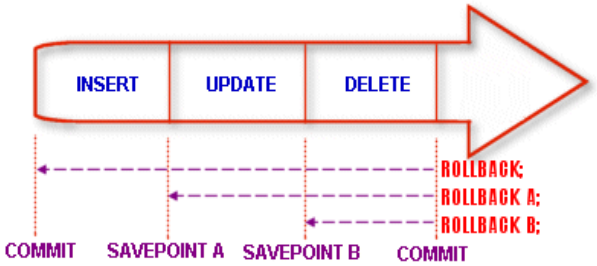
11. Transaction 처리, Database Connection Pool

트랜잭션 처리

- 트랜잭션(Transaction)
  - 데이터 처리를 위한 논리적인 작업 단위
    - 일반적으로 여러 개의 DML문과 질의문들로 구성될 수 있음
  - All or Nothing 방식으로 처리
    - 오류가 없을 경우 실행이 완료(commit)되고 그 결과가 데이터베이스에 영속적으로 유지됨
    - 오류 발생 시 모든 DML문들의 실행 결과가 취소됨
  - ACID 속성을 만족해야 함
    - Atomicity(원자성)
    - Consistency(일관성)
    - Isolation(고립성)
    - Durability(영속성)
  - 예: 계좌 이체, 상품 주문 등

트랜잭션 처리

- 트랜잭션 제어 명령
  - COMMIT
  - ROLLBACK
  - SAVEPOINT



트랜잭션 처리

- Commit
  - 트랜잭션을 정상적으로 완료하고 모든 실행 결과를 영구히 저장함
  - commit이 실행된 후에는 트랜잭션에 의해 변경된 데이터를 복구할 수 없음
  1. 인위적인 commit – 사용자의 COMMIT 명령에 의해 실행됨
  2. 자동 commit – DDL문 실행 시 발생
     - 예: 테이블에 대한 CREATE, ALTER, DROP, RENAME, TRUNCATE 등
- Rollback
  - 데이터의 변경을 모두 취소하고 트랜잭션 시작 전의 상태로 돌아감
  - 트랜잭션에서 데이터 변경 시 이전 값(before image)을 rollback segment에 임시로 저장했다가 rollback 처리 시 사용
  1. 인위적인 rollback – 사용자의 ROLLBACK 명령에 의해 실행됨
  2. 자동 rollback – 트랜잭션 실행 도중 오류 발생 시 자동으로 rollback 처리 됨
- Savepoint
  - 여러 개의 DML 문을 포함하는 트랜잭션에서 사용자가 트랜잭션의 중간 지점에 save-point들을 생성 가능
  - 트랜잭션 rollback 시 특정 save-point까지만 rollback 실행 가능

트랜잭션 처리

- Commit, Rollback 실행 예

```
SQL> CREATE TABLE dept2 AS SELECT * FROM dept;
                                  // 자동 commit 실행 & 새로운 트랜잭션 시작
SQL> SELECT * FROM dept2;         // 복사된 데이터 존재
SQL> DELETE FROM dept2;           // 모든 행들을 삭제
SQL> SELECT * FROM dept2;         // 행이 존재하지 않음
SQL> ROLLBACK;                    // rollback 실행 (모든 행이 복구됨)
SQL> SELECT * FROM dept2;         // 기존의 행들을 출력
SQL> DELETE FROM dept2;           // 모든 행들을 삭제
SQL> COMMIT;                      // commit 실행 (삭제된 상태를 영구히 저장)
                                  // 새로운 트랜잭션 시작
SQL> ROLLBACK;                    // rollback 실행 (삭제된 행들을 복구할 수 없음)
SQL> SELECT * FROM dept2;         // 행이 존재하지 않음
```


트랜잭션 처리

- DELETE와 TRUNCATE TABLE의 차이

```
SQL> CREATE TABLE dept2 AS SELECT * FROM dept;
SQL> DELETE FROM dept2;      // 모든 행 삭제
SQL> ROLLBACK;               // 행 삭제 전으로 rollback (모든 행이 복구됨)
SQL> SELECT * FROM dept2;    // 기존의 행들이 출력됨
SQL> TRUNCATE TABLE dept2;   // table이 truncate 됨(모든 행 삭제)
                             // 자동 commit & 새로운 트랜잭션 시작
SQL> ROLLBACK;               // 삭제된 행들을 복구할 수 없음
SQL> SELECT * FROM dept2;    // 행이 존재하지 않음
```

