7. 데이터베이스와 JSP의 연동

1. 데이터베이스 개요 및 설치
2. SQL 쿼리 개요(생략)
3. JDBC를 사용한 JSP와 데이터베이스 연동
4. 자카르타 DBCP API를 이용한 커넥트풀 사용
5. 트랜잭션 처리
6. 데이터베이스 암호화

데이터베이스 개요 및 설치

데이터베이스와 DBMS

- DBMS(Database Management System)
  - 데이터베이스를 관리하는 프로그램
  - 데이터를 안정적으로 보관할 수 있는 다양한 기능 제공
    - 데이터의 삽입/수정/삭제, 데이터의 무결성 유지, 트랜잭션 관리, 데이터의 백업 및 복원, 데이터 보안 등의 기능을 제공
  - 대표적인 예
    - SYBASE, MS-SQL, ACCESS, ORACLE, MySQL, DB2, INFORMIX

DBMS 설치 및 데이터베이스 생성하기

DBMS 설치

- MySQL 다운로드
  - http://dev.mysql.com/downloads/
- 설치
  - 다운로드 받은 mysql-버전-win32.msi 파일을 마우스 오른쪽 버튼을 눌러 [설치] 메뉴를 선택해서 설치
  - 기본값을 그대로 사용하고 [Next] 버튼을 눌러 설치
  - [character set] 설정 부분이 나오면 [Manual Select Default Character Set/ Collation] 항목을 선택하고 [Character set] 항목의 값을 [euckr]로 선택한 후 [Next] 버튼을 클릭
  - [root 계정]의 패스워드 입력부분이 나오면 패스워드를 두 번 입력한 후 [Next] 버튼을 클릭
  - 마지막에 [Execute] 버튼을 클릭하면 서비스가 시작

DBMS 설치 및 데이터베이스 생성하기

- MySQL 드라이버 설치
  - MySQL과 프로그래밍과 연동 시 필요
  - http://dev.mysql.com/downloads/의 [MySQL Connectors] 항목에서 다운받음
  - 다운받은 mysql-connector-java-버전.zip 파일을 압축을 해제
  - mysql-connector-java-버전-bin.jar 파일을 이클립스의 [프로젝트]-[Webcontent]-[WEB-INF]-[lib]폴더에 복사

DBMS 설치 및 데이터베이스 생성하기

- MySQL에 데이터베이스 추가
  - mysqladmin을 사용
    - *mysqladmin -u root -p create jsptest*
    - *root 계정 비밀번호 입력 후 [enter]*
- 생성된 데이터베이스에 사용자 계정 추가 및 권한 설정
  - MySQL에 루트 계정으로 접속
    - *mysql -u root -p*
  - jsptest 데이터베이스에 로컬호스트(localhost)에 접근할 수 있는 사용자 계정 추가 및 권한을 설정

```
grant select, insert, update, delete, create, drop, alter
on jsptest.* to 'jspid'@'localhost' identified by jsppass';
```

  - jsptest 데이터베이스에 모든 서버(%)에 접근할 수 있는 사용자 계정 추가 및 권한을 설정

```
grant select, insert, update, delete, create, drop, alter
on jsptest.* to 'jspid'@'%' identified by 'jsppass';
```

이클립스에서 [Data Source Explorer] 뷰를 사용한 DBMS 제어

- 이클립스에서 데이터베이스 직접 제어
- [Eclipse IDE for Java EE Developers]에서 제공하는 [Data Source Explorer] 뷰 사용
- [Data Source Explorer] 뷰에서 쿼리문을 실행

