Task1. Apache Commons DBCP 이해하기
1. Database와 Application을 효율적으로 연결하는 커넥션 풀(connection pool) Library는 웹 애플리케이션에서 필수 요소이다.
2. Web Application Server로 상용 제품을 사용하다면 보통 제조사에서 제공하는 Connection Pool 구현체를 사용한다.
3. 그 외에 OpenSource Library로 Apache의 Commons DBCP와 Tomcat-JDBC, BoneCP, C3P0, HikariCP 등이 있다.

4. Connection 생성
  1)Commons DBCP에서 이루어진다.
  2)Commons DBCP는 PoolableConnection 타입의 Connection을 생성하고 생성한 Connection에 ConnectionEventListener를 등록한다.
  3)ConnectionEventListener에는 Application이 사용한 Connection을 Pool로 반환하기 위해 JDBC Driver가 호출할 수 있는 Callback Method가 있다.
  4)이렇게 생성된 Connection은 commons-pool의 addObject() Method로 Connection Pool에 추가된다.
  5)이때 commons-pool은 내부적으로 현재 시간을 담고 있는 Timestamp와 추가된 Connection의 Reference를 한 쌍으로 하는 ObjectTimestampPair라는 자료구조를 생성한다.
  6)그리고 이들을 LIFO(last in first out) 형태의 CursorableLinkedList로 관리한다.

5. Connection 개수 관련 속성
  1)Connection의 개수는 BasicDataSource Class의 다음 속성으로 지정할 수 있다.
  2)initialSize
    -BasicDataSource Class 생성 후 최초로 getConnection() Method를 호출할 때 Connection Pool에 채워 넣을 Connection 개수
  3)maxActive
    -동시에 사용할 수 있는 최대 Connection 개수(기본값: 8)
  4)maxIdle
    -Connection Pool에 반납할 때 최대로 유지될 수 있는 Connection 개수(기본값: 8)
  5)minIdle
    -최소한으로 유지할 Connection 개수(기본값: 0)

6. 유효성 검사 Query 설정
  1)JDBC Connection의 유효성은 validationQuery Option에 설정된 Query를 실행해 확인할 수 있다.
  2)Commons DBCP 1.x에서는 다음과 같은 세 가지 Test Option으로 유효성을 검사한다.
  3)유효성을 검사할 때는 validationQuery Option에 하나 이상의 결과를 반환하는 Query를 설정해야 한다.
  4)Commons DBCP 2.x에서는 validationQuery 옵션이 없을 때 Connection.isValid() 메서드를 호출해 유효성을 검사한다.

  5)testOnBorrow
    -Connection Pool에서 Connection을 얻어올 때 Test 실행(기본값: true)

  6)testOnReturn
    -Connection Pool로 Connection을 반환할 때 Test 실행(기본값: false)

  7)testWhileIdle
    -Evictor Thread가 실행될 때 (timeBetweenEvictionRunMillis > 0) Connection Pool 안에 있는 유휴 상태의 Connection을 대상으로 Test 실행(기본값: false)

  8)validationQuery Option는 DBMS에 따라 다음과 같이 Query를 설정하기를 권장한다.
    -Oracle: select 1 from dual
    -Microsoft SQL Server: select 1
    -MySQL: select 1

-CUBRID: select 1 from db_root

9)검증에 지나치게 자원을 소모하지 않게 testOnBorrow 옵션과 testOnReturn 옵션은 false로 설정하고, 오랫동안 대기 상태였던 Connection이 끊어지는 현상을 막게 testWhileIdle 옵션은 true로 설정하는 것을 추천한다.

7. Oracle JDBC 드라이버 9.x에서는 강제로 Session을 종료했을 때 발생하는 ORA-00028 오류가 난 후 부적절한 상태의 Connection이 Connection Pool로 반납돼 Database에 Login되지 않은 때 발생하는 오류인 ORA-01012 오류가 계속 발생한 사례가 있다.