  - DELETE 실행 후 그 결과를 즉시 데이터베이스에 반영하기 위해서는 명시적으로 COMMIT을 실행해야 함

트랜잭션 처리

- Savepoint 사용 예

```
SQL> CREATE TABLE dept2 AS SELECT * FROM dept WHERE 1=2;
                                  // 자동 commit 실행 & 새로운 트랜잭션 시작
SQL> INSERT INTO dept2 SELECT * FROM dept;
SQL> SAVEPOINT A;                 // save-point A 설정
SQL> DELETE FROM dept2 WHERE deptno=10;
SQL> SELECT * FROM dept2;
SQL> SAVEPOINT B;                 // save-point B 설정
SQL> DELETE FROM dept2 WHERE deptno=20;
SQL> SELECT * FROM dept2;
SQL> SAVEPOINT C;                 // save-point C 설정
SQL> DELETE FROM dept2 WHERE deptno=30;
SQL> ROLLBACK TO C;               // save-point C 지점으로 rollback됨
  또는 ROLLBACK TO A;             // save-point A 까지 rollback됨
  또는 ROLLBACK;                  // 트랜잭션 시작 전으로 rollback됨
```


JDBC Transaction

- JDBC의 트랜잭션 처리 방식
  - 기본적으로 각 DML문이 하나의 트랜잭션으로 실행됨("auto-commit" mode)
    - DML문 실행 후 자동으로 commit이 실행되어 DML문의 실행 결과가 데이터베이스에 즉시 영속적으로 반영됨
  - 여러 개의 DML문들로 구성된 트랜잭션을 정의하려면 auto-commit mode를 해제하고 필요한 시점에 명시적으로 commit 또는 rollback을 실행해야 함
- Transaction 정의
  - java.sql.Connection interface 이용

기능	method
auto-commit 설정/해제	void **setAutoCommit**(boolean autoCommit)
savepoint 설정	Savepoint **setSavepoint**()
commit 실행	void **commit**()
rollback 실행	void **rollback**() void **rollback**(Savepoint savepoint)

JDBC Transaction

- 일괄 갱신(batch update)
  - 여러 개의 DML문을 한꺼번에 전송하여 실행함으로써 성능 향상
    - Network traffic 감소 (only 1 round trip)
    - DBMS에서 효율적으로 처리 가능 (예: 병렬처리 기능 이용)
  - Statement 및 PreparedStatement 에서 사용 가능
  - 실행 후 각 DML문의 실행 결과를 나타내는 정수들의 배열을 반환함
  - 예

```
Connection conn = … ;                          // Connection 객체 생성 및 초기화

Statement stmt = conn.createStatement();
stmt.addBatch("INSERT INTO employees VALUES (1000, 'Joe Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260, 'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000, 260)");
int[] insertCounts = stmt.executeBatch();              // batch update 실행
```

1, 1, 1이 반환되면 성공
0이 하나라도 있으면 어디선가 실패

트랜잭션 실행 예

- 예제 테이블

```
CREATE TABLE Account (          // 계좌
    accNo   NUMBER   PRIMARY KEY,     // 계좌번호
    balance  NUMBER                    // 잔액
);

INSERT INTO Account VALUES (101, 100000);
INSERT INTO Account VALUES (102, 100000);
INSERT INTO Account VALUES (103, 100000);
INSERT INTO Account VALUES (104, 100000);
```


