```
REM CURSOR
1. 앞의 PL/SQL 예문에서는 처리 결과가 1개인 SELECT문을 다뤘다.
2. 하지만, 처리 결과가 여러 개의 행으로 구해지는 SELECT문을 처리하기 위해서는 커서가 필요하다.
3. SQL*Plus 툴 또는 응용프로그램에서 사용자가 실행한 SQL 문의 단위를 의미
4. 오라클 서버는 모든 문장을 Cursor 단위로 처리하고 그 정보를 저장관리한다.
5. 종류
   1) 암시적 커서(implicit cursor)
      - 일반적으로 사용되는 SQL 문, 한번 실행에 하나의 결과를 리턴하는 SQL문
     -오라클 내부에서 각각의 쿼리 결과에 접근하여 사용하기 위한 내부적 커서
      -모든 쿼리가 실행될 때마다 오픈됨
      -오라클 내부에서 접근하고 사용되는 커서이기 때문에 선언, 오픈 등의 작업을 할 필요가 없다
      -커서 이름을 알 수 없지만, 가장 최근에 실행된 SQL 문장에 대한 커서를 내부적으로 소유
      -이를 SQL 커서라하며, 'SQL'이라는 이름으로 접근 가능
       SELECT empno, ename
          INTO   :v_no, :v_ename
          FROM emp
          WHERE deptno = 10;

   2) 명시적 커서(explicit cursor)
      - SQL문을 실행했을 때 그 결과가 여러개인 경우에 암시적 커서를 사용하면
      - 에러가 발생한다. 왜냐하면, 암시적 커서에 사용되는 스칼라변수는 한 번에 하나의 값만을 저장하기 때문.
      - 이렇게 여러 개의 행이 리턴되는 질의문을 실행하는 경우에는 반드시 명시적 커서를 사용해야 한다.

      CURSOR c1 IS
      SELECT empno, ename
      FROM emp
      WHERE deptno = 20;
      OPEN c1;
      LOOP
         FETCH c1 INTO v_no, v_ename
      END LOOP ;
      CLOSE c1;

6. 명시적 커서 실행순서
   1) CURSOR 선언 --> OPEN   --> FETCH --> CLOSE
      -a. CURSOR 선언
            CURSOR cursor_name IS
            [SELECT 문장 ];
     -b. OPEN
            --질의를 수행하고 검색 조건을 충족하는 모든 행으로구성된 결과셋을 생성하기 위해 CURSOR를 OPEN 한다.
            --이제 결과 셋에서 첫번째 행을 가리킨다.
           OPEN cursor_name;
     -c. FETCH
            --결과 셋에서 ROW 단위로 데이터를 읽어 들인다.
            --각 인출(FETCH) 후에 CURSOR은 결과 셋에서 다음 행으로 이동한다.
            LOOP
            FETCH cursor_name INTO 변수;
               EXIT WHEN [처리내용];
            END LOOP ;
     -d. CLOSE
            --CURSOR를 사용할 수 없게 하고 결과 셋의 정의를 해제한다.
           CLOSE cursor_name;

7. Syntax
  DECLARE
     CURSOR cursor_name  IS statement;
   BEGIN
     OPEN cursor_name;
       FETCH cursor_name INTO variable_name;
    CLOSE cursor_name;
   END ;

--부서 테이블 조회하기
```

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE cursor_sample
IS
   v_dept   dept%ROWTYPE;
   CURSOR c1
   IS
      SELECT * FROM dept;
BEGIN
   DBMS_OUTPUT.PUT_LINE('부서번호 | 부서명 | 근무처');
   DBMS_OUTPUT.PUT_LINE('------------------------------ ');
   OPEN c1;

   --만일 LOOP를 사용하지 않으면
   --FETCH c1 INTO v_dept.deptno,  v_dept.dname,  v_dept.loc;
   --DBMS_OUTPUT.PUT_LINE('     ' || v_dept.deptno || '     ' || v_dept.dname || '      ' || v_dept.loc);
   --자동으로 CURSOR는 FETCH한 다음, 다음 ROW로 이동한다.