Data Source Explorer] 뷰에 DBMS 커넥션 생성

- [Data Source Explorer] 뷰의 [Database Connections] 항목을 선택 후 마우스 오른쪽 버튼을 눌러 [New...] 메뉴를 선택
- [New Connection Profile]창의 [Connection Profile Type] 항목을 [MySQL] 선택한 후 [Name]항목에 [mysqlconn]을 입력한 후 [next]버튼 클릭(오라클 설치는 [Connection Profile Type] 항목에서 [Oracle]선택)
- [Specify a Driver and Connection Details] 화면에서 [Drivers] 항목을 [New Driver Definition] 버튼 클릭
- [New Driver Definition] 창에서 [Name/Type]탭에서 사용할 드라이버 (mysql-connector-java-5.6.24-bin.jar) 선택
- [JAR List] 탭에서 JDBC 드라이버 경로 설정, 기존의 드라이버는 제거
- [properties]탭에서 커넥션 설정에 필요한 URL, 데이터베이스 이름, JDBC 드라이버 클래스, 접근에 필요한 계정이름과 패스워드 설정
- URL : jdbc:mysql://localhost:3306/basicJsp, Database Name: basicjap, password : jappass, UserId : jspid 설정 후 [Test Connection] 버튼을 클릭하여 [Success] 대화상자가 표시되면 [OK] 버튼 클릭

스크랩북 작성

- [Data Source Explorer] 뷰에서 커넥션을 선택 후 [Open SQL Scrapbook] 아이콘을 클릭

2. SQL 쿼리의 개요

SQL 쿼리의 개요

SQL 쿼리의 개요

- SQL(Structured Query Language): 데이터베이스 생성부터 레코드 검색 등의 작업을 수행할 때 사용

- SQL문의 종료
  - 데이터 정의문(DDL)- CREATE, ALTER, DROP
  - 데이터 제어문(DCL) - GRANT, REVOKE
  - 데이터 조작문(DML) - SELECT, INSERT, DELETE,
  - 쿼리(Query) - SELECT
  - 트랜잭션(Transaction) 처리 - COMMIT, ROLLBACK

데이터 타입(Data Type) - MySQL

- 숫자 타입
  - TINYINT : 1 byte, -128~127, UNSIGNED 0~255
  - SMALLINT : 2 bytes, -32768~32767, UNSIGNED 0~65535
  - MEDIUMINT : 3 bytes, -8388608~8388607, UNSIGNED 0~16777215
  - INT, INTEGER : 4 bytes, -2147483648~2147483647,
  - UNSIGNED 0~4294967295
  - BIGINT : 8 bytes, -9223372036854775808~9223372036854775807
  - UNSIGNED 18446744073709551615
- s

데이터 타입(Data Type) - MySQL

- 날짜 및 시간 타입
  - DATE : 3 bytes
  - DATETIME : 8 bytes
  - TIMESTAMP : 4 byte

- 문자열 타입
  - CHAR : 1~255
  - VARCHAR : 1~255
  - BLOB(Binary Large Object), TEXT : 1~65535
  - MEDIUMBLOB, MEDIUMTEXT : 1~1677215
  - LONGBLOB, LONGTEXT : 1~4294967295

테이블 작업 관련 쿼리문

테이블 생성- CREATE문

- 작성 예

```
create table member(
  id varchar(50) not null primary key,
  passwd varchar(16) not null,
  name varchar(10) not null,
  reg_date datetime not null
);
```

테이블 작업 관련 쿼리문

테이블 수정- ALTER문

- 테이블의 필드를 추가하는 ALTER문의 일반형

```
ALTER TABLE table_name
  ADD (add_col_name1 type [DEFAULT] [NOT NULL/NULL],
              ....
              add_col_name3 type );
```

- 작성 예

```
alter table member
  add (address varchar(100) not null,
              tel varchar(20) not null);
```

테이블 작업 관련 쿼리문

테이블 수정- ALTER문

- 테이블의 필드를 수정하는 ALTER문의 일반형은 다음과 같다.

```
ALTER TABLE table_name
   MODIFY (modi_col_name1 type [DEFAULT] [NOT  NULL/NULL],
   ....
   modi_col_name3 type );
```

