



데이터베이스(jdbc)



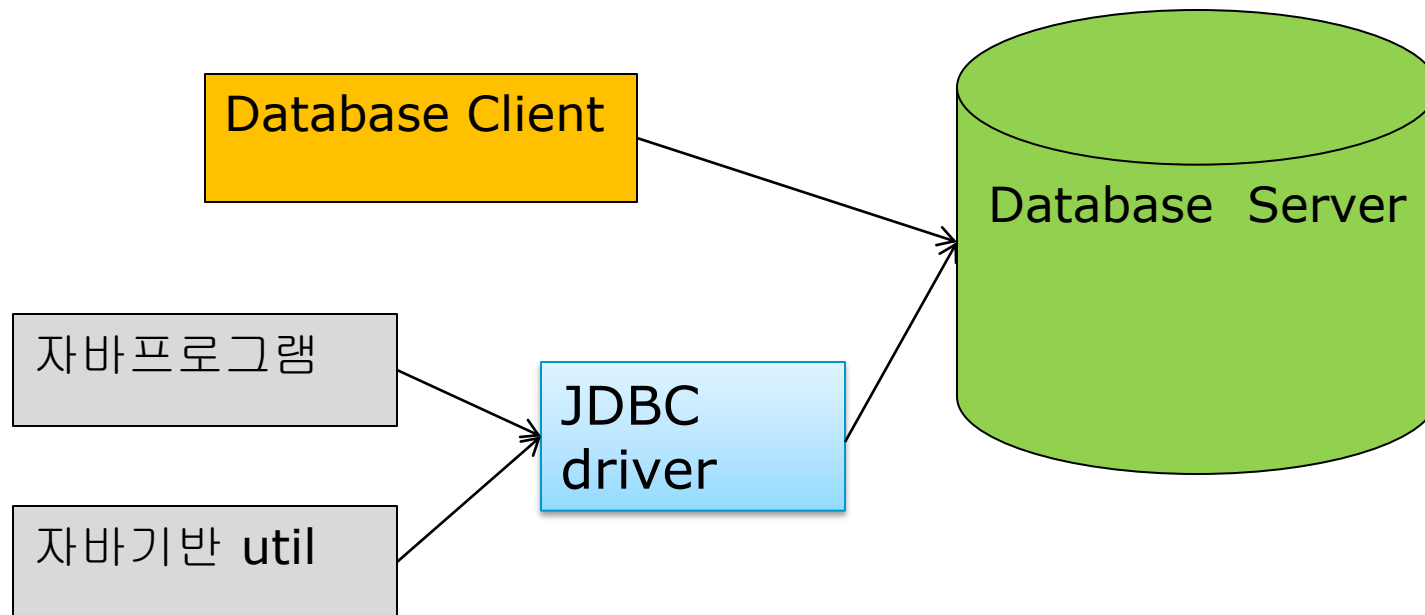
- **jdbc**의 개념과 자바에서 활용되는 **DB** 처리 프로세스에서 명확하게 파악한다.
 - **VO(value object), DAO(data access object), DTO(data tranfer object)**를 구분해서 **DB**처리시 활용 할 수 있도록 한다.
 - 데이터베이스의 내용을 조회, 등록, 수정, 삭제 시, 처리할 모듈을 구성할 수 있다.
-



생각해봅시다 :

- **java**에서 데이터베이스 서버는 어떻게 연결이 될까?
 - 데이터를 조회할 때, 필요로 하는 요소들이 어떤 것이 있을까?
 - **java**에서 데이터베이스와 연결 후, 자원은 어떻게 관리를 할까?
 - **DB** 연결 시, 발생한 예외 내용은 어떤 것이 있을까?
 - **SQL**을 통해 동적 **query**은 어떻게 처리하여야 할까?
-

- JDBC(**J**ava **D**ata**B**ase **C**onnectivity)란 자바로 데이터베이스에 접근할 수 있게 하는 프로그램 **API**를 말한다.





jdbc를 이용해서 DB 접근 :