1)근본적인 원인은 Oracle JDBC Driver가 해당 오류 상황에서 JDBC 명세에 정의된 ConnectionEventListener.connectionErrorOccurred() Method를 제대로 호출하지 않았기 때문이었다.

2)Oracle JDBC 드라이버를 10.x로 Upgrade해서 Test했을 때는 같은 오류가 재현되지 않았다.

3)오류가 발생하는 Version을 사용하는 Application에서 Commons DBCP의 testWhileIdle Option을 true로 설정한 Server에서도 오류가 발생하지 않았다.

4)Commons DBCP에서 vadliationQuery Option을 실행하면서 오류가 발생하면 해당 Connection을 Connection Pool에서 제외했기 때문이다.

5)이렇듯 testWhileIdle Option과 유효성 검사 Query 설정으로 예상치 못한 오류 상황도 대비할 수 있다.

8. Evictor Thread와 관련된 속성

1)Evictor Thread는 Commons DBCP 내부에서 Connection 자원을 정리하는 구성 요소이며 별도의 Thread로 실행된다.

2)이와 관련된 속성은 다음과 같다.

-timeBetweenEvictionRunsMillis
--Evictor Thread가 동작하는 간격.
--기본값은 -1이며 Evictor Thread의 실행이 비활성화돼 있다.
-numTestsPerEvictionRun
--Evictor Thread 동작 시 한 번에 검사할 Connection의 개수
-minEvictableIdleTimeMillis
--Evictor Thread 동작 시 Connection의 유휴 시간을 확인해 설정 값 이상일 경우 Connection을 제거한다(기본값: 30분)

9. Evictor Thread의 역할은 3가지인데 각각의 역할을 수행할 때 위의 속성이 어떻게 참조되는지 살펴보자.

1)Connection Pool 내의 유휴 상태의 Connection 중에서 오랫동안 사용되지 않은 Connection을 추출해 제거한다.
-Evictor Thread 실행 시 설정된 numTestsPerEvictionRun 속성값만큼 CursorableLinkedList의 ObjectTimestampPair를 확인한다.
-ObjectTimestampPair의 Timestamp 값과 현재 시간의 Timestamp 값의 차이가 minEvictableIdleTimeMillis 속성값을 초과하면 해당 Connection을 제거한다.
-Connection 숫자를 적극적으로 줄여야 하는 상황이 아니라면 minEvictableIdleTimeMillis 속성값을 -1로 설정해서 해당 기능을 사용하지 않기를 권장한다.

2)Connection에 대해서 추가로 유효성 검사를 수행해 문제가 있을 경우 해당 Connection을 제거한다.
-testWhileIdle Option이 true로 설정됐을 때만 이 동작을 수행한다.
-첫 번째 작업 시 minEvictableIdleTimeMillis 속성값을 초과하지 않은 Connection에 대해서 추가로 유효성 검사를 수행하는 것이다.

3)앞의 두 작업 이후 남아 있는 Connection의 개수가 minIdle 속성값보다 작으면 minIdle 속성값만큼 Connection을 생성해 유지한다.

4)예를 들어, testWhileIdle=true && timeBetweenEvictionRunMillis > 0이면 위의 3가지 역할을

다 수행하고, testWhileIdle=false && timeBetweenEvictionRunMillis > 0이면 두 번째 동작은 수행하지 않는다.

10. Evictor Thread는 동작 시에 Connection Pool에 잠금(lock)을 걸고 동작하기 때문에 너무 자주 실행하면 Service 실행에 부담을 줄 수 있다.
  1)또한 numTestsPerEvictionRun 값을 크게 설정하면 Evictor Thread가 검사해야 하는 Connection 개수가 많아져 잠금 상태에 있는 시간이 길어지므로 역시 Service 실행에 부담을 줄 수 있다.
  2)게다가 Connection 유효성 검사를 위한 Test Option(testOnBorrow, testOnReturn, testWhileIdle)을 어떻게 설정하느냐에 따라 Application의 안정성과 DBMS의 부하가 달라질 수 있다.
  3)그러므로 Evictor Thread와 Test Option을 사용할 때는 Database 관리자와 상의해서 사용하는 DBMS에 최적화될 수 있는 Option으로 설정해야 한다.