- 테이블의 필드를 삭제하는 ALTER문의 일반형

```
ALTER TABLE table_name
    DROP del_col_name1;
```

테이블 작업 관련 쿼리문

테이블 제거- DROP문

- 문법

  DROP TABLE 삭제할 테이블명;

- 작성 예

  drop table test;

레코드 작업 관련 쿼리문

레코드 추가 - INSERT문

- 레코드를 추가하는 쿼리의 일반형

```
INSERT INTO table_name (col_name1,col_name2…)
VALUES (col_value1, col_value2…)
```

- 사용 예

```
insert into member(id, passwd, name, reg_date)
 values('kingdora@dragon.com','1234','김개동', now());
```

레코드 삭제 - DELETE문

- 레코드 삭제하는 쿼리 일반형

```
delete from table_name  where condition
```

레코드 작업 관련 쿼리문

레코드 검색 - SELECT문

- 레코드를 검색하는 쿼리의 일반형
  - *SELECT col_name1,col_name2*
  - *FROM table_name;)*
- 사용 예 : select * from member;

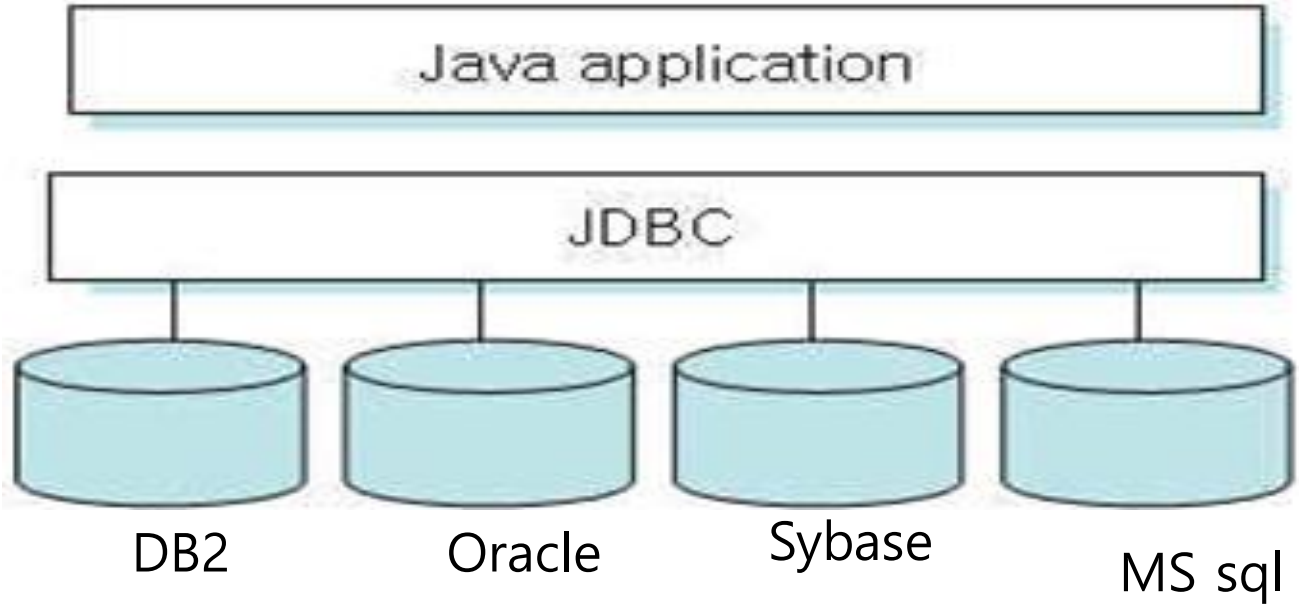
레코드 수정 - UPDATE문

- 레코드를 수정하는 쿼리의 일반형

  UPDATE table_name SET col_name = value WHERE condition;
- 사용 예 : update member set passwd='3579' where id='abc';

