[보충파일] JDBC를 파해쳐 보자...
1) 매개변수를 받아서 한 행만 출력하는 문장을 연습해보자
문제 사례) 부서 번호를 매개변수로 받아서 부서에 대한 정보들을 모두 출력해보자.

```
public static Department selectByDepartment_Id(int department_id) throws Exception{
        String sql="SELECT * FROM DEPARTMENTS WHERE DEPARTMENT_ID = ?";
```

-> 우리가 평소에 데이터베이스에서 조건을 입력받을 때 Where문 후반야에 조건문을 넣어야 한다. 만일 매개변수가 필요로 한 경우에는 Where 문 후반야에 ?를 넣어주거나, Insert 문 Values에 ?를 넣어주는 사례를 들 수 있지만, 자세한 부분은 DML부분에서 설명하겠다.

```
        Connection connect=null;
```

-> Connection은 데이터베이스와 자바 사이를 연결해주는 연결고리 역할의 객체로 생각하면 된다.

```
        PreparedStatement state=null;
```

-> Statement는 Statement에 상속된 PreparedStatement, CallableStatement 2가지로 나뉘게 되는데 우선 우리가 매개변수를 받게 된다면 죽었다 깨어나도 PreparedStatement를 써줘야 한다. 그렇지만 매개변수를 받지 않게 되면 일반적으로 Statement를 써도 크게 상관없다.
또한 PL/SQL 문장을 이용한 프로시저(Procedure)나 함수(Function)를 JDBC를 통해서 호출하고 싶다면 CallableStatement를 이용해주면 된다.

```
        ResultSet result=null;
```

-> ResultSet는 String 형 변수인 sql라는 녀석이 우리가 흔히 쓰는 SQL 문장이 되겠는데 이를 Statement에 올리고 난 후 실행을 하면 얻을 결과들에 대해서 저장을 하는데 있어서 필요한 객체가 있는데 이 녀석이 바로 ResultSet로 볼 수 있겠다.

```
        Department dep=null;
```

-> Department라는 객체는 우리가 테이블을 만들게 되는데 각 컬럼들의 요소를 유두리있게 인스턴스 변수로 저장을 해주고 난 후 getter/setter를 따로 형성해서 이용하면 되겠다. 가령 Departments 테이블의 구조를 예를 들면 department_id는 int형 변수, department_name은 String 형 변수 등으로 인스턴스로 형성해서 각 생성자인 getDepartment_id(), setDepartment_id() 등등을 만들어서 써주면 되겠다.

```
        try{ // JDBC의 생명은 예외이다. 가령 접속이 안 되거나 오라클 내부에서 문제가 발생하는 경우에 따로 처리하는 메소드를 구분해서 써야 하는 것이 특징인데 예외가 걸리게 되면 DAO.java 파일에 있는 printError 메소드를 불러와서 그 예외 번호에 따른 에러 이름을 출력하도록 한다.
                connect=DB.getConnection();
```

-> 우리가 연결하는 객체인 Connection의 인스턴스 변수 connect를 통해서 실질적으로 properties 파일을 통해 데이터베이스를 연결해주는 util 패키지 내에 있는 DB.java 파일에 있는 getConnection 클래스를 통해서 자바와 데이터베이스를 연결할 준비를 한다. 자세히 이야기를 하면 Database와 Java 사이를 연결해주는 드라이버 로딩으로 살펴보면 되겠다. 이는 필자가 설명한 (3-2)를 참조하면 되겠다.

```
                state=connect.prepareStatement(sql);
```

// 우리가 String 형 변수 sql에 작성한 문장들을 Connection 인스턴스 변수 connect 내에서 불러 올 수 있는 prepareStatement 메소드를 통해서 저장시켜준다. prepareStatement 메소드는 PreparedStatement 객체와 햇깔릴 수 있는데 PreparedStatement는 준비된 Statement에 대해 저장을 하는 것이고, prepareStatement는 Statement에 대해서 SQL문장을 적재하고 Statement를 준비하게끔 해주는 메소드로 보면 된다. 그래서 준비를 마친 Statement에 대해서 PreparedStatement의 인스턴스 변수인 state에 저장을 해서 나중에는 준비된 Statement를 저장하는 역할로 생각하면 되겠다.

```
                state.setInt(1, department_id);
```