11. 기본값을 그대로 쓰기를 권장하는 Option
  -다음 값은 특별한 이유가 없다면 기본값을 쓰는 것을 권장한다.
  ①removeAbandoned Option
    -removeAbandoned Option은 false가 기본값이다.
    -removeAbandoned Option은 오랫동안 열려만 있고 Connection.close() Method가 호출되지 않는 Connection을 임의로 닫는 기능을 설정하는 Option이다.
    -removeAbandoned Option을 true로 설정하고 removeAbandonedTimeout Option에 허용할 최대 시간을 지정하면 Commons DBCP에서 자동으로 Connection.close() Method를 호출한다.
    -Application 개발자가 직접 JDBC API를 다루던 때는 Connection.close() Method 호출을 누락해서 전체 System의 자원을 고갈시키는 경우가 많았다.
    -근래에는 대부분의 Application이 Spring이나 MyBatis 등의 Framework를 사용하기 때문에 그럴 위험이 없다.
    -하지만 Connection 자원이 제대로 반납되지 않는다는 의심이 있다면 Commons DBCP 수준에서 방어하기보다는 문제 지점을 찾아서 근본적으로 수정해야 Application을 더 안정적으로 만들 수 있다.
    -removeAbandoned Option을 true로 설정하면 실행 시간이 긴 Query의 Connection을 의도하지 않게 닫는 부작용이 있다.
    -removeAbandoned Option은 기본값이 false로 사용하고, 오래 걸리는 Query는 JDBC Statement의 쿼리 타임아웃(query timeout) 등 다른 속성으로 제어하는 편이 바람직하다.

  ②defaultAutoCommit 속성
    -defaultAutoCommit 속성은 true가 기본값이다.
    -이 속성을 false로 설정하면 Connection을 Connection Pool에서 꺼낼 때 바로 setAutocommit(false) Method를 호출해서 Transaction을 시작하겠다는 의미다.
    -defaultAutoCommit 속성을 false로 설정하면 Application에서 Transaction 처리가 되어 있지 않은 경우에는 INSERT Query나 UPDATE Query가 제대로 반영되지 않는다.
    -따라서 기본값인 true를 그대로 사용하는 것이 무난하다.

12. 마치며
  1)Application에서 주로 실행되는 Query의 성격, 사용자가 대기 가능한 시간, DBMS와 Application Server의 자원, 발생 가능한 예외 상황, Tomcat 설정 등 Connection Pool Library를 제대로 설정하려면 많은 요소를 고려해야 한다.
  2)그래서 특별한 문제가 없으면 대충 설정하는 경우가 오히려 더 많다.
  3)그러나 Connection Pool Library는 전체 Application의 성능과 안정성에 큰 영향을 미치는 구성 요소이므로 사용자가 많은 Service라면 충분히 시간과 노력을 투자해서 고민해야 하는 영역이다.