JDBC를 사용한 JSP와 데이터베이스의 연동

- **JDBC의 개요**
  - 자바 프로그램(JSP포함)과 관계형 테이터 원본(데이터베이스, 테이블...)을 연결하는 인터페이스
  - JDBC 라이브러리는 'java.sql' 패키지에서 제공
  - JDBC 라이브러리는 SQL문을 실행시키기 위한 인터페이스로 설계

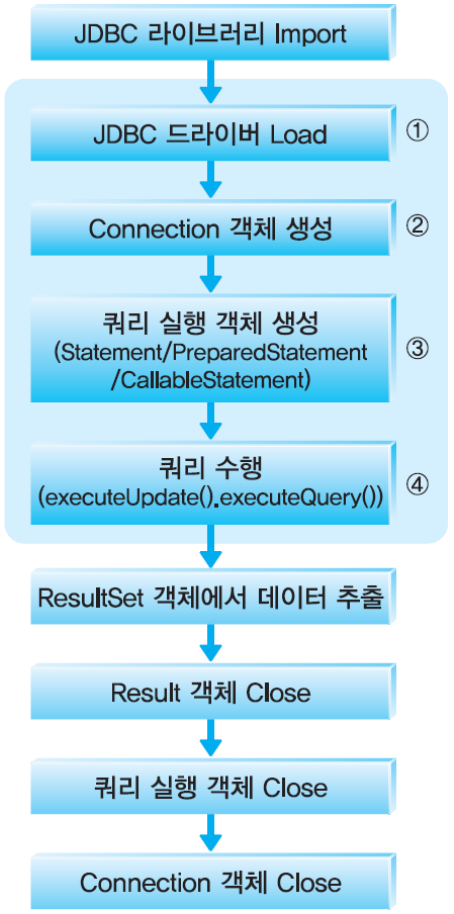


JDBC 드라이버의 종류

- JDBC-ODBC 브리지+ODBC드라이버 (JDBC-ODBC Bridge Plus ODBC Drive)
- 네이티브-API 부분적인 자바 드라이버 (Native-API Partly-Java Driver)
- JDBC-Net 순수 자바 드라이버 (JDBC-Net Pure Java Driver)

JDBC를 사용한 JSP와 데이터베이스의 연동

- JDBC 프로그램은 JDBC 드라이버 로드, Connection 객체 생성, 쿼리 실행 객체 생성, 쿼리 수행의 필수 4단계에 의해 프로그램 됨



JDBC 프로그램의 작성 단계

- 1단계(JDBC 드라이버 Load)
  - Class 클래스의 forName() 메소드를 사용해서 드라이버를 로드
  - 예

```
Class.forName("com.mysql.jdbc.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- 2단계(Connection 객체 생성)
  - DriverManager에 등록된 각 드라이버들을 getConnection(String url) 메소드를 사용해서 식별
  - 예

```
Connection conn= DriverManage.getConnection
    ("jdbc:mysql://localhost:3306/jsptest","jspid","jsppass");
Connection conn=DriverManager.getConnection
    ("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
```

JDBC 프로그램의 작성 단계

- 3단계(Statement/PrepardStatement/CallableStatement 객체 생성)
  - sql쿼리를 생성하며, 반환된 결과를 가져오게 할 작업 영역을 제공
  - 예

```
Statement stmt = conn.createStatement();
PrepardStatement pstmt = prepareStatement(sql);
CallableStatement cstmt = prepareCall();
```

- 4단계(Query 수행)
  - Statement/PrepardStatement/CallableStatement 객체의 executeQuery() 메소드나 executeUpdate() 메소드를 사용해서 쿼리를 실행.
  - stmt.executeQuery() : recordSet 반환 => Select 문에서 사용

