

```

1  REM CURSOR
2  1. 앞의 PL/SQL 예문에서는 처리 결과가 1개인 SELECT문을 다뤘다.
3  2. 하지만, 처리 결과가 여러 개의 행으로 구해지는 SELECT문을 처리하기 위해서는 커서가 필요하다.
4  3. SQL*Plus 툴 또는 응용프로그램에서 사용자가 실행한 SQL 문의 단위를 의미
5  4. 오라클 서버는 모든 문장을 Cursor 단위로 처리하고 그 정보를 저장관리한다.
6  5. 종류
7  1) 암시적 커서(implicit cursor)
8      - 일반적으로 사용되는 SQL 문, 한번 실행에 하나의 결과를 리턴하는 SQL문
9      -오라클 내부에서 각각의 쿼리 결과에 접근하여 사용하기 위한 내부적 커서
10     -모든 쿼리가 실행될 때마다 오픈됨
11     -오라클 내부에서 접근하고 사용되는 커서이기 때문에 선언, 오픈 등의 작업을 할 필요가 없다
12     -커서 이름을 알 수 없지만, 가장 최근에 실행된 SQL 문장에 대한 커서를 내부적으로 소유
13     -이를 SQL 커서라하며, 'SQL'이라는 이름으로 접근 가능
14     SELECT empno, ename
15         INTO :v_no, :v_ename
16         FROM emp
17         WHERE deptno = 10;
18
19  2) 명시적 커서(explicit cursor)
20     - SQL문을 실행했을 때 그 결과가 여러개인 경우에 암시적 커서를 사용하면
21     - 에러가 발생한다. 왜냐하면, 암시적 커서에 사용되는 스칼라변수는 한 번에 하나의 값만을 저장하기 때문.
22     - 이렇게 여러 개의 행이 리턴되는 질의문을 실행하는 경우에는 반드시 명시적 커서를 사용해야 한다.
23
24     CURSOR c1 IS
25     SELECT empno, ename
26     FROM emp
27     WHERE deptno = 20;
28     OPEN c1;
29     LOOP
30         FETCH c1 INTO v_no, v_ename
31     END LOOP ;
32     CLOSE c1;
33
34  6. 명시적 커서 실행순서
35  1) CURSOR 선언 --> OPEN --> FETCH --> CLOSE
36      -a. CURSOR 선언
37          CURSOR cursor_name IS
38          [SELECT 문장 ];
39      -b. OPEN
40          --질의를 수행하고 검색 조건을 충족하는 모든 행으로구성된 결과셋을 생성하기 위해 CURSOR를 OPEN 한다.
41          --이제 결과 셋에서 첫번째 행을 가리킨다.
42          OPEN cursor_name;
43      -c. FETCH
44          --결과 셋에서 ROW 단위로 데이터를 읽어 들인다.
45          --각 인출(FETCH) 후에 CURSOR은 결과 셋에서 다음 행으로 이동한다.
46          LOOP
47              FETCH cursor_name INTO 변수;
48              EXIT WHEN [처리내용];
49          END LOOP ;
50      -d. CLOSE
51          --CURSOR를 사용할 수 없게 하고 결과 셋의 정의를 해제한다.
52          CLOSE cursor_name;
53
54  7. Syntax
55  DECLARE
56      CURSOR cursor_name IS statement;
57  BEGIN
58      OPEN cursor_name;
59      FETCH cursor_name INTO variable_name;
60      CLOSE cursor_name;
61  END ;
62
63  --부서 테이블 조회하기