13. Library Downloads
  1)The DBCP Component
    -http://commons.apache.org/proper/commons-dbcp/
    -commons-dbcp2-2.7.0-bin.zip

```
122     -commons-dbcp2-2.7.0.jar
123
124   2)Apache Commons Pool
125     -http://commons.apache.org/proper/commons-pool/
126     -commons-pool2-2.8.0-bin.zip
127     -commons-pool2-2.8.0.jar
128
129   3)The Logging Component
130     -http://commons.apache.org/proper/commons-logging/
131     -commons-logging-1.2-bin.zip
132     -commons-logging-1.2.jar
133
134   4)해당 jar를 Classpath 또는 Buildpath(Eclipse)에 저장한다.
135
136
137 14. Example
138   1)BasicDataSourceExample.java
139       /**
140       * To run this example, you'll want:
141       * commons-pool2-2.8.0.jar
142       * commons-dbcp2-2.7.5.jar
143       * commons-logging-1.2.jar
144       * ojdbc8.jar
145       * in your classpath.
146       */
147       import java.sql.Connection;
148       import java.sql.ResultSet;
149       import java.sql.SQLException;
150       import java.sql.Statement;
151       import javax.sql.DataSource;
152       import org.apache.commons.dbcp2.BasicDataSource;
153
154       public class BasicDataSourceExample {
155           public static void main(String[] args) {
156               System.out.println("Setting up data source.");
157               DataSource dataSource = setupDataSource();
158               System.out.println("Done.");
159
160               Connection conn = null;
161               Statement stmt = null;
162               ResultSet rset = null;
163
164               try {
165                   System.out.println("Creating connection.");
166                   conn = dataSource.getConnection();
167                   System.out.println("Creating statement.");
168                   stmt = conn.createStatement();
169                   System.out.println("Executing statement.");
170                   rset = stmt.executeQuery("SELECT * FROM dept");
171                   System.out.println("Results:");
172                   int numcols = rset.getMetaData().getColumnCount();
173                   while(rset.next()) {
174                       for(int i = 1 ; i <= numcols ; i++) {
175                           System.out.print("\t" + rset.getString(i));
176                       }
```

```
                System.out.println("");
            }
            printDataSourceStats(dataSource);
        } catch(SQLException e) {
            e.printStackTrace();
        } finally {
            try { if (rset != null) rset.close(); } catch(Exception e) { }
            try { if (stmt != null) stmt.close(); } catch(Exception e) { }
            try { if (conn != null) conn.close(); } catch(Exception e) { }
        }
    }

    public static DataSource setupDataSource() {
        BasicDataSource ds = new BasicDataSource();
        ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        ds.setUrl("jdbc:oracle:thin:@localhost:1521:ORCL");
        ds.setUsername("scott");
        ds.setPassword("tiger");
        return ds;
    }

    public static void printDataSourceStats(DataSource ds) {
        BasicDataSource bds = (BasicDataSource) ds;
        System.out.println("NumActive: " + bds.getNumActive());
        System.out.println("NumIdle: " + bds.getNumIdle());
    }

    public static void shutdownDataSource(DataSource ds) throws SQLException {
        BasicDataSource bds = (BasicDataSource) ds;
        bds.close();
    }
}
```

