Labsheet-9

(Prof. R. Gururaj)

SQL Cursor

A cursor is a special construct in PL/SQL used to hold data rows returned by SQL query.
It can be seen as a reserved area of memory in which the output of a query can be stored.
It held in the reserved area of DBMS at server.
Cursor commands:
Open
Fetch
Close
Attributes:

```
%rowcount   // returns number of rows fetched so far
%found       //  returns true if last fetch returned a row otherwise False
%notfound   //  returns true if last fetch did not return a row
%isopen  //  returns true if cursor is open
```

Example 1:
// Procedure to print book id and price in the format, using cursors
"For the book with id : XXX the price is :###"

```
SQL> create or replace procedure proc55 as
  2  id number;
  3  pr number;
  4  cursor book_cursor is
  5  select bid,price from book;
  6  begin
  7  open book_cursor;
  8  loop
  9  fetch book_cursor into id,pr;
 10  exit when book_cursor%notfound;
 11  dbms_output.put_line('For the book with id:  '||id||' the price is: '||pr);
 12  end loop;
 13  close book_cursor;
 14  end;
 15  /

Procedure created.

SQL> exec proc55;
For the book with id:  101 the price is: 333
For the book with id:  107 the price is: 800
For the book with id:  128 the price is: 175
For the book with id:  205 the price is: 230
For the book with id:  201 the price is: 800
```

Sequences in PLSQL

Sequence objects are used to generate keys automatically

```
SQL> create sequence bookid_seq start with 500;

Sequence created.

SQL> insert into book values(bookid_seq.nextval, ‘Biology’ , 650);  // bid=500

SQL> insert into book values(bookid_seq.nextval, ‘History’  , 800); // bid=501
```

PL-SQL Triggers

A *trigger* is a procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a data manipulation event.

1.A trigger is invoked before or after a data row is inserted, deleted or updated.

2.A trigger is associated with a database table.

3.Each table may have one or more triggers.

4.Triggers can be used to enforce constraints

5.Triggers can be used to insert/update records and to call stored procedures.

6.Used for auditing purpose (creating logs)

7.Generation of derived values.

Example-1

```
create trigger T1 after insert on book
begin
dbms_output.put_line('Inserted a new record into Book table');
end;
/

Trigger created.

SQL> insert into book values(201,'ECONOMICS',345);
Inserted a new record into Book table

1 row created.
```

Example:2

```
create trigger T2 after insert on Book
declare
```

```
  3  totalbooks number;
  4  begin
  5  select count(*) into totalbooks from book;
  6  dbms_output.put_line('Inserted a new record');
  7  dbms_output.put_line('   After new Entry Total number of books is
:'||totalbooks);
  8  end;
  9  /

Trigger created.

SQL> insert into book values(205,'STATS',230);
Inserted a new record
After new Entry Total number of books is :5
Inserted a new record into Book table

1 row created.
```

Example-3

```
SQL> create trigger T3 before insert on Book
  2   declare
  3  totalbooks number;
  4  begin
  5  select count(*) into totalbooks from book;
  6  dbms_output.put_line('   before  new Entry Total number of books is :'||totalbooks);
  7  end;
  8  /

Trigger created.
```

Now insert new record into book and see this.

What are Row-level and Table-level Triggers

Example-4

To demonstrate a Table-level Triggers

```
SQL> create or replace trigger T4 before update of price on Book
  2  begin
  3  dbms_output.put_line('update done:');
  4  end;
  5  /
Trigger created.
SQL> update book set price =600 where price>300;
update done:
2 rows updated.
```

Example 5: to demonstrate a row-level trigger

```
SQL> create or replace trigger T4 before update of price on Book
```

```
  2  for each row
  3  begin
  4  dbms_output.put_line('update done:');
  5  end;
  6  /

Trigger created.

SQL> update book set price =800 where price>300;
update done:
update done:
2 rows updated.
```

// Use of :old and :new

Example 6:

```
SQL> create or replace trigger T5 before update of price on Book
  2  for each row
  3  begin
  4  dbms_output.put_line('Old price: '||:old.price||'  new price is: '||:new.price);
  5  end;
  6  /

Trigger created.
```

```
SQL> update book set price=333 where price=222;
Old price: 222  new price is: 333
update done:

1 row updated.
```

Note that whenever you use ***:old*** or ***:new*** syntax it must be a row-level trigger.

For the book with id : XXX the price is :###"

For the book with id : 101 the price is :120

For the book with id : 107 the price is :1100

For the book with id : 128 the price is :110