهم ضمن الـ body.

وهي فقط عملية ترتيب للكود ونستدعي التابع عن طريق packageName.functionName

مثال على ذلك :

```
CREATE OR REPLACE PACKAGE BODY c_package AS
   PROCEDURE addCustomer(c_id  customers.id%type,
      c_name customerS.No.ame%type,
      c_age  customers.age%type,
      c_addr  customers.address%type,
      c_sal   customers.salary%type)
   IS
   BEGIN
      INSERT INTO customers (id,name,age,address,salary)
         VALUES(c_id, c_name, c_age, c_addr, c_sal);
   END addCustomer;

   PROCEDURE delCustomer(c_id   customers.id%type) IS
   BEGIN
      DELETE FROM customers
      WHERE id = c_id;
   END delCustomer;

   PROCEDURE listCustomer IS
   CURSOR c_customers is
      SELECT  name FROM customers;
   TYPE c_list is TABLE OF customerS.No.ame%type;
   name_list c_list := c_list();
   counter integer :=0;
   BEGIN
      FOR n IN c_customers LOOP
      counter := counter +1;
      name_list.extend;
      name_list(counter) := n.name;
      dbms_output.put_line('Customer(' ||counter|| ')'||name_list(counter));
      END LOOP;
   END listCustomer;

END c_package;
/
```

```
CREATE OR REPLACE PACKAGE c_package AS
   -- Adds a customer
   PROCEDURE addCustomer(c_id   customers.id%type,
   c_name  customerS.No.ame%type,
   c_age  customers.age%type,
   c_addr customers.address%type,
   c_sal  customers.salary%type);

   -- Removes a customer
   PROCEDURE delCustomer(c_id  customers.id%TYPE);
   --Lists all customers
   PROCEDURE listCustomer;

END c_package;
/
```

وتُستخدم كالتالي :

```
DECLARE
   code customers.id%type:= 8;
BEGIN
   c_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);
   c_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);
   c_package.listcustomer;
   c_package.delcustomer(code);
   c_package.listcustomer;
END;
/
```

Triggers:

هناك 3 أنواع لل Triggers :

1- على مستوى الـ User

2- على مستوى الـ Table

3- على مستوى الـ Database كاملةً

عند إنشاء Trigger فإنه يكون:

1- Before event

2- After event

3- Instead of event

و الـ event يمكن أن يكون :

1- DML Statement أي Database Manipulation مثل delete, insert, update

2- DDL Statement أي Database Definition مثل create, alter, drop

3- Database Operation مثل logon, logout, startup, shutdown

1. Trigger على مستوى الـ Table :

```
CREATE OR REPLACE TRIGGER <trigger_name>
<BEFORE | AFTER | INSTEAD OF> <ACTION> [OR <ACTION> OR <ACTION> [OF <column_name_list>]]
ON <table_name> | DATABASE |
FOR EACH ROW
DECLARE
        <variable definitions>
BEGIN
        <trigger_code>
EXCEPTION
         <exception clauses>
END <trigger_name>;
```

تذكرة : الـ trigger يختلف عن الإجرائية بانه ينقدح أوتوماتيكياً في المكان الذي قمت بتحديده.

a) Before

تعني ان الـ trigger سيقدح قبل العملية المحددة ونقوم ضمنه باختبار مجموعة constrains لم نستطع اختبارها ضمن check constrains.

تذكرة : يتم بناء check constrains ضمن الـ create table ونضع فيه شروط على الـ columns.

مثال على الـ check constrains :

لدينا حقل يأخذ محرف واحد فقط فـ نكتب check constrains نقوم بتسميته ولهذا الحقل قيمتين إما A أو B وأي قيمة خارجهما سيقوم الـ check constrains بإظهار error ويمكن القيام بمعالجة إضافية في هذا الـ check :

```
CREATE TABLE mytable (
        col1 NUMBER(3)
                CONSTRAINT mytable_pk PRIMARY KEY,
        col2 VARCHAR2(1)
                CONSTRAINT mytable_col2_nn NOT NULL
                CONSTRAINT mytable_col2_ch CHECK (col2 IN ('A', 'B', 'C')),
        col3 NUMBER(6)
                CONSTRAINT mytable_col3_nn NOT NULL
                CONSTRAINT mytable_col3_ch CHECK (col3 > 0)
);
```

مثال:

لدينا جدول employee وجدول رتبة وظيفية بينهما علاقة many to many ، ونريد زيادة رواتب الموظفين بحيث لكل رتبة وظيفية نسبة مئوية مختلفة للزيادة .