2)PoolingDataSourceExample.java

```
/**
* To run this example, you'll want:
* commons-pool2-2.8.0.jar
* commons-dbcp2-2.7.5.jar
* commons-logging-1.2.jar
* ojdbc8.jar
* the classes for your (underlying) JDBC driver
* in your classpath.
*/
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.apache.commons.pool2.ObjectPool;
import org.apache.commons.pool2.impl.GenericObjectPool;
import org.apache.commons.dbcp2.ConnectionFactory;
import org.apache.commons.dbcp2.PoolableConnection;
import org.apache.commons.dbcp2.PoolingDataSource;
import org.apache.commons.dbcp2.PoolableConnectionFactory;
import org.apache.commons.dbcp2.DriverManagerConnectionFactory;
```

```
232
233         public class PoolingDataSourceExample {
234             public static void main(String[] args) {
235                 System.out.println("Loading underlying JDBC driver.");
236                 try {
237                     Class.forName("oracle.jdbc.driver.OracleDriver");
238                 } catch (ClassNotFoundException e) {
239                     e.printStackTrace();
240                 }
241                 System.out.println("Done.");
242
243                 System.out.println("Setting up data source.");
244                 DataSource dataSource =
                    setupDataSource("jdbc:oracle:thin:@localhost:1521:ORCL");
245                 System.out.println("Done.");
246
247                 Connection conn = null;
248                 Statement stmt = null;
249                 ResultSet rset = null;
250
251                 try {
252                     System.out.println("Creating connection.");
253                     conn = dataSource.getConnection();
254                     System.out.println("Creating statement.");
255                     stmt = conn.createStatement();
256                     System.out.println("Executing statement.");
257                     rset = stmt.executeQuery("SELECT * FROM dept");
258                     System.out.println("Results:");
259                     int numcols = rset.getMetaData().getColumnCount();
260                     while(rset.next()) {
261                         for(int i=1;i<=numcols;i++) {
262                             System.out.print("\t" + rset.getString(i));
263                         }
264                         System.out.println("");
265                     }
266                 } catch(SQLException e) {
267                     e.printStackTrace();
268                 } finally {
269                     try { if (rset != null) rset.close(); } catch(Exception e) { }
270                     try { if (stmt != null) stmt.close(); } catch(Exception e) { }
271                     try { if (conn != null) conn.close(); } catch(Exception e) { }
272                 }
273             }
274
275             public static DataSource setupDataSource(String connectURI) {
276                 ConnectionFactory connectionFactory =
277                     new DriverManagerConnectionFactory(connectURI, "scott", "tiger");
278
279                 PoolableConnectionFactory poolableConnectionFactory =
280                     new PoolableConnectionFactory(connectionFactory, null);
281
282                 ObjectPool<PoolableConnection> connectionPool =
283                         new GenericObjectPool<>(poolableConnectionFactory);
284
285                 poolableConnectionFactory.setPool(connectionPool);
```

```
286
287             PoolingDataSource<PoolableConnection> dataSource =
288                     new PoolingDataSource<>(connectionPool);
289
290             return dataSource;
291         }
292     }
293
294 3)PoolingDriverExample.java
295     /**
296     * To run this example, you'll want:
297     * commons-pool-2.3.jar
298     * commons-dbcp-2.1.jar
299     * commons-logging-1.2.jar
300     * in your classpath.
301     */
302     import java.sql.Connection;
303     import java.sql.DriverManager;
304     import java.sql.ResultSet;
305     import java.sql.SQLException;
306     import java.sql.Statement;
307
308     import org.apache.commons.dbcp2.ConnectionFactory;
309     import org.apache.commons.dbcp2.DriverManagerConnectionFactory;
310     import org.apache.commons.dbcp2.PoolableConnection;
311     import org.apache.commons.dbcp2.PoolableConnectionFactory;
312     import org.apache.commons.dbcp2.PoolingDriver;
313     import org.apache.commons.pool2.ObjectPool;
314     import org.apache.commons.pool2.impl.GenericObjectPool;
315
316     public class PoolingDriverExample {
317         public static void main(String[] args) {
318             System.out.println("Loading underlying JDBC driver.");
319             try {
320                 Class.forName("oracle.jdbc.driver.OracleDriver");
321             } catch (ClassNotFoundException e) {
322                 e.printStackTrace();
323             }
324             System.out.println("Done.");
325
326             System.out.println("Setting up driver.");
327             try {
328                 setupDriver("jdbc:oracle:thin:@localhost:1521:ORCL");
329             } catch (Exception e) {
330                 e.printStackTrace();
331             }
332             System.out.println("Done.");
333
334             Connection conn = null;
335             Statement stmt = null;
336             ResultSet rset = null;
337
338             try {
339                 System.out.println("Creating connection.");
340                 conn =
```

```
            DriverManager.getConnection("jdbc:apache:commons:dbcp:example");
            System.out.println("Creating statement.");
            stmt = conn.createStatement();
            System.out.println("Executing statement.");
            rset = stmt.executeQuery("SELECT * FROM dept");
            System.out.println("Results:");
            int numcols = rset.getMetaData().getColumnCount();
            while(rset.next()) {
                for(int i=1;i<=numcols;i++) {
                    System.out.print("\t" + rset.getString(i));
                }
                System.out.println("");
            }
        } catch(SQLException e) {
            e.printStackTrace();
        } finally {
            try { if (rset != null) rset.close(); } catch(Exception e) { }
            try { if (stmt != null) stmt.close(); } catch(Exception e) { }
            try { if (conn != null) conn.close(); } catch(Exception e) { }
        }

        try {
            printDriverStats();
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            shutdownDriver();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void setupDriver(String connectURI) throws Exception {
        ConnectionFactory connectionFactory =
            new DriverManagerConnectionFactory(connectURI, "scott", "tiger");

        PoolableConnectionFactory poolableConnectionFactory =
            new PoolableConnectionFactory(connectionFactory, null);

        ObjectPool<PoolableConnection> connectionPool =
            new GenericObjectPool<>(poolableConnectionFactory);

    poolableConnectionFactory.setPool(connectionPool);

    Class.forName("org.apache.commons.dbcp2.PoolingDriver");
    PoolingDriver driver = (PoolingDriver)
    DriverManager.getDriver("jdbc:apache:commons:dbcp:");

        driver.registerPool("example",connectionPool);
    }

    public static void printDriverStats() throws Exception {
        PoolingDriver driver = (PoolingDriver)
```

```
            DriverManager.getDriver("jdbc:apache:commons:dbcp:");
394         ObjectPool<? extends Connection> connectionPool =
            driver.getConnectionPool("example");
395
396         System.out.println("NumActive: " + connectionPool.getNumActive());
397         System.out.println("NumIdle: " + connectionPool.getNumIdle());
398       }
399
400       public static void shutdownDriver() throws Exception {
401         PoolingDriver driver = (PoolingDriver)
            DriverManager.getDriver("jdbc:apache:commons:dbcp:");
402         driver.closePool("example");
403       }
404     }
405
406
407
```