      LOOP
         FETCH c1 INTO v_dept.deptno,  v_dept.dname,  v_dept.loc;
         EXIT WHEN c1%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('     ' || v_dept.deptno || '     ' || v_dept.dname || '      ' || v_dept.
         loc);
      END LOOP ;
   CLOSE c1;
END ;
/

EXEC cursor_sample;




--부서번호를 부여받아 해당 사원들의 사번, 이름, 봉급을 출력하시오.
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE emp_process
(  t_deptno     IN  emp.deptno%TYPE)
IS
    v_empno       emp.empno%TYPE;
    v_ename       emp.ename%TYPE;
    v_sal          emp.sal%TYPE;
    CURSOR  emp_cursor
    IS
    SELECT empno, ename, sal
    FROM emp
    WHERE deptno = t_deptno;
BEGIN
    OPEN emp_cursor;
    LOOP
          FETCH emp_cursor INTO v_empno, v_ename, v_sal;
     EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;

     DBMS_OUTPUT.PUT_LINE(v_empno || '  ' || v_ename || '  ' || v_sal);
    END LOOP;
    CLOSE emp_cursor;
END ;
/

SQL>EXEC emp_process(20);

8. Cursor 와 FOR LOOP
   --OPEN ~ FETCH ~ CLOSE 없이 FOR ~ LOOP ~ END LOOP를 사용하여 간단하게 커서를 사용할 수 있다.
   --LOOP에서 각 반복마다 CURSOR를 열고 행을 인출(FETCH)하며 모든 행이 처리되면 자동으로 CURSOR가 CLOSE되기
   때문에 사용하기가 편리
```

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE emp_process
(  t_deptno     IN  emp.deptno%TYPE)
IS
      v_emp          emp%ROWTYPE;
      CURSOR  emp_cursor
      IS
      SELECT empno, ename, sal
      FROM emp
      WHERE deptno = t_deptno;
 BEGIN
      FOR  v_emp  IN emp_cursor LOOP
          EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(v_emp.empno || '  ' || v_emp.ename || '  ' || v_emp.sal);
      END LOOP;
 END ;
 /
 SQL>EXEC emp_process(10);
```

9. **Cursor** 의 상태
   1) **%**NOTFOUND **:** 커서 영역의 자료가 모두 **FETCH** 됐다면 **TRUE** 반환
   2) **%FOUND :** 커서 영역에 **FETCH** 되지 않을 자료가 남아있다면 **TRUE**
   3) **%**ISOPEN **:** 커서가 **OPEN** 상태이면 **TRUE**
   4) **%**ROWCOUNT **:** 커서가 얻어 **온** 레코드의 **갯**수

```
SET SERVEROUTPUT ON
DECLARE
     v_count  NUMBER;
BEGIN
     SELECT COUNT(*)
     INTO v_count
     FROM emp
     WHERE deptno = 10;
     DBMS_OUTPUT.PUT_LINE('Selected Count is ' || SQL%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE('Row Count is ' || v_count);
END ;
/
```