트랜잭션 실행 예

- TransactionTest.java

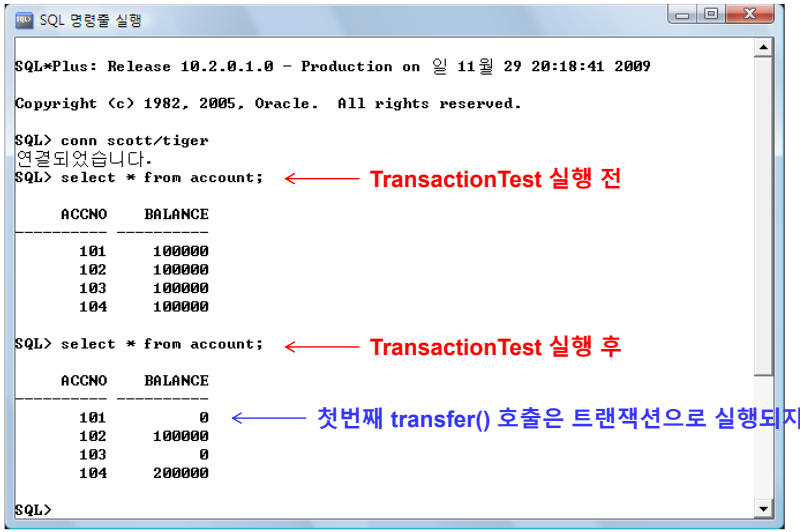
```
public class TransactionTest {
    static Connection conn = null;
    public static void main(String args[])  {
        String url = "jdbc:oracle:thin:@localhost:1521/xepdb1";
        String user = "...", passwd = "...";
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException ex) { ex.printStackTrace(); }
        try {
            conn = DriverManager.getConnection(url, user, passwd);
                                        // auto-commit mode (default)
            transfer(101, 200, 100000);     // 주의: accNo 200은 존재하지 않음
            conn.setAutoCommit(false);      // auto-commit 기능 해제
            transfer(102, 200, 100000);     // 주의: accNo 200은 존재하지 않음
            conn.setAutoCommit(false);      // auto-commit 기능 해제
            transfer(103, 104, 100000);

            if (conn != null)
                try { conn.close(); } catch (SQLException ex) { … }
        } catch (SQLException ex) { ex.printStackTrace(); }
    }
```



```
    public static void transfer(int senderAccNo, int receiverAccNo, int amount) {
        PreparedStatement pStmt = null;
        try {
            String sql1 = "UPDATE account SET balance = balance - ? WHERE accNo = ?";
            pStmt = conn.prepareStatement(sql1);
            pStmt.setInt(1, amount); pStmt.setInt(2, senderAccNo);
            if (pStmt.executeUpdate() != 1) { throw new AppException(); }
            pStmt.close();

            String sql2 = "UPDATE account SET balance = balance + ? WHERE accNo = ?";
            pStmt = conn.prepareStatement(sql2);
            pStmt.setInt(1, amount); pStmt.setInt(2, receiverAccNo);
            if (pStmt.executeUpdate() != 1) { throw new AppException(); }
            conn.commit();                                  // transaction commit
        } catch (Exception ex) {
            try {
                conn.rollback();                            // transaction rollback
            } catch (SQLException e) { e.printStackTrace(); }
            ex.printStackTrace();
        } finally {
            if (pStmt != null) {
                try { pStmt.close(); } catch (SQLException ex) { ex.printStackTrace(); }
            }
        }
    }
}
```

트랜잭션 실행 예

◆ 실행 결과



TransactionTest 실행 전

TransactionTest 실행 후

첫번째 transfer() 호출은 트랜잭션으로 실행되지 않았음


```
public static void transfer(int senderAccNo, int receiverAccNo, int amount) {    // 수정
    PreparedStatement pStmt = null;
    try {
        conn.setAutoCommit(false);          // auto-commit 해제 → transaction 시작
        String sql1 = "UPDATE account SET balance = balance - ? WHERE accNo = ?";
        pStmt = conn.prepareStatement(sql1);
        pStmt.setInt(1, amount); pStmt.setInt(2, senderAccNo);
        if (pStmt.executeUpdate() != 1) { throw new AppException(); }
        pStmt.close();

        String sql2 = "UPDATE account SET balance = balance + ? WHERE accNo = ?";
        pStmt = conn.prepareStatement(sql2);
        pStmt.setInt(1, amount); pStmt.setInt(2, receiverAccNo);
        if (pStmt.executeUpdate() != 1) { throw new AppException(); }
        conn.commit();                                  // transaction commit
    } catch (Exception ex) {
        try { conn.rollback();                          // transaction rollback
        } catch (SQLException e) { e.printStackTrace(); }
        ex.printStackTrace();
    } finally {
        try { conn.setAutoCommit(true); }               // auto-commit 재설정
        catch (SQLException ex) { ex.printStackTrace(); }
        if (pStmt != null) {
            try { pStmt.close(); } catch (SQLException ex) { ex.printStackTrace(); }
        }
    }
}
```



```
public static void transfer_batch1(int senderAccNo, int receiverAccNo, int amount) {
    Statement pStmt = null;                     // Statement 이용
    try {
        conn.setAutoCommit(false);          // auto-commit 해제 → transaction 시작
        stmt = conn.createStatement();
        stmt.addBatch("UPDATE account SET balance = balance - " + amount +
                      " WHERE accNo = " + senderAccNo);
        stmt.addBatch("UPDATE account SET balance = balance + " + amount +
                      " WHERE accNo = " + receiverAccNo);
        int[] cnt = stmt.executeBatch();

        if (cnt[0] != 1 || cnt[1] != 1) { throw new AppException(); }
        conn.commit();                                  // transaction commit
    } catch (Exception ex) {
        try { conn.rollback();                          // transaction rollback
        } catch (SQLException e) { e.printStackTrace(); }
        ex.printStackTrace();
    } finally {
        try { conn.setAutoCommit(true); }               // auto-commit 재설정
        } catch (SQLException ex) { ex.printStackTrace(); }
        if (stmt != null)
            try { stmt.close(); } catch (SQLException ex) { ex.printStackTrace(); }
    }
}
```