-> state 인스턴스 변수 내에 있는 setInt 메소드랑 setString 메소드가 존재하는데 우리가 매개변수를 실제 실행할 메소드에 대해 받았다고 친다면, SQL 문장에 분명히 어디엔가 적용을 시켜야 한다. 그래서 방금 ?에 작성한 곳에 적용을 할 수 있도록 첫 번째(1) 물음표에 department_id를 저장하는 것이다. 주의할 점이 있다. 실제로 sql에 쓰인 물음표의 순서는 첫 번째부터 시작이 된다. 0번째가 아니라는 뜻이다.

```
                result=state.executeQuery();
```

-> state에서 적용된 SQL 문장에 대해 결과를 실행하려면 우리가 프로시저나 함수 등등을 불러올 때 많이 살펴본 execute라는 녀석이 있다. executeQuery, executeUpdate 등으로 나뉠수가 있는데 Select문 같은 경우에는 executeQuery를 쓰게 되고, DML 문장(Update, Insert, Delete)에 대해선 executeUpdate를 쓰게 된다.

```
                // department_id가 primary key이므로, 결과가 하나만 생성될 것을 알고 있다.
                // 따라서, 반복문이 아닌 if문을 사용하였다.
                // 질의를 수행하면, cursor는 첫 번째 결과행 이전을 가리키고 있다.
                // 첫 번째 next를 수행하면 첫 번째 결과행을 반환한다.
                if(result.next()){
```

-> department_id가 Primary Key라는 사실을 안다면 분명히 결과는 0~1개 중에는 나올 것이다. 그래서 하나의 행을 뽑아낼 때에는 if문으로 무난하게 작성하면 되겠다. ResultSet의 메소드인 next()는 신해철의 그거 말고 결과 값들을 출력하는데 있어서 다음 값들이 존재하는지에 대한 boolean 대수로 살펴보면 되겠다.

```
                        dep=makeDepartment(result);
```

-> 이로써 result를 makeDepartment 메소드에 넣어서 Department 인스턴스 변수인 dep에 저장을 하게 되는데 makeDepartment 메소드는 어떻게 구성되어 있는지에 대하여 아래에서 살펴보도록 하자.

```
                }
                else{
                        NotRecordFoundException e=new NotRecordFoundException();
                        throw e;
```
-> 만일 결과가 아무 것도 나오지 않는다면 분명히 예외를 던져줘야 한다. PL/SQL의 예외에서 살펴보면 NO_DATA_FOUND와 같은 역할을 하는데 여기서는 NotRecordFoundException이라는 예외를 통해서 문장이 이상 없이 돌아가게끔 한다. 예외 주제에 이름도 드럽게 긴데 그냥 된장찌개에 밥 말아 먹는 기분으로만 살펴주면 되겠다.
```
                }
        }
        catch(SQLException e){
                printError(e);
```
-> 만일 문장을 작성하다 보면 어딘가에 예외가 꼭 걸리는 법이다. 예를 들어 부서 번호를 430으로 쳤다가 결과는 아무 것도 안 뜨는 경우라든지, 아니면 숫자로 매개변수를 받아야 하는데 영어로 입력했든지... 이런 경우가 존재하는데 이를 해결하기 위해서 DAO.java 파일에 있는 printError() 메소드를 통해서 어디서 무엇이 잘 못 되었는지에 대해서 출력을 해 주는 메소드로 생각하면 되겠다.
```
        }
        finally{
                close(connect, state, result);
```
-> 우리가 밥을 먹었다면 설거지를 해야 하듯이, 문장의 마지막 부분에는 Connection , Statement / PreparedStatement / CallableStatement , ResultSet의 인스턴스 변수들에 대해서 각각 close() 메소드로 마무으리를 해줘야 한다. 또한 그 인스턴스 변수들이 각각 null인지에 대해서도 확인차가 필요로 하니 참조하면 되겠다.
```
        }
        return dep;
```
-> 최종적으로 나온 Department의 결과를 반환하는 것으로 종결지으면 되겠다.
```
}
```