يتحقق ذلك عن طريق Trigger يعمل قبل تنفيذ insert الزيادة بحيث يتحقق من الرتبة الوظيفية ، ولا يمكن اختبار ذلك ضمن check constrains لأن الاختبار لا يتم على الـ row الحالي وإنما من جدول آخر عن طريق تنفيذ query فاذا كانت رتبته لا تتناسب مع الزيادة التي حصل عليها يقوم الـ trigger بـ raise للـ exception.

اذاً في الـ before نضع مجموعة validation constrains على مستوى الـ table وفي هذه الحالة اذا قام الـ trigger بـ raise فلن تتحقق عملية الـ insert لأن الـ DB ستقوم بعمل rollback.

b) After

لنفرض أنه عند القيام بـ insert في جدول ما فإن هذا الـ employee يحصل على زيادة بنسبة معينة في راتبه.

اذاً:

1- Before: عملية insert لم تنفذ بعد فيمكن التحقق من القيم ضمن الـ constrains و تغيير قيم الـ table .

2- After: نقوم بالكتابة في جدول آخر بعد التأكد من ان الـ insert صحيحة لكن لا يمكن تغيير قيم الـ table.

✓ For each row :

- الـ Trigger في SQL server : ينقدح لمرة واحدة فقط إذا كان الـ update على سطر واحد أو مليون سطر .
- الـ Trigger في Oracle : عند كتابة for each row فإنه ينقدح بعدد الـ rows كلها و بهذه الحالة يتوقف الـ trigger على مستوى الـ row لذلك يمكن التعديل فيه.
- في حالة الـ update يمكن تحديد حقول فيقدح الـ Trigger عند تعديلها UPDATE OF column_name .
- تعمل فقط مع DML event بالتالي الـ Trigger على مستوى Schema او DB لا يمكن كتابة for each row .

أمثلة:

نبني table اسمه mytable فيه ثلاث حقول ولدينا constrains بسيط على الحقل الثاني

```
CREATE TABLE mytable (
        col1 NUMBER(3)
                CONSTRAINT mytable_pk PRIMARY KEY,
        col2 VARCHAR2(1)
                CONSTRAINT mytable_col2_nn NOT NULL
                CONSTRAINT mytable_col2_ch CHECK (col2 IN ('A', 'B', 'C')),
        col3 NUMBER(6)
                CONSTRAINT mytable_col3_nn NOT NULL
                CONSTRAINT mytable_col3_ch CHECK (col3 > 0)
);
```

ونبني عليه trigger اسمه secure-mytable يمنع التعديل خارج أوقات الدوام

sysDate يعيد التاريخ الحالي فاذا كان اليوم هو جمعة او سبت أوالوقت خارج المجال من الساعة 8:30 صباحاً الى 3:30 فسيقوم بعمل raise فيمنع أي تعليمة insert او update او delete على هذا الـ table .

```
CREATE OR REPLACE TRIGGER secure_mytable
BEFORE INSERT OR UPDATE OR DELETE
ON mytable
DECLARE
 my_exc EXCEPTION;
BEGIN
  IF (TO_CHAR(SYSDATE, 'DY') IN ('FRI', 'SAT')) OR
  (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:30' AND '15:30') THEN
        RAISE my_exc;
  END IF;
EXCEPTION
  WHEN my_exc THEN
        RAISE_APPLICATION_ERROR(-20500, 'Changes only allowed during office
        hours' );
END; /
```

نبني sequence وهو بمثابة عداد للجدول Id ففي SQL server كنا نحدد حقل من نوع Identity ونعيطه قيمة الزيادة بينما في oracle نقوم بإنشاء sequence ونقوم بتسميته ونعطيه القيمة الابتدائية وأكبر قيمة وقيمة الزيادة .

```
CREATE SEQUENCE mytableseq
  MINVALUE 1
  MAXVALUE 999
  INCREMENT BY 1
  START WITH 1
  NOCACHE
  ORDER
  NOCYCLE;
```