```
ResultSet rs = stmt.executeQuery ("select * from 소속기관");
```

JDBC 프로그램의 작성 단계

- 4단계(Query 수행)
  - stmt.executeUpdate(): 성공한row 수 반환 => Insert 문, Update 문, Delete 문에서 사용
    - *String sql="update member set passwd='3579' where id='abc' ";*
    - *stmt.executeUpdate(sql);*

- 5단계(ResultSet 처리)
  - 한 레코드씩 처리되고, 다음 행으로 이동시 next() 메소드를 사용
  - 사용 예

```
while (rs.next ()){
   out.println(rs.getString ("id"));
   out.println(rs.getString ("pass"));
}
```

JDBC 프로그래밍에 사용되는 객체

- DriverManager 클래스
  - JDBC 드라이버를 사용해서 JSP에서 사용할 수 있는 커넥션을 생성
  - 특정 드라이버 클래스를 지정하면, 자동으로 로딩되어 객체가 생성
    - 예 : *Class.forName("com.mysql.jdbc.Driver");*
  - getConnection() 메소드를 사용해서 Connection 객체 생성
    - 예 : *Connection conn = DriverManger.getConnection(url,user,pass);*
- Connection 인터페이스
  - 특정 데이터 원본과 연결된 커넥션을 의미
  - SQL 쿼리문을 실행 시 필요
    - 특정한 SQL문을 정의하고 실행시킬 수 있는 Statement/ PrepardStatement/CallableStatement 객체를 생성 시 필요

JDBC 프로그래밍에 사용되는 객체

- Statement 인터페이스
  - Connection 객체의 메소드를 사용해 객체 생성
    - *Statement stmt = connection.createStatement();*
  - executeQuery() 메소드 또는 executeUpadte() 메소드를 호출해 쿼리를 실행
  - 쿼리의 수행속도가 가장 느려 요즘에는 거의 사용 안 함

- CallableStatement 인터페이스
  - Connection 객체의 prepareCall() 메소드를 사용해서 객체 생성
  - 스토어드 프로시저 (Stored Procedure) 사용을 제공
    - 데이터베이스에 저장된 프로시저(함수)를 단지 호출하는 것만으로 처리가 가능
    - 예 : *conn.prepareCall("{call query1 }");*

JDBC 프로그래밍에 사용되는 객체

PreparedStatement 인터페이스

- Connection 객체의 prepareStatement() 메소드를 사용해서 객체 생성
- 쿼리문이 미리 컴파일되어 속도 빠름

```
PreparedStatement pstmt=conn.prepareStatement(sql);
```

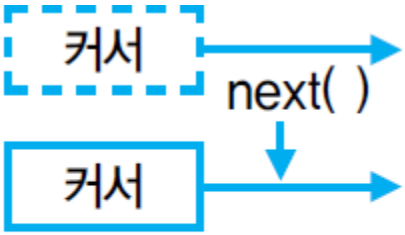
- 각각의 인수에 대해 위치홀더(placeholder)를 사용하여 SQL 문장을 정의, 위치홀더는 물음표(?)로 표현
- setXxx() 메소드를 사용해서 실제 값으로 대치