<클래스 별 정리>
우리가 방금 살펴봤던 Connection, Statement / PreparedStatement / CallableStatement, ResultSet는 자바 프로그램 안에 자동적으로 존재하지 않는다. 그래서 본문 앞에 java.sql.(Collection, Statement, ResultSet 등등)을 import해줘야 쓸 수 있으니 참조하면 좋겠다. 또한 이러한 인스턴스 변수들은 초기에 null로 초기화를 해주고 난 뒤에 설정을 하는 원리로 생각하면 되겠다.
```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

<순서를 쉽게 외우고 싶어요!!!>
우리가 문장을 살펴보게 되면 어딘가가 연관이 되는 부분이 있을 것이다. 필자가 여러분들을 위해서 소개시켜줄 연상법이 있는데 아래와 같은 표를 통해서 그 인스턴스 변수에 따른 실행방안에 대해서 서로 연관이 되는 것을 발견할 수 있다. 이를 통해서 공부를 하는 방안이 있으니 급할 때 참조하면 되겠다.

	connect를 통해서 JDBC 드라이브 적재 후 DB connection 연결	connect →	DB. getConnection()
	statement →	connect.preparedStatement(SQL문장)	SQL 문장을 연결해주는 역할
	statement →	setInt/setString	매개변수 지정. ?의 순서는 첫 번째부터 시작
resultSet →	statement. executeQuery()	select문은 executeQuery() DML문장은 executeUpdate()	
하단 부에는 ResultSet에 대해서 다루고 난 뒤, Connection, Statement, ResultSet 인스턴스 변수를 close()하는 것으로 종결			

-> 순서는 아래 쪽으로 내려가면서 읽어보면 되겠다.

2) ResultSet를 어떻게 정리해줘야 할까?

```
private static Department makeDepartment(ResultSet result) throws Exception{
        Department dep=new Department();
        dep.setDepartment_id(result.getInt("department_id"));
        dep.setDepartment_name(result.getString("department_name"));
        dep.setManager_id(result.getInt("manager_id"));
        dep.setLocation_id(result.getInt("location_id"));
        return dep;
}
```

우리가 ResultSet를 저장하고 난 후에는 분명 Department 객체에 각각 인스턴스에 정리를 해둘 필요가 있다. 그래서 위의 코드와 같은 방법으로 작성하면 되겠다. ResultSet에는 각 컬럼 별로 이름이 나오게 되는데 컬럼의 이름에 대해서는 자바이니만큼 소문자로 작성해보는 것이 좋겠다. 여기서 주의할 점은 가령 우리가 날짜에 대하여 문자열로 변환하고 출력을 하는 경우가 있는데 이럴 경우에는 꼭 Alias(별칭)을 이용해서 조치를 하도록 하자!!! 가령 예를 들어 hire_date를 2016/2/3 형식으로 문자열로 저장을 하고 싶을 때는 String sql에 다음과 같이 작성을 해준다.

String sql="Select TO_CHAR(hire_date, 'YYYY/MM/DD') hiredate from employees where employee_id=?";

그리고 Employee 객체가 따로 형성이 되었다고 가정을 하면 hire_date에 관련된 값들을 ResultSet에 저장이 되고 난 후에 인스턴스에 넣을 때는 이처럼 작성을 해주면 되겠다.

Employee emp=new Employee();

emp.setHiredate(result.getString("hiredate"));

또한 ResultSet에는 각 컬럼의 데이터 형식 별로 Integer(정수형 컬럼), Double(실수형 컬럼), String(문자열 컬럼) 등으로 나누어서 저장이 되어 있는데 예를 들어 Employees, Departments 테이블로 사례를 든다면, 다음과 같은 메소드들을 통해서 컬럼의 값들을 불러오고 싶을 때 참조하면 되겠다.

result.getInt(_____)

Departments -> department_id, location_id, manager_id

Employees -> employee_id, salary, department_id

result.getString(_____)

Departments -> department_name

Employees -> first_name, last_name, TO_CHAR(hire_date, 'YYYY/MM/DD'), email, job_id 등등

3) 모든 행들에 대해서 출력하기

문제 사례) Departments 테이블에 있는 모든 세부 정보에 대해 출력하는 문장을 JDBC로 연동시켜보자.