408 Task 2. c3p0(https://www.mchange.com/projects/c3p0/)
409 1. What is c3p0?
410 1)c3p0 is an easy-to-use library for making traditional JDBC drivers "enterprise-ready" by augmenting them with functionality defined by the jdbc3 spec and the optional extensions to jdbc2.
411 2)As of version 0.9.5, c3p0 fully supports the jdbc4 spec.
412 3)In particular, c3p0 provides several useful services:
413 -A class whichs adapt traditional DriverManager-based JDBC drivers to the newer javax.sql.DataSource scheme for acquiring database Connections.
414 -Transparent pooling of Connection and PreparedStatements behind DataSources which can "wrap" around traditional drivers or arbitrary unpooled DataSources.
415
416 4)The library tries hard to get the details right:
417 -c3p0 DataSources are both Referenceable and Serializable, and are thus suitable for binding to a wide-variety of JNDI-based naming services.
418 -Statement and ResultSets are carefully cleaned up when pooled Connections and Statements are checked in, to prevent resource- exhaustion when clients use the lazy but common resource-management strategy of only cleaning up their Connections. (Don't be naughty.)
419 -The library adopts the approach defined by the JDBC 2 and 3 specification (even where these conflict with the library author's preferences).
420 --DataSources are written in the JavaBean style, offering all the required and most of the optional properties (as well as some non-standard ones), and no-arg constructors. All JDBC-defined internal interfaces are implemented (ConnectionPoolDataSource, PooledConnection, ConnectionEvent-generating Connections, etc.)
421 --You can mix c3p0 classes with compliant third-party implementations (although not all c3p0 features will work with external implementations of ConnectionPoolDataSource).
422
423 5)c3p0 hopes to provide DataSource implementations more than suitable for use by high-volume "J2EE enterprise applications". Please provide feedback, bug-fixes, etc!
424
425 2. Prerequisites
426 -c3p0-0.9.5.5 requires a level 1.6.x or above Java Runtime Environment.
427
428 3. Installation
429 -Place the files lib/c3p0-0.9.5.5.jar and lib/mchange-commons-java-0.2.19.jar

somewhere in your CLASSPATH (or any other place where your application's classloader will find it).