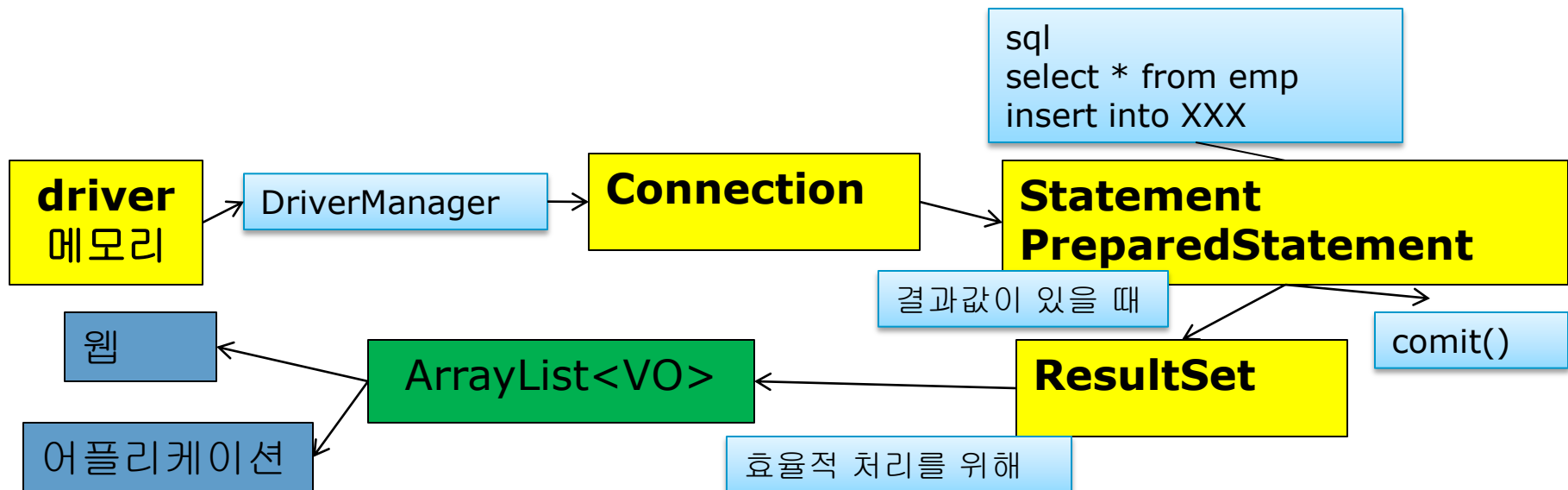
- driver 다운로드 및 lib에 위치 지정
 - 오라클 서버
 - C:\oracle\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar
 - Web(jsp기반) 프로그램
 - \WebContent\WEB-INF\lib 에 해당 lib파일을 위치
 - 드라이버 로드
 - 클래스를 jvm 메모리에 로딩
 - lib에 있는 package명.클래스
 - **oracle.jdbc.driver.OracleDriver.class**
 - Class.forName(driver명)
-



jdbc를 이용해서 DB 접근 :

■ DB와 연결

- url : jdbc driver(thin):IP정보:port정보:SID
 - jdbc:oracle:thin:@localhost:1521:xe
- Connection **con** =
DriverManager.getConnection(url, id, pass);





jdbc를 이용해서 DB 접근 :

- 대화하기(Statement, PreparedStatement)
 - sql문 작성
 - select * from emp
 - 연결된 객체(con)과 Statement 연결
 - Statement **stmt**=**con**.createSatement();
 - 실제 sql문 처리..결과이 있을 때, ResultSet
 - ResultSet **rs** = **stmt**.executeQuery(sql문)
- 결과값 처리(ResultSet) – select 문일 때.
 - next() : 데이터 있을 때까지 호출(다음 row에 데이터가 있는지 여부boolean)
 - while(rs.next())
 - getXXX("sql의 title명");
 - rs.getString("name")



jdbc를 이용해서 DB 접근 :

- 결과값이 없는 sql문 (insert, update, delete)
 - con.commit() 호출 : 등록, 수정, 삭제 확정
 - con.rollback() 호출 : 에러
- ResultSet → ArrayList<VO> 변환 처리.
 - sql의 list와 맞는 VO 객체 생성(java)
 - ArrayList<Emp> list = new ArrayList<Emp>();
 - Emp e=null;
 - while(rs.next()){ // rs.next() : row단위로 반복
 - e = new Emp();
 - e.setEmpno(rs.getInt("empno"));
 - ...
 - ...
 - list.add(e);
 - }



jdbc에서 예외 처리 :

- Database 접속은 외부에 연결하여 데이터 IO(input/out)가 일어나기에 자바에서는 반드시 예외처리하게끔 강제하고 있다.
 - 드라이버 메모리 설정
 - 데이터베이스 연결, sql 처리, 결과값 받는 내용 → `SQLException`
 - `try{`
 - DB처리 관련된 코드
 - `}catch(Exception e){}`
- Exception에서의 자원해제
 - `.close()` : 자원해제를 해야지 효과적으로 메모리 관리가 된다.
 - `finally{}` 구문에서 수행
 - 수정, 삭제, 입력 시, 예외발생하면 `rollback()` 호출해서 원복처리