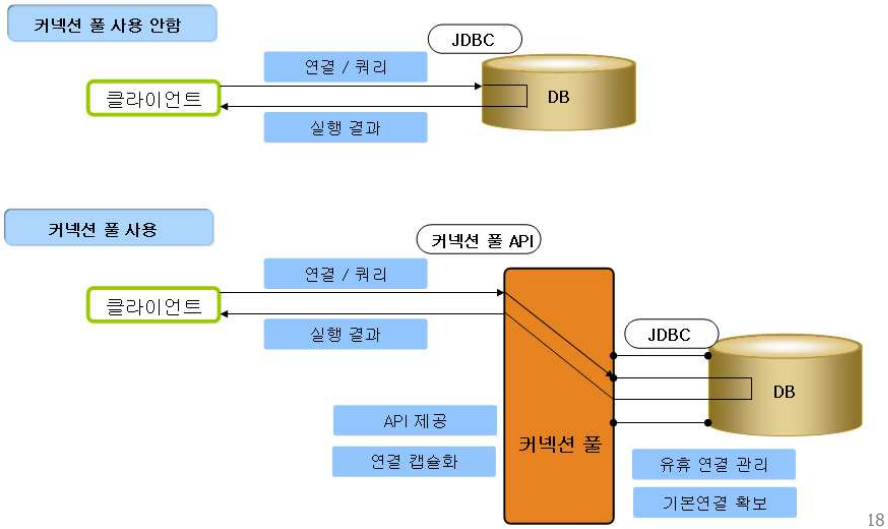
```
public static void transfer_batch2(int senderAccNo, int receiverAccNo, int amount) {
    PreparedStatement pStmt = null;             // PreparedStatement 이용
    try {
        conn.setAutoCommit(false);          // auto-commit 해제 → transaction 시작
        String sql1 = "UPDATE account SET balance = balance - ? WHERE accNo = ?";
        pStmt = conn.prepareStatement(sql1);
        pStmt.setInt(1, amount);
        pStmt.setInt(2, senderAccNo);
        pStmt.addBatch();

        pStmt.clearParameters();                    // 질의 파라미터들을 삭제(optional)
        pStmt.setInt(1, -amount);
        pStmt.setInt(2, receiverAccNo);
        pStmt.addBatch();

        int[] cnt = pStmt.executeBatch();
        if (cnt[0] != 1 || cnt[1] != 1) { throw new AppException(); }
        conn.commit();                                  // transaction commit
    } catch (Exception ex) {
        try { conn.rollback(); } catch { … } …  // transaction rollback
    } finally {
        …
        if (stmt != null)
            try { pStmt.close(); } catch (SQLException ex) { ex.printStackTrace(); }
    }
}
```

Database Connection Pool

- 개요
  - DBMS와 연결(connection)을 생성하는 것은 많은 자원(memory, CPU 등)과 시간이 요구됨
  - 애플리케이션에서 DBMS 접속이 필요할 때마다 connection 생성 및 삭제를 반복하는 것은 비효율적이며 성능 저하를 유발
  - Connection Pool 이용
    - 일정 개수의 DBMS connection들을 (미리) 생성하고 pool로 관리함으로써 시스템 성능을 향상시킴
    - 애플리케이션은 필요할 때 connection pool에 connection을 요청하고 사용이 끝난 후에는 다시 connection pool에 반환함
    - Connection pool에서는 유휴 connection들을 삭제하지 않고 유지하여 애플리케이션에서 재사용 되도록 함

Database Connection Pool

- 개요



Commons DBCP

- Apache Commons에서 구현한 Database Connection Pool library
  - Commons DBCP2 (commons-dbcp2-2.x.0.jar)
    - http://commons.apache.org/proper/commons-dbcp/
    - Commons Pool2 library에 의존 (commons-pool2-2.x.0.jar)
      - ✓ http://commons.apache.org/proper/commons-pool/
- 사용 방법
  - Commons DBCP library를 Maven을 통해 다운로드 받아 애플리케이션 내에 포함시켜 사용
    - https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2/2.9.0
  - 애플리케이션 서버에서 제공하는 DBCP library 이용 가능
    - 예: Tomcat에 포함된 DBCP library
      - ✓ <TOMCAT_HOME>/lib/tomcat-dbcp.jar