- no cache لان الـ cache من الممكن أن يسبب gap مثلاً: اذا تم تحديد cache لـ 100 قيمة للـ sequence وكان قد وجد القيمة 160 على سبيل المثال فيصبح لديه caching في الـ memory 161,162...259 عند القيام بعمل shutdown ثم startup سيبدأ الـ sequence بالقيمة 260.
- order أي أن تكون القيم مرتبة لأنه من الممكن ان تكون عشوائية بدون order.
- cycle أن يعيد الترقيم بعد انتهائه حيث أن الـ sequence ليس primary هو فقط ترتيب للقيم وفي هذه الحالة (أن يعود الترقيم للبداية) يجب ان أضمن انتهاء المعالجة القيم السابقة قبل ان أرقم من جديد.

نبني trigger آخر قبل عملية الـ insert يقوم بأخذ القيمة التالية من الـ sequence واسنادها للعمود الأول من الجدول السابق كقيمة جديدة له .

```
CREATE OR REPLACE TRIGGER pk_mytable
BEFORE INSERT
ON mytable
FOR EACH ROW
BEGIN
 SELECT mytableseq.NEXTVAL
 INTO :NEW.col1
 FROM dual;
 dbms_output.put_line ('col1 = ' || :NEW.col1);
EXCEPTION
 WHEN OTHERS THEN
RAISE_APPLICATION_ERROR(-20111, 'Can not generate seq value');
END;
```

يتم أخذ القيمة التالية للـ sequence ووضعها في العمود الأول كقيمة جديدة

أيضاً نبني trigger آخر يحدد قيم العمود الثالث col3 حسب قيم العمود الاول col1 :

```
CREATE OR REPLACE TRIGGER ch_col3_mytable
BEFORE INSERT OR UPDATE
ON myuser.mytable
FOR EACH ROW
BEGIN
 IF (:NEW.col2 = 'A' AND
    :NEW.col3 NOT BETWEEN 50000 AND 100000) THEN
RAISE_APPLICATION_ERROR(-20101, 'col3 value must be between 50000 and 80000');
 ELSIF (:NEW.col2 = 'B' AND
        :NEW.col3 NOT BETWEEN 25000 AND 49999) THEN
RAISE_APPLICATION_ERROR(-20101, 'col3 value must be between 25000 and 49999');
 ELSIF (:NEW.col2 = 'C' AND
        :NEW.col3 NOT BETWEEN 0 AND 24999) THEN
RAISE_APPLICATION_ERROR(-20101, 'col3 value must be between 0 and 24999');
 END IF;
END;
```

ملاحظة هامة:

يجب ترقيم الـ exceptions التي يقوم الـ trigger بـ raise لها وإرفاقها برسالة لنستطيع معالجتها عند حدوثها ويبدأ الترقيم من السالب لأن الـ exceptions الموجبة تكون خاصة بـ oracle.

بعد إنشاء الـ Table والـ Triggers نقوم بعملية insert كالتالي :

```
INSERT INTO mytable (col2, col3) VALUES ('A', 70000);
```

في حال كان الوقت خارج أوقات الدوام سيقوم الـ trigger secure_mytable بعمل raise لـ exception .
نقوم بـ select * from my table فنلاحظ أن العمود الأول هو القيمة التي قام الـ sequence بإدخالها أوتوماتيكياً للعمود الأول ضمن الـ trigger pk_mytable

وبتنفيذ التعليمة التالية :

```
INSERT INTO mytable (col2, col3) VALUES ('D', -7);
```

نلاحظ وجود error check constrains لأن القيمة ليست A او B او C ولم يتم قدح trigger ch_col3_mytable ما يعني أنه أولا يتم التحقق من الـ check constrains ثم الـ trigger.

الـ triggers تفيد في الـ tracking حيث نقوم ببناء حساب ثاني مقابل الحساب الأصلي الذي نعمل عليه ونضع فيه نفس الـ tables وبداخلها نفس القيم ولكن نضيف اليه بعض الحقول الإضافية لنضع فيها التغيير الحاصل على الحقول الأساسية (تخزين Log لـ DB) .