```
public static ArrayList<Department> selectAll() throws Exception{
        String sql="SELECT * From Departments";
```

-> 우리가 흔히 모든 컬럼들에 출력할 때 쓰는 문자가 바로 *(별)이 되겠는데 이를 쓸 때 주의할 점이 날짜형, 시간형 데이터 컬럼이 있다면 TO_CHAR로 변경하고 작성해줘야 하니 유의해서 쓰면 된다.

```
        Connection connect=null;
        ResultSet result=null;
        PreparedStatement state=null;
```

-> 늘 그래왔듯이 Connection, ResultSet, PreparedStatement에 대해서 각각 null로 초기화해줘야 한다. 물론 전체 행에 대해 출력을 한다면 Statement를 이용해도 무관하다만, 실제로 데이터베이스를 다루는 회사에서 이거 쓰다가 뒷통수를 맞을 수 있으니 웬만하면 PreparedStatement를 이용하도록 하자.

```
        try{
                connect=DB.getConnection();
                state=connect.prepareStatement(sql);
                result=state.executeQuery();
```

-> 위 세 문장들은 1)에서 자세히 설명되었으니 참조하면 되겠다.

```
                ArrayList<Department> dept=new ArrayList<Department>();
```

-> 우리가 부서에 관련된 정보들을 저장하려고 한다. 그러나 하나만 저장하는 것이 절대로 아니다. 그래서 Collection 패키지 내에 존재하는 ArrayList라는 컬렉션(추상 자료구조 모음터)친구를 이용해서 저장을 해 나가는 방식을 생각해 볼 필요가 있다.

```
                while(result.next()){
```

-> ResultSet에서 계속 반복을 돌게 된다. 그렇지만 알아둘 필요가 있는 것은 ResultSet가 어떤 원리를 이용해서 돌아가는지에 대해서 숙지를 할 필요성이 있다. 아래 표를 이용해서 설명하겠다.

Department_id	Department_name	Manager_ID	Location_ID
100	행정부	808	1300
200	정보부	708	1400
300	인사부	608	1500

ResultSet는 정보들을 모두 저장해 나가는 원리로 next() 메소드를 통해서 행을 가리키게 되면서 그 행들에 대해서 정보를 출력해 나가는 원리로 작동이 되지만, 우리가 알아둬야 할 것은 ResultSet에 존재하는 포인터와 같은 역할을 하는 역할이 있다. 그렇지만 여기서 중요한 것은... 그 포인터에 대해서 알아야 할 것이 아니라 그 포인터가 어떤 원리로 작동을 하는가에 대해서 알아둘 필요가 있다. 예를 들어 Department_id가 100인 부서의 RowNumber를 1이라고 가정을 하자. 그 포인터는 처음에 1을 가리키는 것도 아니고, 아무 것도 가리키지 않게 된다. 여기서 알아둬야 할 것은 그 포인터가 만일 next() 메소드를 계속 실행해 나가면 1번 행부터 2번 행, 3번 행... 이런 식으로

계속 값들을 참조하게 되어서 행정부부터 인사부를 계속 저장해 나아가는 원리로 살펴볼 수 있다. 마치 우리가 자료구조론 시간에 배우는 연결리스트를 연상할 수 있어서 이를 이용해서 튜플들을 불러오는 원리로 생각하고 while문을 작성하면 되겠다!!!