Commons DBCP 사용 예

```
import org.apache.commons.dbcp2.BasicDataSource;   // Commons DBCP2
public class ConnectionManager {
    private BasicDataSource ds = null;
    public Connection getConnection() throws SQLException {
        if (ds == null)  setupDataSource();
        return ds.getConnection();                  // DataSource로부터 connection 획득
    }
    private void setupDataSource() {                // BasicDataSource 객체 생성 및 setup
        String url = "jdbc:oracle:thin:@localhost:1521/xepdb1";          // DBMS 접속 정보
        String className = "oracle.jdbc.driver.OracleDriver";
        String userName = "…", password = "…";

        ds = new BasicDataSource();
        ds.setDriverClassName(className);        // JDBC driver class 설정
        ds.setUrl(url);                          // DBMS 접속 URL 설정
        ds.setUsername(userName);                // DBMS 사용자 설정
        ds.setPassword(password);                // DBMS 사용자 암호 설정
        ds.setMaxTotal(10);  ds.setInitialSize(10);       // connection pool parameter 설정
        ds.setMinIdle(5);  ds.setMaxWaitMillis(5000);
        ds.setPoolPreparedStatements(true);
    }
}
```

Commons DBCP 사용 예

```
Import dbcp.ConnectionManager;
public class DbcpTest {
    public static void main(String args) {
        Connection conn = null;
        PreparedStatement pstmt=null;
        ResultSet rs =null;
        String query = "SELECT * FROM EMP";
        try {
            ConnectionManager cm = new ConnectionManager();   // conn manager 생성
            conn = cm.getConnection();                        // database connection 획득
            pstmt = conn.prepareStatement(query);
            rs = pstmt.executeQuery();
            System.out.println("No    Name");
            while (rs.next()) {
                int no = rs.getInt("EMPNO");
                String name = rs.getString("ENAME");
                System.out.println(no + " " + name);
            }
            pstmt.close();
        } catch (Exception e) { e.printStackTrace();}
        finally { … }
    }
}
```


Commons DBCP: 외부 설정파일 이용

```
public class ConnectionManager2 {
    private BasicDataSource ds = null;
    private void ConnectionManager2() {
        InputStream input = null;
        Properties prop = new Properties();
        try {
            input = getClass().getResourceAsStream("/context.properties");
            prop.load(input);              // properties file로부터 DB 설정 정보를 loading
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally { … }
        ds = new BasicDataSource();
        ds.setDriverClassName(prop.getProperty("db.driver"));
        ds.setUrl(prop.getProperty("db.url"));
        ds.setUsername(prop.getProperty("db.username"));
        ds.setPassword(prop.getProperty("db.password"));
        …
    }
    public Connection getConnection() throws SQLException {
        return ds.getConnection();
    }
}
```

```
# context.properties file
db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521/xepdb1
db.username=scott
db.password=TIGER
```


Tomcat DBCP 활용

- DBCP 및 JNDI(Java Naming and Directory Interface) 이용
  - Tomcat은 내부적으로 DBCP library를 포함하고 있고 JNDI 서비스를 제공함
  - 웹 애플리케이션 또는 Tomcat에서 설정 파일을 통해 data source 정보를 설정
    - Tomcat에서 DataSource 객체 생성 및 JNDI 서비스 등록
  - 애플리케이션에서 JNDI 서비스 검색을 통해 DataSource 객체 획득 및 사용 가능

- DBCP 관련 설정
  - META-INF/Context.xml: 애플리케이션(또는 Tomcat)에서 resource(data source) 설정
  - WEB-INF/web.xml: 애플리케이션에서 이용하는 resource(data source) 참조 설정
  - 주의사항
    - Tomcat 7+에서는 web.xml의 참조 설정을 생략 가능하나, 애플리케이션의 자원 요구를 명시적으로 나타내기 위해 설정을 유지하는 것이 권장됨
    - Tomcat 실행 시 data source 접근에 필요한 JDBC driver를 자동 탐지 및 등록하기 위해 JDBC driver 파일을 <TOMCAT_HOME>/lib에 미리 저장하는 것이 바람직함