```

```

64 SET SERVEROUTPUT ON
65 CREATE OR REPLACE PROCEDURE cursor_sample
66 IS
67     v_dept dept%ROWTYPE;
68     CURSOR c1
69     IS
70         SELECT * FROM dept;
71 BEGIN
72     DBMS_OUTPUT.PUT_LINE('부서번호 | 부서명 | 근무처');
73     DBMS_OUTPUT.PUT_LINE('-----');
74     OPEN c1;
75
76     --만일 LOOP를 사용하지 않으면
77     --FETCH c1 INTO v_dept.deptno, v_dept.dname, v_dept.loc;
78     --DBMS_OUTPUT.PUT_LINE(' ' || v_dept.deptno || ' ' || v_dept.dname || ' ' || v_dept.loc);
79     --자동으로 CURSOR는 FETCH한 다음, 다음 ROW로 이동한다.
80
81     LOOP
82         FETCH c1 INTO v_dept.deptno, v_dept.dname, v_dept.loc;
83         EXIT WHEN c1%NOTFOUND;
84         DBMS_OUTPUT.PUT_LINE(' ' || v_dept.deptno || ' ' || v_dept.dname || ' ' || v_dept.
            loc);
85     END LOOP ;
86     CLOSE c1;
87 END ;
88 /
89
90 EXEC cursor_sample;
91
92
93
94
95 --부서번호를 부여받아 해당 직원들의 사번, 이름, 봉급을 출력하시오.
96 SET SERVEROUTPUT ON
97 CREATE OR REPLACE PROCEDURE emp_process
98 ( t_deptno IN emp.deptno%TYPE)
99 IS
100     v_empno emp.empno%TYPE;
101     v_ename emp.ename%TYPE;
102     v_sal emp.sal%TYPE;
103     CURSOR emp_cursor
104     IS
105         SELECT empno, ename, sal
106         FROM emp
107         WHERE deptno = t_deptno;
108 BEGIN
109     OPEN emp_cursor;
110     LOOP
111         FETCH emp_cursor INTO v_empno, v_ename, v_sal;
112         EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
113
114         DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_ename || ' ' || v_sal);
115     END LOOP;
116     CLOSE emp_cursor;
117 END ;
118 /
119
120 SQL>EXEC emp_process(20);
121
122 8. Cursor 와 FOR LOOP
123 --OPEN ~ FETCH ~ CLOSE 없이 FOR ~ LOOP ~ END LOOP를 사용하여 간단하게 커서를 사용할 수 있다.
124 --LOOP에서 각 반복마다 CURSOR를 열고 행을 인출(FETCH)하며 모든 행이 처리되면 자동으로 CURSOR가 CLOSE되기
    때문에 사용하기가 편리

```

```

125 SET SERVEROUTPUT ON
126 CREATE OR REPLACE PROCEDURE emp_process
127 ( t_deptno IN emp.deptno%TYPE)
128 IS
129     v_emp emp%ROWTYPE;
130     CURSOR emp_cursor
131     IS
132     SELECT empno, ename, sal
133     FROM emp
134     WHERE deptno = t_deptno;
135 BEGIN
136     FOR v_emp IN emp_cursor LOOP
137         EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
138         DBMS_OUTPUT.PUT_LINE(v_emp.empno || ' ' || v_emp.ename || ' ' || v_emp.sal);
139     END LOOP;
140 END ;
141 /
142 SQL>EXEC emp_process(10);
143
144

```

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 갯수

```

151 SET SERVEROUTPUT ON
152 DECLARE
153     v_count NUMBER;
154 BEGIN
155     SELECT COUNT(*)
156     INTO v_count
157     FROM emp
158     WHERE deptno = 10;
159     DBMS_OUTPUT.PUT_LINE('Selected Count is ' || SQL%ROWCOUNT);
160     DBMS_OUTPUT.PUT_LINE('Row Count is ' || v_count);
161 END ;
162 /
163

```

10. 매개변수와 커서

- 1) 명시적 커서를 사용해서 여러 행의 결과를 가진 테이블을 검색하는 커서를 작성할 때
- 2) 이 때 선언되는 SELECT 문이 반복적으로 재사용되어야 하는 경우 커서를 여러번 선언하게 된다.
- 3) 프로그래밍을 복잡하게 하고, 향후 유지보수에도 어려움을 준다.
- 4) 따라서, 이럴 때 반복적으로 실행될 수 있는 SELECT 문은 한번만 선언하고 경우에 따라 달라지는
- 5) 조건 값을 실행할 때마다 다르게 설정하여 사용할 수 있다.
- 6) 이럴 때 매개변수와 커서를 함께 사용하는 방법

```

172 CREATE OR REPLACE PROCEDURE emp_process
173 IS
174     v_empno emp.empno%TYPE;
175     v_ename emp.ename%TYPE;
176     v_sal NUMBER(7,2);
177     CURSOR emp_cursor(v_deptno NUMBER)
178     IS
179     SELECT empno, ename, sal
180     FROM emp
181     WHERE deptno = v_deptno;
182 BEGIN
183     OPEN emp_cursor(10);
184     LOOP
185         FETCH emp_cursor INTO v_empno, v_ename, v_sal;
186         EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
187         DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_ename || ' ' || v_sal);