10. 매개변수와 커서
   1) 명시적 커서**를** 사용해서 여러 행의 결과**를** 가진 테이블을 검색하는 커서**를** 작성할 때
   2) 이 때 선언되는 **SELECT** 문이 반복적**으**로 재사용되어**야** 하는 경우 커서**를** 여러번 선언하게 된다**.**
   3) 프로그래밍을 복잡하게 하고**,** 향후 유지보수에도 어려움을 준다**.**
   4) 따**라**서**,** 이럴 때 반복적**으**로 실행될 수 있는 **SELECT** 문은 한번만 선언하고 경우에 따**라** 달**라**지는
   5) 조건 값을 실행할 때마다 다르게 설정하여 사용할 수 있다**.**
   6) 이럴 때 매개변수와 커서**를** 함께 사용하는 **방**법

```
CREATE OR REPLACE PROCEDURE emp_process
IS
   v_empno          emp.empno%TYPE;
   v_ename      emp.ename%TYPE;
   v_sal     NUMBER(7,2);
       CURSOR  emp_cursor(v_deptno   NUMBER)
   IS
       SELECT empno, ename, sal
       FROM emp
       WHERE deptno = v_deptno;
BEGIN
   OPEN  emp_cursor(10);
   LOOP
     FETCH  emp_cursor  INTO  v_empno, v_ename, v_sal;
     EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(v_empno || '    ' || v_ename || '    ' || v_sal);
```

```
      END LOOP;
      CLOSE emp_cursor;

      OPEN  emp_cursor(20);
      LOOP
         FETCH  emp_cursor  INTO  v_empno, v_ename, v_sal;
         EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE(v_empno || '    ' || v_ename || '    ' || v_sal);
      END LOOP;
      CLOSE emp_cursor;
       END ;
   /

  EXEC emp_process;

--Cursor 를 사용하지 않을 때의 우편번호 검색
--우편번호검색, 단 동이름은 역삼동으로만 할 것
   CREATE OR REPLACE PROCEDURE sp_zipcode_select
   (
      v_dong1   IN    zipcode.DONG%TYPE,
      v_zipcode  OUT zipcode.ZIPCODE%TYPE,
      v_sido   OUT zipcode.SIDO%TYPE,
      v_gugun   OUT zipcode.GUGUN%TYPE,
      v_dong OUT zipcode.DONG%TYPE,
      v_bunji OUT zipcode.BUNJI%TYPE
   )
   IS
   BEGIN
      SELECT zipcode.ZIPCODE, sido, gugun, dong, bunji
      INTO v_zipcode, v_sido, v_gugun, v_dong, v_bunji
      FROM ZIPCODE
      WHERE dong LIKE CONCAT(CONCAT('%', v_dong1),'%');
   END sp_zipcode_select;
   /

--Cursor 를 사용한 우편번호 검색
  SET SERVEROUTPUT ON
  CREATE OR REPLACE PROCEDURE sp_zipcode_select1
  (  t_dong     IN  zipcode.DONG%TYPE)
  IS
        v_zipcode          zipcode%ROWTYPE;
        CURSOR  zipcode_cursor
        IS
        SELECT zipcode, sido, gugun, dong, bunji
        FROM zipcode
        WHERE dong LIKE CONCAT(CONCAT('%', t_dong),'%');
   BEGIN
        FOR  v_zipcode  IN zipcode_cursor LOOP
            EXIT WHEN zipcode_cursor%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('(' || v_zipcode.zipcode || ')  ' || v_zipcode.sido || '  ' || v_zipcode.
         gugun || '  ' || v_zipcode.dong || '  ' || v_zipcode.bunji);
        END LOOP;
   END ;
   /
   SQL>EXEC sp_zipcode_select1('상하');

11. Using Ref Cursors To Return Recordsets
 --Since Oracle 7.3 the REF CURSOR type has been available to allow recordsets to be returned from
 stored procedures and functions. Oracle 9i introduced the predefined SYS_REFCURSOR type, meaning
 we no longer have to define our own REF CURSOR types.
   CREATE OR REPLACE PROCEDURE get_emp_rs
   (p_deptno    IN  emp.deptno%TYPE,
   p_recordset OUT SYS_REFCURSOR)
```

```
AS
BEGIN
   OPEN p_recordset FOR
   SELECT ename, empno, deptno
   FROM   emp
   WHERE  deptno = p_deptno
   ORDER BY ename;
END GetEmpRS;
/

SET SERVEROUTPUT ON SIZE 1000000
DECLARE
   l_cursor  SYS_REFCURSOR;
   l_ename   emp.ename%TYPE;
   l_empno   emp.empno%TYPE;
   l_deptno  emp.deptno%TYPE;
BEGIN
   get_emp_rs (p_deptno    => 30,
               p_recordset => l_cursor);

   LOOP
      FETCH l_cursor
      INTO  l_ename, l_empno, l_deptno;
      EXIT WHEN l_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(l_ename || ' | ' || l_empno || ' | ' || l_deptno);
   END LOOP;
   CLOSE l_cursor;
END;
/

import java.sql.*;
import oracle.jdbc.*;

public class TestResultSet  {
  public TestResultSet() {
    try {
      DriverManager.registerDriver (new oracle.jdbc.OracleDriver());
      Connection conn = DriverManager.getConnection("jdbc:oracle:oci:@w2k1", "scott", "tiger");