Tomcat DBCP 활용

- Data Source 설정
  - Tomcat Server에 설정 시: <TOMCAT_HOME>/conf/server.xml
  - 웹 애플리케이션에 설정 시: <WebApp>/META-INF/Context.xml
  - 예

```
<Context>
    <Resource name="jdbc/testDB"
              auth="Container"
              type="javax.sql.DataSource"
              driverClassName="oracle.jdbc.driver.OracleDriver"
              url="jdbc:oracle:thin:@localhost:1521/xepdb1"
              username="scott"
              password="TIGER"
              maxTotal="20"
              maxIdle="10"
              maxWaitMillis="-1" … />
</Context>
```

Tomcat DBCP 활용

- Valid attributes for a <Resource> element

Attribute	Description
name	The name of the resource to be created, relative to the java:comp/env context.
auth	Specify whether the web Application code signs on to the corresponding resource manager programatically, or whether the Container will sign on to the resource manager on behalf of the application. The value of this attribute must be Application or Container.
type	The fully qualified Java class name expected by the web application when it performs a lookup for this resource.
…	

Tomcat DBCP 활용

- DBCP2 *BasicDataSource* Parameters

Parameter	Description
username	The connection username to be passed to our JDBC driver to establish a connection.
password	The connection password to be passed to our JDBC driver to establish a connection.
url	The connection URL to be passed to our JDBC driver to establish a connection.
driverClassName	The fully qualified Java class name of the JDBC driver to be used.
connectionProperties	The connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be [propertyName=property;]* NOTE - The "user" and "password" properties will be passed explicitly, so they do not need to be included here.

Tomcat DBCP 활용

- DBCP2 *BasicDataSource* Parameters

Parameter	Default	Description
initialSize	0	The initial number of connections that are created when the pool is started.
maxTotal (maxActive)	8	The maximum number of active connections that can be allocated from this pool at the same time, or negative for no limit.
maxIdle	8	The maximum number of connections that can remain idle in the pool, without extra ones being released, or negative for no limit.
minIdle	0	The minimum number of connections that can remain idle in the pool, without extra ones being created, or zero to create none.
maxWaitMillis (maxWait)	indefinitely	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely.
…		

Tomcat DBCP 활용

- Data Source 참조 설정 (optional)
  - 웹 애플리케이션의 배포기술자(web.xml) 파일에서 설정
    - <WebApp>/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" … >
    <display-name>DBCP Web Test</display-name>
    <welcome-file-list>
        <welcome-file>/index.jsp</welcome-file>
    </welcome-file-list>
    …
    <resource-ref>
        <description>Oracle DataSource Example</description>
        <res-ref-name>jdbc/testDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    …
</web-app>
```

Tomcat DBCP 활용

◆ 예: DAO Class

```
import javax.sql.DataSource;
import javax.naming.*;   // Context, InitialContext
…
public class EmpDAO {
    private DataSource ds;
    public EmpDAO() throws Exception {
        Context init = new InitialContext();
        ds = (DataSource)init.lookup("java:comp/env/jdbc/testDB");
    }

    public List<Emp> findEmpList() {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        String query = "SELECT empno, ename FROM emp";
        List<Employee> empList = new ArrayList<Employee>();

        try {
            conn = ds.getConnection();
            pstmt = conn.prepareStatement(query);
            rs = pstmt.executeQuery();
```


Tomcat DBCP 활용

◆ 예: DAO Class (계속)

```
            while (rs.next()) {
                Employee emp = new Employee();
                emp.setEmpNo(rs.getString("empno"));
                emp.seteName(rs.getString("ename"));
                empList.add(emp);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (pstmt != null) {
                try { pstmt.close(); }
                catch (SQLException ex) { ex.printStackTrace(); }
            }
            if (conn != null) {
                try { conn.close(); }
                catch (SQLException ex) { ex.printStackTrace(); }
            }
        }
        return empList;
    }
}
```


References

◆ Apache Tomcat 8
- JNDI Resources HOW-TO
  - https://tomcat.apache.org/tomcat-8.0-doc/jndi-resources-howto.html
- JNDI Datasource HOW-TO
  - https://tomcat.apache.org/tomcat-8.0-doc/jndi-datasource-examples-howto.html
- Apache Tomcat 8 Configuration Reference
  - https://tomcat.apache.org/tomcat-8.0-doc/config/context.html#Resource_Definitions

◆ Apache Commons DBCP
- BasicDataSource Configuration Parameters
  - http://commons.apache.org/proper/commons-dbcp/configuration.html