```
                dept.add(makeDepartment(result));
```

-> 여기서 dept라는 리스트에 makeDepartment라는 방금 2번에서 살펴본 메소드를 통해서 한 행씩 정보를 저장해 나가는 원리로 해서 결과를 저장하게끔 한다. 이는 여러 행이 반복돌 때 까지 문장이 작동을 한다. 언제까지? 마지막 인덱스를 참조할 때 까지!!!

```
                }
                return dept; // 언제나 그래왔듯이 앞에서 앞서 만든 데이터에 대해서 반환을 하는 것으로 종결을
```

지으면 되겠다. 여기서 예외를 두지 않은 것을 살펴볼 수 있지만, 이 코드에서는 departments 테이블에 적어도 데이터가 1개 이상이 있다고 가정 하에 작성이 되었다.

```
        }
        finally{
                close(connect, state, result); // 늘 그래왔듯이 마지막에 나온 사람이 불끄고 가야 되는 건 잊지
```

말도록 하자.

```
        }
}
```

4) Insert문에서 JDBC를 연동시켜보자

문제 사례) Jobs 테이블을 이용해서 데이터를 Insert하는 문장에 대해 작성해보자.

```
public static void insert(String job_id, String job_title, int max_salary, int min_salary) throws Exception {
        String sql = "INSERT INTO Jobs(job_id, job_title, max_salary, min_salary) VALUES (?, ?, ?, ?)";
```

-> Insert문을 실행하기 위해서는 매개변수(영어로 파라미터라 하죠?)를 이용해서 값들을 추가하는 원리를 이용하는데 방금 1번에서 SetInt, SetString을 이용해서 값들을 추가해 나간다.

```
        Connection connect=null;
        PreparedStatement state=null;
        ResultSet rs=null;
        try {
                connect = DB.getConnection();
                state= connect.prepareStatement(sql);
```

-> 우선 state까지는 sql 문장을 통해서 접근 하는 방법은 같다. 또한 매개변수를 지정하는 방법도 setInt, setString 메소드를 이용하는 방법까지는 같다. 그러나 필자가 강조하고 싶은 것은 DML문장을 통해서 실행을 하기 위해서는 executeUpdate() 메소드를 이용해야 된다는 점을 강조하고 싶다. 이는 아래 부분을 살펴보도록 하자.

```
                state.setString(1, job_id);
                state.setString(2, job_title);
                state.setInt(3, max_salary);
                state.setInt(4, min_salary);
```

-> 늘 그래왔듯이 ?의 순서는 1번부터 시작해서 4번까지 데이터의 형식에 맞게 유두리있게 작성을 해야 한다.

```
                state.executeUpdate();
```

-> executeUpdate() 메소드... DML 관련 문장을 sql에 저장하고 난 뒤에 적용이 가능한 문장으로 살펴볼 수 있는데 DML에서 대표적으로 쓰이는 update문 때문에 이를 쓴다고 생각하면 쉽게 생각할 수도 있겠지만, DML문장에서는 무조건 죽었다 깨어나도 executeUpdate() 메소드를 이용해야 한다.

```
        }
        catch (SQLException e) {
                if (e instanceof java.sql.SQLIntegrityConstraintViolationException)
```

-> 여기서 중복된 값들이 추가되는 경우에 쓰는 예외가 바로 SQLIntegrity블라블라 한것인데 중간에 Constraint가 있는 것으로 봐서 Primary Key에 관련된 제약조건을 위배하는 경우에 발생하는 예외로 살펴보면 되겠다. 이러한 예외이름에 대해선 외우다가 시간만 버리게 된다. 그러니 제약조건을 위배한 경우에 있는 예외가 이러한게 있구나라고 짚고 넘어가면 좋겠다. instanceof 라는 연산자는 이러한 예외에 속한 종류들에 대해서 처리를 하는 연산자이니 나중에 고급자바를 공부하실 분들은 집에서 검색해서 찾아보면 좋겠다.