```

```

188     END LOOP;
189     CLOSE emp_cursor;
190
191     OPEN emp_cursor(20);
192     LOOP
193         FETCH emp_cursor INTO v_empno, v_ename, v_sal;
194         EXIT WHEN emp_cursor%ROWCOUNT > 5 OR emp_cursor%NOTFOUND;
195         DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_ename || ' ' || v_sal);
196     END LOOP;
197     CLOSE emp_cursor;
198     END ;
199 /
200
201 EXEC emp_process;
202
203 --Cursor 를 사용하지 않을 때의 우편번호 검색
204 --우편번호검색, 단 동이름은 역삼동으로만 할 것
205 CREATE OR REPLACE PROCEDURE sp_zipcode_select
206 (
207     v_dong1 IN zipcode.DONG%TYPE,
208     v_zipcode OUT zipcode.ZIPCODE%TYPE,
209     v_sido OUT zipcode.SIDO%TYPE,
210     v_gugun OUT zipcode.GUGUN%TYPE,
211     v_dong OUT zipcode.DONG%TYPE,
212     v_bunji OUT zipcode.BUNJI%TYPE
213 )
214 IS
215 BEGIN
216     SELECT zipcode.ZIPCODE, sido, gugun, dong, bunji
217     INTO v_zipcode, v_sido, v_gugun, v_dong, v_bunji
218     FROM ZIPCODE
219     WHERE dong LIKE CONCAT(CONCAT('%', v_dong1), '%');
220 END sp_zipcode_select;
221 /
222
223 --Cursor 를 사용한 우편번호 검색
224 SET SERVEROUTPUT ON
225 CREATE OR REPLACE PROCEDURE sp_zipcode_select1
226 ( t_dong IN zipcode.DONG%TYPE)
227 IS
228     v_zipcode zipcode%ROWTYPE;
229     CURSOR zipcode_cursor
230     IS
231     SELECT zipcode, sido, gugun, dong, bunji
232     FROM zipcode
233     WHERE dong LIKE CONCAT(CONCAT('%', t_dong), '%');
234 BEGIN
235     FOR v_zipcode IN zipcode_cursor LOOP
236         EXIT WHEN zipcode_cursor%NOTFOUND;
237         DBMS_OUTPUT.PUT_LINE('(' || v_zipcode.zipcode || ') ' || v_zipcode.sido || ' ' || v_zipcode.
            gugun || ' ' || v_zipcode.dong || ' ' || v_zipcode.bunji);
238     END LOOP;
239 END ;
240 /
241 SQL>EXEC sp_zipcode_select1('상하');
242
243 11. Using Ref Cursors To Return Recordsets
244 --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.
245 CREATE OR REPLACE PROCEDURE get_emp_rs
246 (p_deptno IN emp.deptno%TYPE,
247  p_recordset OUT SYS_REFCURSOR)

```

```

248 AS
249 BEGIN
250     OPEN p_recordset FOR
251     SELECT ename, empno, deptno
252     FROM emp
253     WHERE deptno = p_deptno
254     ORDER BY ename;
255 END GetEmpRS;
256 /
257
258 SET SERVEROUTPUT ON SIZE 1000000
259 DECLARE
260     l_cursor SYS_REFCURSOR;
261     l_ename emp.ename%TYPE;
262     l_empno emp.empno%TYPE;
263     l_deptno emp.deptno%TYPE;
264 BEGIN
265     get_emp_rs (p_deptno => 30,
266                p_recordset => l_cursor);
267
268     LOOP
269         FETCH l_cursor
270         INTO l_ename, l_empno, l_deptno;
271         EXIT WHEN l_cursor%NOTFOUND;
272         DBMS_OUTPUT.PUT_LINE(l_ename || ' | ' || l_empno || ' | ' || l_deptno);
273     END LOOP;
274     CLOSE l_cursor;
275 END;
276 /
277
278 import java.sql.*;
279 import oracle.jdbc.*;
280
281 public class TestResultSet {
282     public TestResultSet() {
283         try {
284             DriverManager.registerDriver (new oracle.jdbc.OracleDriver());
285             Connection conn = DriverManager.getConnection("jdbc:oracle:oci:@w2k1", "scott", "tiger");
286             CallableStatement stmt = conn.prepareCall("BEGIN get_emp_rs(?, ?); END;");
287             stmt.setInt(1, 30); // DEPTNO
288             stmt.registerOutParameter(2, OracleTypes.CURSOR); //REF CURSOR
289             stmt.execute();
290             ResultSet rs = ((OracleCallableStatement)stmt).getCursor(2);
291             while (rs.next()) {
292                 System.out.println(rs.getString("ename") + ":" + rs.getString("empno") + ":" + rs.getString(
293                     "deptno"));
294             }
295             rs.close();
296             rs = null;
297             stmt.close();
298             stmt = null;
299             conn.close();
300             conn = null;
301         } catch (SQLException e) {
302             System.out.println(e.getLocalizedMessage());
303         }
304     }
305
306     public static void main (String[] args) {
307         new TestResultSet();
308     }
309 }

