**웰컴저축은행 필기시험**

- 성능과 최적화-

부서 명 :

교육생 명 :

1. 다음 코드는 product테이블에 있는 pirce의 컬럼의 값만 쿼리 조건에 따라 리턴 받아 실행하는 컬렉션 코드의 일부분입니다. 효율성 있는 코드로 변경하세요

**(하기 코드에 직접 수정사항 기재)**

java.sql.ResultSet

rs=stmt.executeQuery("select \* from product where rownum<=10 order by price desc");

ArrayList list=new ArrayList();

while(rs.next()){

int price=rs.getInt(“price”);

list.add(price);

}

[답]

java.sql.ResultSet rs=stmt.executeQuery("select price from product where rownum<=10 order by price desc");

int [] top10 price=new int[10];

top10price[i] =rs.getInt("price");

}

1. 다음 코드는 콜렉션 사용 코드의 일부입니다. 보다 효율적으로 고치세요.

**(하기 코드에 직접 수정사항 기재)**

public Member getElement(ArrayList<Member> list, int index){

Member m;

if(index > 0 && index <list.size() ){

m=list.get(index);

}else{

m=null;

}

return m;

}

[답]

public Member getElement(List<Member> list, int index){

returnlist.get(index);

}

3. 다음은 메모리 누수를 일으키는 코드입니다. 쿼리를 한 번 호출하기 위해서 매번 파일을 읽을 수 밖에 없는 구조이기 때문에 IO가 발생하면서 대기하는 IO wait가 발생하는 것을 피할 수가 없습니다. 소드의 코드를 찾아 튜닝 한다면 어느 라인을 어떻게 고치면 좋을까요?

**(하기 코드에 직접 수정사항 기재)**

import java.io.File;

import java.io.FileReader;

import java.util.HashMap;

public class MyManager {

private static String query = null;

public MyManager(String query) {

this.query=query;

}

public static String getSql(String fileName,String idSql) {

try {

int len = (int) (new File(fileName).length());

FileReader reader = new FileReader(fileName);

char buf[] = new char[len];

reader.read(buf);

query=new String(buf);

HashMap<String, String> document = createHashMap(query);

return document.get(idSql);

} catch(Exception ex) {

System.out.println(ex);

}

return null;

}

private static HashMap<String, String> createHashMap(String query) {

//... 생략

}

}

[답] private static String query = null; 를 private *String query = null; 으로 수정한다*

4. Custom Exception은 값을 검사해 하위 비지니스 로직을 수행할 수 없는 것은 현재 값이 어떤 call stack을 가지는지에 대한 trace는 필요하지 않습니다 exception stack trace를 통해 생성되는 메소드로 필요없는 trace로 작성하지 않도록 재정의 해서 사용되는 메소드는 무엇인가요?

[답]

Throwable.fillInStackTrace

5. 문자열을 사용하는 다음 예제를 보다 효율적으로 변경하시오.

**(하기 코드에 직접 수정사항 기재)**

String result = “”;

for (int i = 0; i < 100000; i++) {

result += “\*”;

}

​

[답]

StringBuilder result = new StringBuilder();

for (int i = 0; i < 100000; i++) {

result.append(“\*”);

}

6. 자바에서 문자열을 다루기 위해서는 String, StringBuffer, StringBuilder 클래스를 사용한다. StringBuffer와 StringBuilder 클래스에 관한 설명 중 잘못된 것은?

1) StirngBuffer, StringBuilder 클래스 모두 각 메소드 마다 Synchronized keyword가 처리 되어 있다.

2) 멀티스레드 환경에서 동기화 보장을 위해서는 StringBuffer 클래스를 사용한다.

3) 단일스레드 환경에서는 StringBuilder 클래스를 사용하는 것이 좋다.

4) 단일스레드 환경에서 StringBuffer를 사용하면 동기화 처리로 인해 성능이 좋지 않다.

​

[답] 1

​

7. 자바에서 파일을 복사하는 방법으로는 Stream(InputStream, OutputStream), Buffer, Channel을 이용한 방법이 있다. 다음중 성능이 낮은 것부터 높은 순서로 나열 한 것은?

1) Stream < Buffer < Channel

2) Buffer< Stream < Channel

3) Buffer < Channel < Stream

4) Channel < Buffer < Stream

[답] 1

​

8. Java IO는 프로그램 성능에 가장 큰 영향을 끼치는 요소 중의 하나이다. IO를 효율적으로 사용하기 위한 방법 중 잘못 된 것은?

1) 일반적인 Stream을 그냥 사용하는 것 보다 Buffer 관련 API를 사용하는 것이 좋다.

2) DB의 쿼리나 설정 파일들은 중요한 정보이기 때문에 요청할 때 마다 새롭게 읽어야만 한다.

3) 자바 IO는 자바에서 커널측에 파일 관련 정보를 요청하는 방식이었기 때문에, 이것을 개선한 JDK 1.4에 추가된 NIO를 이용해서 자바에서 직접 통제하는 것이 좋다.

4) File 클래스의 lastModified() 메소드의 사용은 지양하는 것이 좋다.

[답] 2

9. JDBC API 중 ResultSet의 last() 메소드에 대한 설명 중 잘못 된 것은?

1) last() 메소드의 수행시간은 데이터의 건수와 DB의 통신 속도에 따라 다르다.

2) 데이터가 많을수록 대기시간이 증가하기 때문에 속도차이가 크다.

3) 행의 개수를 셀 때에는 count(\*) 쿼리를 한번 더 실행 하는 것이 좋다.

4) last() 메소드는 ResultSet의 Cursor 위치를 마지막으로 이동하기 위한 메소드이기 때문에 성능에는 영향이 없다.

[답] 4

10. JDBC API 사용 시 성능향상을 위해서는 Connection Pool 사용이 권장된다. 다음 중 Connection Pool 사용에 대해 잘못 된 것은?

1) 클라이언트의 요청이 있을 때 마다 매번 Connection 객체를 생성하지 않도록 해 준다.

2) Connection Pool의 크기를 설정하지 않으면 1개의 기본 값이 설정 되기 때문에, 메모리 낭비를 줄일 수 있다.

3) Connection, Statement,  ResultSet 등과 같은 API에서 객체 사용 후 close를 하지 않으면 Connection Pool 객체가 반환되지 않는다.

4) 너무 자주 commit을 발생하는 것은 Connection Pool의 성능에 영향을 줄 수 있다.

[답] 2