```
String sql= "insert into member values (?,?,?,?,?)";
PreparedStatement pstmt=conn.prepareStatement(sql);
pstmt.setString(1,id);
pstmt.setString(2,passwd);
```

JDBC 프로그래밍에 사용되는 객체

ResultSet 인터페이스

- Select문을 실행하는 executeQuery() 메소드 수행 성공 시, 결과물로 ResultSet 객체 반환
- ResultSet 객체는 '커서(cursor)'라 불리는 것을 가지고 있는데, 이것을 사용해 ResultSet 객체에서 특정 레코드를 참조
- ResultSet 객체의 next() 메소드를 사용해서 다음 위치로 커서 이동



id	passwd	age
XXX	XXX	XXX
XXX	XXX	XXX

4. 자카르타 DBCP API를 이용한 커넥션 풀 사용

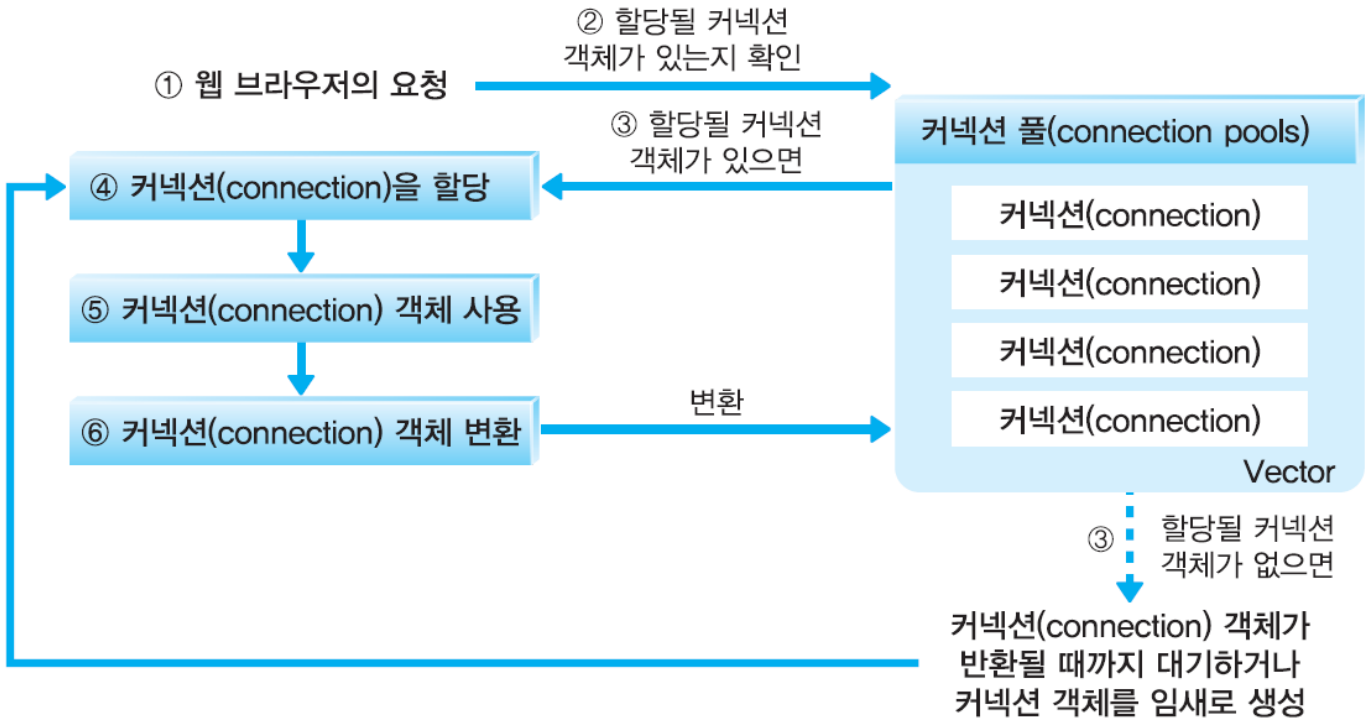
커넥션 풀의 개요

- 데이터베이스 커넥션은 데이터베이스에 한번 연결하기 위한 작업
  - 이러한 작업들을 매번 새로운 데이터베이스 연결에 대한 요청이 들어올 때 마다 수행하는 것은 시스템에 과부하를 줌
    - 커넥션 풀을 사용해서 개선
- 커넥션 객체들을 만들어 놓은 후, 커넥션 객체가 필요한 경우 작성한 객체를 할당해 주고, 사용이 끝난 후에는 다시 커넥션 풀로 회수하는 방법을 사용

▲ service( ) 메소드와 커넥션 객체

커넥션 풀의 전략

- service()메소드(사용자 요청)당 1개씩 할당
- 커넥션의 개수를 제한
- 커넥션 객체 관리자가 다 쓰면 자원을 회수



▲ 커넥션 풀의 구현 방법

자카르타 DBCP API를 이용한 커넥션 풀

- 톰캣의 5.0.x 버전부터 DBCP API를 사용한 커넥션 풀 제공
- 커넥션 풀 사용 방법
  - DBCP API관련 jar파일 설치
  - DBCP에 관한 정보 설정 - server.xml
  - JNDI 리소스 사용 설정 - web.xml
  - JSP페이지에서 커넥션 풀 사용

자카르타 DBCP API를 이용한 커넥션 풀

DBCP API 관련 jar 파일 설치

- DBCP API 관련 jar 파일은 http://commons.apache.org/ 사이트에서 다운로드받아 압축 해제
  - [Collections] : 자카르타(Jakarta) DBCP API가 사용하는 자카르타 Pool API의 jar 파일
  - [DBCP] : 자카르타(Jakarta) DBCP API 관련 jar 파일
  - [Pool] : Pool API가 사용하는 자카르타 Collection API의 jar 파일
- [프로젝트]-[Web-INF]-[lib]에 JAR DBCP API 관련 jar 파일 배치

자카르타 DBCP API를 이용한 커넥션 풀

- DBCP에 관한 정보 설정 - server.xml
  - 설정할 내용
    - <Resource name="jdbc/jsptest " auth="Container"
    - type="javax.sql.DataSource " driverClassName="com.mysql.jdbc.Driver"
    - username="jspid " password="jsppass"
    - url="jdbc:mysql://localhost:3306/jsptest " maxWait="5000 " />
  - 설정할 파일의 위치
    - 실제 서비스 환경(톰캣홈\conf)
    - 이클립스 가상 환경([Project Explorer] 뷰의 [Servers]-[Tomcat v8.0 Server ~])

자카르타 DBCP API를 이용한 커넥션 풀

JNDI 리소스 사용 설정 - web.xml

- server.xml에 저장된 JNDI 리소스를 자바빈 또는 JSP페이지에서 사용하기 위해 설정

- 설정할 내용