```
                System.out.println("동일한 Job Id가 존재합니다.");
```

-> 여기서 위배된 조건은 Primary Key에 대해서 걸린다. Primary Key 제약조건은 필자가 지난 요약정리에서 Unique, Not Null이랑 결합되었다고 노래를 부르고 다녔다. 그래서 중복된 값들이 추가되는 경우에는 경고를 알리고 예외처리를 시작한다.

```
                printError(e);
        }
        finally {
            close(con, stmt, rs); // 늘 그래왔듯이 마지막에 나온 사람이 불끄고 가야 되는 건 잊지 말자.
        }
}
```

-> 만일 Update 문에서는 어떻게 이용이 될까? 방금 전에 작성한 문장을 살짝만 바꿔서 쓰면 된다. 그렇지만 Update 문에서 추가되는 것은 int n을 이용해서 수정되는 행 번호를 골라오는 것을 목표로 삼게 된다. 어떠한 원리로 작동이 되는지에 대해 살펴보도록 하자.

5) Update문에서 JDBC를 연동시켜보자

문제 사례) Jobs 테이블을 이용해서 데이터를 Update하는 문장에 대해 작성해보자. 매개변수로는 직책을 변경할 직무ID, 변경할 직책을 받으면 된다.

```
public static void update(String job_id, String job_title) throws Exception {
        String sql = "UPDATE JOBS SET JOB_TITLE=? WHERE JOB_ID=?";
        Connection con=null;
        PreparedStatement stmt=null;
        ResultSet rs=null;
        try {
                con = DB.getConnection();
                stmt= con.prepareStatement(sql);
                stmt.setString(1, job_title);
                stmt.setString(2, job_id);
```

-> 이번에는 1번과 2번의 물음표를 통해서 job_id를 조건으로 받아들여서 job_title로 저장하는 문장을 필자가 작성해 봤다. 우리가 Update 문을 통해서 문장을 작성하면 하나만 설정이 가능한 것이 아니라 콤마로 구분을 해서 다음과 같은 방안으로도 수정이 가능한 사실을 알고 넘어가야 한다.

```
Update Jobs
Set Job_Title='IT Programmer', Min_Salary=6500
where Job_ID='IT_PROG';
```

```
                int n= stmt.executeUpdate();
```

-> Update 문에서 n을 써야 하는데 이 n은 무엇일까? 바로 수정이 들어갈 행의 개수이다. 행의 개수를 executeUpdate()를 통해서 받아오고 난 후에 이용하면 되는데 우선 job_id가 Primary Key 제약조건이 걸려져 있어서 하나만 추출하게끔 작성했다. 만일 범위를 정해놓고 하는 방식이여도 이대로 이용하면 되겠다. 참조를 살펴보자.

```
                if (n==0) {
                        NotRecordFoundException e=new NotRecordFoundException("JOB_ID에 해당하는
데이터가 없음");
                        throw e;
                }
```

-> Update 문에서 수정할 데이터가 없다면 n은 자동적으로 0이 나오게 된다. 그래서 n이 0인 경우에는 예외를 통해서 처리를 하게끔 문장을 작성해주면 되겠다.

```
        }
        catch (SQLException e) {
                printError(e);
        }
        finally {
                close(con, stmt, rs);
        }
```

-> 마지막으로는 항상 늘 그래왔듯이 close로 마무으리를 해주면 되겠다.

```
}
```

(참조) Update문이 하나만이 아닌 여러 행을 수정하는 경우

문제 사례) Jobs 테이블을 이용해서 최소 봉급이 10000 달러 이하인 부서의 최소 봉급을 4 달러로 바꿔보는 문장을 연습해보자.