공통**DB**모듈 만들기 :

- XXXDB.java 생성
- vo 패키지 생성
 - VO 클래스 생성
- XXXDB.java
 - 사용할 DB관련 필드 선언



DB 접속 처리 과제 :

- A02_ComDB.java
 - DB 접속 처리 모듈 선언 및 처리 완료
 - public void setConn()
- 데이터베이스 ArrayList<Student>

| 이름 | 학년 | 반 | 국어 | 영어 | 수학 |
|-----|----|---|----|----|----|
| 홍길동 | 2 | 3 | 70 | 80 | 90 |
| 신길동 | 1 | 5 | 80 | 90 | 90 |

- A02_ComDB.java 하위에
 - public ArrayList<Student> list(){
 - return list;
 - }
- Main()
 - A02_ComDB db = new A02_ComDB();
 - ArrayList<Student> st=db.list(); st를 for문으로 출력



ArrayList<VO> 데이터 가져오기 :

- Data
 - 테이블 구조
 - sql : select * from emp
 - 나타나는 데이터의 column명과 type(문자/숫자/날짜형/boolean 등) 확인
 - VO : 단위데이터를 저장할 class code
 - sql의 column명과 type에 합당한 VO class 생성
 - Database 모듈
 - public ArrayList<Emp> empList(){}
 - 메서드 선언
-



data list 가져오는 메서드 :

- `public ArrayList<Emp> empList(){`
 - `return`할 결과값 선언
 - `ArrayList<Emp> list = new ArrayList<Emp>();`
 - `setConn() : DB연결 처리 메서드 호출`
 - 대화처리 :
 - `con.createStatement()`
 - `rs = stmt.executeQuery(sql)`
 - 결과값 받기
 - `Emp emp=null;`
 - `while(rs.next())` → 데이터를 row단위로 가져오면서 다음 row에 데이터 있는지 여부 확인
 - `rs.getInt("컬럼명")`
 - `list.add(emp) : ArrayList에 객체 넣기`
 - 예외 처리 : 자원해제



예외와 자원해제 ●

- 자원 할당 순서
 - Connection → Statement → ResultSet
- 자원 해제 순서
 - ResultSet → Statement → Connection
 - close()
 - if(자원!=null){
 - 자원.close()
 - }
 - finally에서 처리 한다.



웹화면에서 **DB**모듈 불러오기 :

- import 하기 : `<%@ page`
 - DB모듈
 - `import="jspexp.z01_database.A01_EmpDB"`
 - VO와 ArrayList
 - `import="jspexp.z02_vo.Emp,java.util.*"`
 - `<%%>` 안에 선언
 - for문을 `table<%%>, <%= %>` 출력 처리
-



확인예제 :

- dept테이블 이용해서 화면 list 처리(1단계)
- student테이블 생성 데이터 입력 화면 list처리(2단계)

| 이름 | 학년 | 반 | 국어 | 영어 | 수학 |
|-----|----|---|----|----|----|
| 홍길동 | 2 | 3 | 70 | 80 | 90 |
| 신길동 | 1 | 5 | 80 | 90 | 90 |

- 테이블 및 VO 객체 만들기
 - 1단계
 - `select * from dept`
 - `class Dept{`
 - `private int deptno;`
 - `private String dname;`
 - `private String loc;`
 - `}`
 - 2단계
 - student 테이블 생성 (구조 확인)
 - `create table student();`
 - 이름name, 학년grade, 반sect, 국어 kor, 영어eng, 수학math
 - 테이블 데이터 입력
 - `insert into student values('홍길동', 2, 1, 90,80,70);`
 - VO 클래스 생성



2단계 풀이 :

```
create table student(  
    name varchar2(50),  
    grade number,  
    sect number,  
    kor number,  
    eng number,  
    math number  
);  
insert into student  
    values('홍길동',2,1,90,80,70);  
select * from student;
```



2단계 풀이 :

```
public class Student {  
    private String name;  
    private int grade;  
    private int sect;  
    private int kor;  
    private int eng;  
    private int math;
```



조회 처리 :

- 조회 데이터를 입력하면, DAO를 통하여 조회 조건 값이 입력되고, 입력된 조건에 의해 list 처리하는 프로세스를 말한다.