```
<resource-ref>
    <description>jsptest db</description>
    <res-ref-name>jdbc/jsptest</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

자카르타 DBCP API를 이용한 커넥션 풀

- JSP페이지 또는 자바빈에서 커넥션 풀 사용

  - 추가할 코드
    - *Context initCtx = new InitialContext();*
    - *Context envCtx = (Context) initCtx.lookup("java:comp/env");*
    - *DataSource ds = (DataSource)envCtx.lookup("jdbc/jsptest");*
    - *Connection conn = ds.getConnection();*

5. 트랜잭션 처리

- **트랜잭션은 여러 단계의 작업을 하나로 처리하는 것**
- **하나로 인식된 작업이 모두 성공적으로 끝나면 commit이 되고, 하나라도 문제가 발생하면 rollback되어서 작업을 수행하기 전 단계로 모든 과정이 회수**
- **JSP에서 제공하는 트랜잭션 처리에 메소드**
  - commit(): 트랜잭션의 commit을 수행
  - rollback() : 트랜잭션의 rollback을 수행
- **JSP는 오토커밋(Autocommit)**
  - 트랜잭션을 처리할 때는 오토커밋을 해제
    - setAutoCommit(false);

사용 예

```
conn.setAutoCommit(false);
  생략
    sql = "insert into buy
            (buy_id, buyer, book_id, book_title, buy_price, buy_count,";
    sql += "book_image, buy_date, account, deliveryName, deliveryTel
                        , deliveryAddress)";
    pstmt.executeUpdate();
생략
  pstmt = conn.prepareStatement( "delete from cart where buyer=?");
  pstmt.setString(1, id);
  pstmt.executeUpdate();
  conn.commit();
  conn.setAutoCommit(true);
```