      CallableStatement stmt = conn.prepareCall("BEGIN get_emp_rs(?, ?); END;");
      stmt.setInt(1, 30); // DEPTNO
      stmt.registerOutParameter(2, OracleTypes.CURSOR); //REF CURSOR
      stmt.execute();
      ResultSet rs = ((OracleCallableStatement)stmt).getCursor(2);
      while (rs.next()) {
        System.out.println(rs.getString("ename") + ":" + rs.getString("empno") + ":" + rs.getString(
        "deptno"));
      }
      rs.close();
      rs = null;
      stmt.close();
      stmt = null;
      conn.close();
      conn = null;
    }
    catch (SQLException e) {
      System.out.println(e.getLocalizedMessage());
    }
  }

  public static void main (String[] args) {
    new TestResultSet();
  }
}
```

```
--우편번호검색
   CREATE OR REPLACE PROCEDURE getZipcode
   (v_dongName   IN      zipcode.dong%TYPE,
   out_cursor_zipcodes    OUT SYS_REFCURSOR
   )
   AS
   BEGIN
      OPEN out_cursor_zipcodes FOR
      SELECT zipcode, sido, gugun, dong, bunji
      FROM zipcode
      WHERE dong LIKE CONCAT(CONCAT('%', v_dongName), '%');
   END;

   import java.sql.CallableStatement;
   import java.sql.Connection;
   import java.sql.DriverManager;
   import java.sql.ResultSet;
   import java.sql.SQLException;

   import oracle.jdbc.OracleTypes;

   public class ZipSearch1 {
      private static final String DBDRIVER;
      private static final String DBURL;
      private static final String DBUSER;
      private static final String DBPWD;
      private Connection conn;
      private CallableStatement cstmt;
      private ResultSet rs;

      static{
         DBDRIVER = "oracle.jdbc.driver.OracleDriver";
         DBURL = "jdbc:oracle:thin:@192.168.110.128:1521:orcl";
         DBUSER = "scott";
         DBPWD = "tiger";
      }
      private void loadDriver(){
         try{
            Class.forName(DBDRIVER);
         }catch(ClassNotFoundException ex){
            System.out.println("Driver Not Loaded.");
         }
      }
      private void dbConnect(){
         try{
            this.conn = DriverManager.getConnection(DBURL,DBUSER, DBPWD);
         }catch(SQLException ex){
            System.out.println("Connection Failure");
         }
      }
      private void createStatement(){
         try{
            this.cstmt = this.conn.prepareCall("{call getZipcode(?,?)}");
            this.cstmt.setString(1, "역삼");
            this.cstmt.registerOutParameter(2, OracleTypes.CURSOR);
         }catch(SQLException ex){
            ex.printStackTrace();
         }
      }
      private void runStatement(){
         try{
            this.cstmt.execute();
```

```
            this.rs = (ResultSet) cstmt.getObject(2);
            if(!rs.next()){
               System.out.println("Data Not Found");
               return;
            }
            do{
                  System.out.printf("(%s) %s %s %s %s\n",
                        this.rs.getString("zipcode"), this.rs.getString("sido"),
                              this.rs.getString("gugun"), this.rs.getString("dong"),
                              this.rs.getString("bunji"));
            }while(rs.next());
         }catch(SQLException ex){
            ex.printStackTrace();
         }
      }
      private void dbClose(){
         try{
            if(this.rs != null) this.rs.close();
            if(this.cstmt != null) this.cstmt.close();
            if(this.conn != null) this.conn.close();
         }catch(SQLException ex){
            ex.printStackTrace();
         }
      }
      public static void main(String[] args) {
         ZipSearch1 zip = new ZipSearch1();
         zip.loadDriver();
         zip.dbConnect();
         zip.createStatement();
         zip.runStatement();
         zip.dbClose();
      }
   }

```