```
public static void update4(int min_salary) throws Exception {
        String sql = "UPDATE JOBS SET MIN_SALARY=4 WHERE Min_Salary<=?";
        … 이하 생략 …(위와 같다.)
            int n= stmt.executeUpdate();
            if (n==0) {
                        NotRecordFoundException e=new NotRecordFoundException("해당된 부서가 없습니다.");
                        throw e;
            }
            System.out.println(n+"개의 행을 수정하였습니다.");
        }
        … 이하 생략 …
```

여기서 n이 나오는 값은 열의 개수가 나오게 되어서 최소 봉급이 10000 달러 이하인 부서에 대해 최소 봉급을 4달러로 바꾸는 문장은 다음과 같이 작성하면 여러 행들에 대해서 바꿀 수 있으니 공부하는데 참조하자.

또한 DML 관련 문장들에서는 ResultSet가 보이지 않게 된다. 이는 Select 문에 대해서 쿼리들을 저장할 때만 이용하는 객체로 살펴볼 수가 있다. 그래서 DML 문장을 적용할 때에는 굳이 ResultSet에 대한 이용 부담은 없어도 되겠다.

6) Delete문에서 JDBC를 연동시켜보자

문제 사례) Jobs 테이블을 이용해서 데이터를 Delete하는 문장에 대해 작성해보자. 매개변수로는 직무ID만 받아서 해보자.

```
public static void deleteJOB_ID(String job_id) throws Exception {
        String sql = "Delete JOBS WHERE Job_ID=?";
        Connection con=null;
        PreparedStatement stmt=null;
        ResultSet rs=null;
        try {
            con = DB.getConnection();
            stmt= con.prepareStatement(sql);
            stmt.setString(1, job_id);
            con.setAutoCommit(false);
            int n= stmt.executeUpdate();
            if (n==0) {
                        NotRecordFoundException e=new NotRecordFoundException("JOB_ID에 해당하는
데이터가 없음");
                        throw e;
            }
        }
        catch (SQLException e) {
                printError(e);
        }
         finally {
            close(con, stmt, rs);
         }
}
```

그 to the 렇 to the 다. 방금 Update 문에서 설명한 것에 대해서 약간만 수정해서 작성하면 큰 문제가 없다. 물론 범위를 짜서 해당이 안 되는 값들에 대해 삭제하는 문장에 대해서도 5번의 참조처럼 하면 된다. 허나 삭제를 할 경우에는 자동적으로 commit가 되기 때문에 필자가 요약정리에서 5번에 작성한 메소드에 대해서 알아두면 약이 되겠다.

7) 저장된 프로시저를 통해서 데이터를 출력하는 문장 만들기

문제 사례) Departments 테이블을 이용해서 지역번호가 1700인 부서에 대한 정보들을 출력하는 문장으로 마무리 지어보자. 이는 SYS_REFCURSOR를 이용해 보겠다.

다음과 같은 프로시저를 통해서 봉급의 합이 10000이상 30000이하인 부서의 번호를 우선적으로 따오게 된다.

```
Create Or Replace Procedure Department_1700(dep_cursor OUT SYS_REFCURSOR)
IS
Begin
  open dep_cursor for Select * from departments where location_id=1700;
End;
/
```

그러면 이러한 문장들을 Java에서 연동을 하도록 다음과 같이 작성해 보겠다.

```
public static ArrayList<Department> locate1700() throws Exception{
                String sql = "begin hr.Department_1700(?); end;";

                Connection connect=null;
                CallableStatement state=null;

                ResultSet result=null;
                ArrayList<Department> list=new ArrayList<Department>();
                try{
                        connect=DB.getConnection();
                        state=connect.prepareCall(sql);
                        state.registerOutParameter(1, OracleTypes.CURSOR);
                        state.executeQuery();
                        result=(ResultSet)state.getObject(1);
                        while(result.next()){
                                list.add(makeDepartment(result));
                                result=(ResultSet)state.getObject(1);
                        }
                }
                catch (SQLException e){
                        printError(e);
                }
                finally{
                        close(connect, state, result);
                }
```

```
        return list;
}
```