```

```

310
311 --우편번호검색
312 CREATE OR REPLACE PROCEDURE getZipcode
313 (v_dongName IN zipcode.dong%TYPE,
314 out_cursor_zipcodes OUT SYS_REFCURSOR
315 )
316 AS
317 BEGIN
318 OPEN out_cursor_zipcodes FOR
319 SELECT zipcode, sido, gugun, dong, bunji
320 FROM zipcode
321 WHERE dong LIKE CONCAT(CONCAT('%', v_dongName), '%');
322 END;
323
324 import java.sql.CallableStatement;
325 import java.sql.Connection;
326 import java.sql.DriverManager;
327 import java.sql.ResultSet;
328 import java.sql.SQLException;
329
330 import oracle.jdbc.OracleTypes;
331
332 public class ZipSearch1 {
333     private static final String DBDRIVER;
334     private static final String DBURL;
335     private static final String DBUSER;
336     private static final String DBPWD;
337     private Connection conn;
338     private CallableStatement cstmt;
339     private ResultSet rs;
340
341     static{
342         DBDRIVER = "oracle.jdbc.driver.OracleDriver";
343         DBURL = "jdbc:oracle:thin:@192.168.110.128:1521:orcl";
344         DBUSER = "scott";
345         DBPWD = "tiger";
346     }
347     private void loadDriver(){
348         try{
349             Class.forName(DBDRIVER);
350         }catch(ClassNotFoundException ex){
351             System.out.println("Driver Not Loaded.");
352         }
353     }
354     private void dbConnect(){
355         try{
356             this.conn = DriverManager.getConnection(DBURL,DBUSER, DBPWD);
357         }catch(SQLException ex){
358             System.out.println("Connection Failure");
359         }
360     }
361     private void createStatement(){
362         try{
363             this.cstmt = this.conn.prepareCall("{call getZipcode(?,?)}");
364             this.cstmt.setString(1, "역삼");
365             this.cstmt.registerOutParameter(2, OracleTypes.CURSOR);
366         }catch(SQLException ex){
367             ex.printStackTrace();
368         }
369     }
370     private void runStatement(){
371         try{
372             this.cstmt.execute();

```

```

373         this.rs = (ResultSet) pstmt.getObject(2);
374         if(!rs.next()){
375             System.out.println("Data Not Found");
376             return;
377         }
378         do{
379             System.out.printf("(%s) %s %s %s %s\n",
380                 this.rs.getString("zipcode"), this.rs.getString("sido"),
381                 this.rs.getString("gugun"), this.rs.getString("dong"),
382                 this.rs.getString("bunji"));
383         }while(rs.next());
384     }catch(SQLException ex){
385         ex.printStackTrace();
386     }
387 }
388 private void dbClose(){
389     try{
390         if(this.rs != null) this.rs.close();
391         if(this.pstmt != null) this.pstmt.close();
392         if(this.conn != null) this.conn.close();
393     }catch(SQLException ex){
394         ex.printStackTrace();
395     }
396 }
397 public static void main(String[] args) {
398     ZipSearch1 zip = new ZipSearch1();
399     zip.loadDriver();
400     zip.dbConnect();
401     zip.createStatement();
402     zip.runStatement();
403     zip.dbClose();
404 }
405 }
406

```