4. Downloads
1)https://sourceforge.net/projects/c3p0/
2)c3p0-0.9.5.5.bin.zip
3)c3p0-0.9.5.5.jar and lib/mchange-commons-java-0.2.19.jar

5. Example
1)DatabaseUtility.java

```
import java.beans.PropertyVetoException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class DatabaseUtility {
    public static ComboPooledDataSource getDataSource() throws
    PropertyVetoException {
        ComboPooledDataSource cpds = new ComboPooledDataSource();
        cpds.setDriverClass("oracle.jdbc.driver.OracleDriver");
        cpds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:ORCL");
        cpds.setUser("scott");
        cpds.setPassword("tiger");

        // Optional Settings
        cpds.setInitialPoolSize(5);
        cpds.setMinPoolSize(5);
        cpds.setAcquireIncrement(5);
        cpds.setMaxPoolSize(20);
        cpds.setMaxStatements(100);

        return cpds;
    }

    public static void main(String[] args) throws SQLException {
        Connection connection = null;
        PreparedStatement pstmt = null;
        ResultSet resultSet = null;
        try {
            ComboPooledDataSource dataSource = DatabaseUtility.getDataSource();
            connection = dataSource.getConnection();
            pstmt = connection.prepareStatement("SELECT * FROM dept");

            System.out.println("The Connection Object is of Class: " +
            connection.getClass());

            resultSet = pstmt.executeQuery();
            while (resultSet.next()) {
                System.out
                        .println(resultSet.getString(1) + "," + resultSet.getString(2) + "," +
                        resultSet.getString(3));
```

```
            }

        } catch (Exception e) {
            connection.rollback();
            e.printStackTrace();
        }
      }
    }
```

Task 3 HikariCP(https://github.com/brettwooldridge/HikariCP)

1. What is HikariCP?
   1)Fast, simple, reliable.
   2)HikariCP is a "zero-overhead" production ready JDBC connection pool.
   3)At roughly 130Kb, the library is very light.

2. Maven 환경에서
   -Java 8 thru 11 maven artifact:

```
    <dependency>
      <groupId>com.zaxxer</groupId>
      <artifactId>HikariCP</artifactId>
      <version>3.4.2</version>
    </dependency>
```

3. Downloads
   1)https://search.maven.org/search?q=com.zaxxer.hikaricp
   2)HikariCP-3.4.2.jar

4. Initialization
   1)You can use the HikariConfig class like so:

```
    HikariConfig config = new HikariConfig();
    config.setJdbcUrl("jdbc:mysql://localhost:3306/simpsons");
    config.setUsername("bart");
    config.setPassword("51mp50n");
    config.addDataSourceProperty("cachePrepStmts", "true");
    config.addDataSourceProperty("prepStmtCacheSize", "250");
    config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");

    HikariDataSource ds = new HikariDataSource(config);
```

   2)or directly instantiate a HikariDataSource like so:

```
    HikariDataSource ds = new HikariDataSource();
    ds.setJdbcUrl("jdbc:mysql://localhost:3306/simpsons");
    ds.setUsername("bart");
    ds.setPassword("51mp50n");
    ...
```

   3)or property file based:

```
    // Examines both filesystem and classpath for .properties file
    HikariConfig config = new HikariConfig("/some/path/hikari.properties");
    HikariDataSource ds = new HikariDataSource(config);
```

```
535    4)Example property file:
536      dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
537      dataSource.user=test
538      dataSource.password=test
539      dataSource.databaseName=mydb
540      dataSource.portNumber=5432
541      dataSource.serverName=localhost
542
543    5)or java.util.Properties based:
544
545      Properties props = new Properties();
546      props.setProperty("dataSourceClassName",
         "org.postgresql.ds.PGSimpleDataSource");
547      props.setProperty("dataSource.user", "test");
548      props.setProperty("dataSource.password", "test");
549      props.setProperty("dataSource.databaseName", "mydb");
550      props.put("dataSource.logWriter", new PrintWriter(System.out));
551
552      HikariConfig config = new HikariConfig(props);
553      HikariDataSource ds = new HikariDataSource(config);
554
555  5. Requirements
556    1)Java 8+ (Java 6/7 artifacts are in maintenance mode)
557    2)slf4j library(http://www.slf4j.org/)
```