이름 : []

부서명 : []

조회

| 사번 | 이름 | 급여 | 입사일 |
|----|----|----|-----|
| | | | |
| | | | |
| | | | |



조회 처리 :

- 조회 메서드를 통해 조회 조건 관련된 내용 전달
 - ArrayList<Emp> search(Emp sch){
 - sql 구문
 - select * from emp
 - where ename like '%'|| sch.getName()
||'%'
 - app(main)
 - DataDAO dao = new DataDAO();
 - Emp sch = new Emp();
sch.setEname("홍길동")
 - dao.search(sch);
-



조회 처리 :

- 웹 화면에서 조회 검색 처리..
- `<input name="name"/>`에 value으로 입력을 하고, 조회 버튼으로 다음 page에 submit() 했을 때.
- `String name = request.getParameter("name");`
 - `DataDAO dao = new DataDAO();`
 - `Emp sch = new Emp(); sch.setEname(name);`
 - `dao.search(sch);`

| 이름 : [] | | | 조회 |
|-------------------------------|----|----|-----|
| 사번 | 이름 | 급여 | 입사일 |
| | | | |
| | | | |
| | | | |



PreparedStatement

- sql을 먼저 서버단에 보내고, 다음에 전달 값을 보내는 방식으로 처리되는 **Statement** 객체 중 하나로 속도, 보안에 대한 이슈를 해결해 준다.
 - PreparedStatement 객체 필드로 선언
 - sql = "select * from where job=?"
 - pstmt=con.prepareStatement(sql);
- sql 에 ?에 해당하는 data를 아래 구조 매핑
 - pstmt.setString("Manager");
- 실행후, 아래 구조로 데이터를 받아옴
 - ResultSet rs = pstmt.executeQuery();



PreparedStatement 확인예제 :

- 학생 list 화면 데이터에서
 - 이름 [] [검색] (1단계)
 - 이름 [] [select 반/번호] [](2단계)
 - [select 국어/영어/수학] []
[select/이상/이하/동일]
 - (속제)
-



PreparedStatement 데이터검색 :

- DB
 - 테이블 생성, 데이터 입력
 - sql ?
- VO객체(read)
- DAO
 - db 서버 연결
 - sql : select * from where no=? and name=? :데이터가 들어갈 부분
 - pstmt=con.prepareStatement(sql);
 - ?에 매핑되는 데이터..
 - pstmt.setInt(1, 25);
 - pstmt.setString(2, "홍길동");
 - rs = pstmt.executeQuery(X);



문제 은행 list 만들기 :

- 테이블 구성

| 문제내용[] | | 유형[select 주관/객관] | 검색 |
|---------------|----|------------------|----|
| 번호 | 문제 | 정답 | 타입 |
| | | | |
| | | | |

- 데이터 입력
- VO 만들기
- DAO 만들기



데이터 베이스 입력 처리 :

- sql
 - insert into emp(empno, ename) values(?,?)
- transaction : con.setAutoCommit(false)
 - commit
 - 정상처리 시, : commit();
 - rollback
 - 예외발생 시, : rollback();
- pstmt.con.prepareStatement(sql);
- pstmt.setInt(1, dto.getEmpno());
- pstmt.setString(2, dto.getEname());
- pstmt.executeUpdate(); (등록, 수정, 삭제)
- con.comit(); 정상처리시 commit
- 예외처리
 - con.rollback();



입력 모듈(메서드) 만들기-확인예제 :

- dept 테이블 입력모듈(1단계)
 - STUDENT/테이블 입력처리(2단계)
 - 입력 sql 작성 및 test
 - 메서드 만들기
 - 연결
 - autocommit
 - pstmt 처리.
 - commit
 - 예외처리
-



등록 화면(jsp) 및 처리 :

- 메인 조회 (order by 사원번호 desc)
 - 등록 버튼 클릭 시, 등록화면으로 이동
 - 사원 등록 화면
 - 사원명:[]
 - 직책 :[select]
 - 관리자:[select]
 - 연봉 :[]
 - 보너스 : []
 - 부서[select] [등록]
 - 등록처리jsp ➔ 메인 조회으로 이동처리
-



등록 처리 **proc.jsp**

- import DAO
 - <%객체 생성
 - 요청값 처리
 - request.getParameter("ename")
 - 요청값을 입력 dto에 할당.
 - dao의 insert메서드의 매개값으로 전달
 - response.sendRedirect(초기 리스트 페이지 이동)
-



등록 화면처리(확인예제) :

- 부서 list
 - 학생 list
 - 등록 버튼
 - 등록 화면
 - 등록 처리 `proc.jsp` 처리
-



정리 및 확인하기 :



감사합니다 !