ملاحظة 1:

لا يجب أن يكون الـ transaction كبير لكي لا يصبح الـ undo table كبير بدوره فإذا امتلأ لا يمكن افراغه إلا بعمل drop للـ DB وبنائها من جديد بعمل export لها وهذا العمل لا نقوم به إلا كل 6 اشهر او سنة .

ملاحظة 2:

الـ Data التي نضيفها الى Audit table (الجدول الخاص بالـ tracking) فقط في الـ update والـ delete أي فقط نحفظ التغيير أما في insert لا نضيف شيء

لتحقيق الـ tracking نقوم ببناء جدول مقابل للجدول السابق mytable يحوي القيم القديمة والجديدة للجدول ويحوي الأعمدة الإضافية .

```
CREATE TABLE audit_mytable (
  o_col1 NUMBER(3),
  o_col2 VARCHAR2(1),
  o_col3 NUMBER(6),
  n_col1 NUMBER(3),
  n_col2 VARCHAR2(1),
  n_col3 NUMBER(6),
  audit_seq NUMBER(6),
  audit_kind VARCHAR2(1),
  audit_date DATE
);
```

الأعمدة الإضافية .

audit_seq : رقم العملية على مستوى الـ DB

audit_kind : نوع العملية

audit_date : تاريخ العملية

ننشئ sequence على مستوى الـ DB بحيث يمثل عداد للعمليات على جميع الجداول :

```
CREATE SEQUENCE auditseq
  MINVALUE 1
  MAXVALUE 999999
  INCREMENT BY 1
  START WITH 1
  NOCACHE
  ORDER
  NOCYCLE;
```

وأخيراً ننشئ trigger والذي يتولى مهمة تخزين العمليات في جدول audit_mytable وذلك عند إجراء عملية على جدول mytable :

```
CREATE OR REPLACE TRIGGER audit_mytable_values
AFTER INSERT OR UPDATE OR DELETE
ON myuser.mytable
FOR EACH ROW
DECLARE
 v_audit_kind VARCHAR2(1);
 v_audit_seq   NUMBER(6);
BEGIN
 IF INSERTING THEN
  v_audit_kind := 'I';
 ELSIF UPDATING THEN
  v_audit_kind := 'U';
 ELSIF DELETING THEN
  v_audit_kind := 'D';
 END IF;

 SELECT myuser.auditseq.nextval
 INTO v_audit_seq
 FROM dual;

 INSERT INTO myuser.audit_mytable VALUES
 (:OLD.col1, :OLD.col2, :OLD.col3, :NEW.col1, :NEW.col2, :NEW.col3,
  v_audit_seq, v_audit_kind, SYSDATE);
END;
```

ملاحظة:

- وجود Constraints & Triggers أمر مهم وأفضل من المعالجة فقط في الـ Interface .
- من المهم القيام بتخزين Log للـ DB أي القيام بـ audit وذلك لحفظ المعلومات السابقة والعودة لها عند الحاجة .
- ليس من الضروري تخزين القيم الحديثة والقديمة فقد نكتفي بالقديمة فقط .
- علمية الـ insert لا داعي للقيام بـ audit لها .

2. Trigger على مستوى الـ DB :

نبني table لتخزين العمليات على الـ DB ومن يقوم بها والتاريخ:

```
CREATE TABLE ddl_log (
operation       VARCHAR2(30),
obj_owner       VARCHAR2(30),
obj_name        VARCHAR2(30),
attempt_by      VARCHAR2(30),
attempt_dt      DATE );
```

وسنبني trigger نوعه before drop, create حيث هنا نتعامل مع DDL حيث كل شخص قام بـ drop او create على مستوى الـ DB سنقوم بتسجيله ضمن DDL-log table الذي بنيناه.

```
CREATE OR REPLACE TRIGGER bcs_trigger
BEFORE CREATE OR DROP
ON DATABASE
BEGIN
 INSERT INTO ddl_log
 SELECT ora_sysevent, ora_dict_obj_owner,
 ora_dict_obj_name, USER, SYSDATE
 FROM DUAL;
END bcs_trigger;
```

ملاحظة:

يمكن بناء Trigger على عملية الـ connect (after login) .

انتهت المحاضرة

Written by :

Aisha Awaty

Wordpress and preparation :

Anas Alazmeh

Reviewed by :

Mouayyad Taja