

mvdb: Relationale Operationen mit Map Reduce

Tobias Lerch, Yanick Eberle, Pascal Schwarz

21. Mai 2013

1 Einleitung

In dieser Aufgabe geht es darum, einfache relationale Operationen mittels Map Reduce umzusetzen. Die Daten, auf welchen die Abfragen ausgeführt werden sollen, liegen dabei in CSV-Form vor.

Es sind Daten aus den Tabellen *mitglieder* und *registrierungen* abzufragen. Die CSV-Dateien werden hier der Übersichtlichkeit wegen gelistet.

Inhalt von *Registrierungen*:

1	M001;F1;A01;07.11.2007
2	M004;F1;A01;29.06.2007
3	M005;F1;A01;04.07.2007
4	M002;F3;A02;17.05.2007
5	M004;F3;A01;29.06.2007
6	M005;F3;A02;01.12.2007
7	M003;F1;A03;07.11.2007
8	M001;F2;A04;12.10.2007
9	M006;F4;A05;16.05.2007

Inhalt von *Mitglieder*:

1	M001;A. Huber; Basel;15.05.1978
2	M004;S. Baumann; Bern;21.03.1982
3	M005;U. Schoch; Basel;01.09.1975
4	M002;E. Mueller; Bern;30.07.1985
5	M003;K. Buser; Riehen;13.04.1972
6	M006;E. Mueller; Reinach BL;28.10.1980

2 Group By mit Count

Diese Operation soll mittels Map Reduce umgesetzt werden:

```
1 SELECT mnr, COUNT(*)
2 FROM registrierungen
3 GROUP BY mnr;
```

Die Lösung dieser Aufgabe haben wir sehr ähnlich umgesetzt wie die Aufgabe, bei der es darum ging, die Anzahl Vorkommnisse eines Wortes in den Eingabedaten zu zählen. Im Wesentlichen übernehmen die Komponenten die folgenden Aufgaben:

mapper Spalte *mnr* aus jeder Zeile extrahieren und diese als Key weitergeben (die Value spielt keine Rolle).

reducer Die Anzahl Values pro Key (pro Wert in *mnr* wird einmal ein reducer mit diesem Wert als Key aufgerufen) zählen und diese Anzahl zusammen mit dem Key ausgeben.

Die Aufgabe haben wir dann mit dem folgenden Code gelöst:

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.conf.Configuration;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.Mapper;
9 import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12 import org.apache.hadoop.util.GenericOptionsParser;
13
14 public class GroupCount {
15
16     public static void main(String[] args) throws IOException,
17         ClassNotFoundException, InterruptedException {
18         Configuration conf = new Configuration();
19         String[] otherArgs = new GenericOptionsParser(conf, args)
20             .getRemainingArgs();
21         if (otherArgs.length != 2) {
22             System.err.println(" Usage: groupcount.jar <in> <out>");
23             System.exit(2);
24         }
25         Job job = new Job(conf, "groupcount");
26         job.setMapperClass(GCountMapper.class);
27         job.setReducerClass(GCountReducer.class);
28         job.setOutputKeyClass(Text.class);
29         job.setOutputValueClass(LongWritable.class);
30         FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
31         FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
32         System.exit(job.waitForCompletion(true) ? 0 : 1);
33     }
34
35     public static class GCountMapper extends Mapper<LongWritable, Text, Text,
36         LongWritable> {
37         private static final LongWritable one = new LongWritable(1);
38         @Override
39         protected void map(LongWritable key, Text val,
40             org.apache.hadoop.mapreduce.Mapper<LongWritable,
41                 Text, Text, LongWritable>.Context ctx)
42             throws IOException, InterruptedException {
43             String line = val.toString();
44             String[] fields = line.split(";");
```

```

42         ctx.write(new Text(fields[0]), one);
43     }
44 }
45
46 public static class GCountReducer extends Reducer<Text, LongWritable, Text,
47     LongWritable> {
48     @Override
49     protected void reduce(Text key, Iterable<LongWritable> vals,
50         org.apache.hadoop.mapreduce.Reducer<Text,
51             LongWritable, Text, LongWritable>.Context ctx)
52         throws IOException, InterruptedException {
53         long count = 0;
54         for (LongWritable val : vals) count++;
55         ctx.write(key, new LongWritable(count));
56     }
57 }

```

Nach dem Platzieren der Datei *registrierungen.csv* in den Ordner *InGCount* und dem Exportieren des Codes in eine JAR-Datei können wir mit dem folgenden Aufruf die Abfrage ausführen:

```

1 [iso@iso-t530arch hadoop-0.22.0]$ bin/hadoop jar groupcount.jar InGCount OutGCount
2 13/05/15 15:18:21 INFO jvm.JvmMetrics: Initializing JVM Metrics with
   processName=JobTracker, sessionId=
3 13/05/15 15:18:21 INFO util.NativeCodeLoader: Loaded the native-hadoop library
4 13/05/15 15:18:21 WARN conf.Configuration: mapred.used.genericoptionsparser is
   deprecated. Instead, use mapreduce.client.genericoptionsparser.used
5 13/05/15 15:18:21 WARN mapreduce.JobSubmitter: No job jar file set. User classes may
   not be found. See Job or Job#setJar(String).
6 13/05/15 15:18:21 INFO input.FileInputFormat: Total input paths to process : 1
7 13/05/15 15:18:21 INFO mapreduce.JobSubmitter: number of splits:1
8 13/05/15 15:18:21 INFO mapreduce.Job: Running job: job_local_0001
9 13/05/15 15:18:21 INFO mapred.LocalJobRunner: Waiting for map tasks
10 13/05/15 15:18:21 INFO mapred.LocalJobRunner: Starting task:
   attempt_local_0001_m_000000_0
11 13/05/15 15:18:21 INFO util.ProcessTree: setsid exited with exit code 0
12 13/05/15 15:18:21 INFO mapred.Task: Using ResourceCalculatorPlugin :
   org.apache.hadoop.mapreduce.util.LinuxResourceCalculatorPlugin@68c0e5e1
13 13/05/15 15:18:21 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
14 13/05/15 15:18:21 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
15 13/05/15 15:18:21 INFO mapred.MapTask: soft limit at 83886080
16 13/05/15 15:18:21 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
17 13/05/15 15:18:21 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
18 13/05/15 15:18:21 INFO mapred.LocalJobRunner:
19 13/05/15 15:18:21 INFO mapred.MapTask: Starting flush of map output
20 13/05/15 15:18:21 INFO mapred.MapTask: Spilling map output
21 13/05/15 15:18:21 INFO mapred.MapTask: bufstart = 0; bufend = 117; bufvoid = 104857600
22 13/05/15 15:18:21 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend =
   26214364(104857456); length = 33/6553600
23 13/05/15 15:18:21 INFO mapred.MapTask: Finished spill 0
24 13/05/15 15:18:21 INFO mapred.Task: Task:attempt_local_0001_m_000000_0 is done. And
   is in the process of committing
25 13/05/15 15:18:21 INFO mapred.LocalJobRunner: map > sort
26 13/05/15 15:18:21 INFO mapred.Task: Task 'attempt_local_0001_m_000000_0' done.
27 13/05/15 15:18:21 INFO mapred.LocalJobRunner: Finishing task:
   attempt_local_0001_m_000000_0
28 13/05/15 15:18:21 INFO mapred.LocalJobRunner: Map task executor complete.
29 13/05/15 15:18:21 INFO mapred.Task: Using ResourceCalculatorPlugin :
   org.apache.hadoop.mapreduce.util.LinuxResourceCalculatorPlugin@2467149d
30 13/05/15 15:18:21 INFO mapred.Merger: Merging 1 sorted segments
31 13/05/15 15:18:21 INFO mapred.Merger: Down to the last merge-pass, with 1 segments
   left of total size: 130 bytes
32 13/05/15 15:18:21 INFO mapred.LocalJobRunner:
33 13/05/15 15:18:21 WARN conf.Configuration: mapred.skip.on is deprecated. Instead, use
   mapreduce.job.skiprecords

```

```

34 13/05/15 15:18:21 INFO mapred.Task: Task:attempt_local-0001_r-000000-0 is done. And
    is in the process of committing
35 13/05/15 15:18:21 INFO mapred.LocalJobRunner:
36 13/05/15 15:18:21 INFO mapred.Task: Task attempt_local-0001_r-000000-0 is allowed to
    commit now
37 13/05/15 15:18:21 INFO output.FileOutputCommitter: Saved output of task
    'attempt_local-0001_r-000000-0' to OutGCount
38 13/05/15 15:18:21 INFO mapred.LocalJobRunner: reduce > sort
39 13/05/15 15:18:21 INFO mapred.Task: Task 'attempt_local-0001_r-000000-0' done.
40 13/05/15 15:18:22 INFO mapreduce.Job: map 100% reduce 100%
41 13/05/15 15:18:22 INFO mapreduce.Job: Job complete: job_local-0001
42 13/05/15 15:18:22 INFO mapreduce.Job: Counters: 21
43     FileInputFormatCounters
44         BYTES_READ=207
45     FileSystemCounters
46         FILE_BYTES_READ=951
47         FILE_BYTES_WRITTEN=134336
48     Map-Reduce Framework
49         Combine input records=0
50         Combine output records=0
51         CPU_MILLISECONDS=0
52         Failed Shuffles=0
53         GC time elapsed (ms)=0
54         Map input records=9
55         Map output bytes=117
56         Map output records=9
57         Merged Map outputs=0
58         PHYSICALMEMORY_BYTES=0
59         Reduce input groups=6
60         Reduce input records=9
61         Reduce output records=6
62         Reduce shuffle bytes=0
63         Shuffled Maps =0
64         Spilled Records=18
65         SPLIT_RAW_BYTES=143
66         VIRTUALMEMORY_BYTES=0

```

Die Anzeige des Output-Files zeigt das Resultat der Abfrage:

```

1 [iso@iso-t530arch hadoop-0.22.0]$ cat OutGCount/part-r-00000
2 M001      2
3 M002      1
4 M003      1
5 M004      2
6 M005      2
7 M006      1

```

Dieses Resultat deckt sich nicht nur mit dem erwarteten Resultat der SQL-Abfrage, sondern auch mit dem Statusoutput am Ende der Hadoop-Ausführung. Insgesamt sind 9 Datensätze in *registrierungen.csv* enthalten (*Map input records*, *Map Output records* und *Reduce input records*) und das Resultat der Abfrage enthält noch 6 Einträge, da es 6 verschiedene Werte in der Spalte *mnr* gibt (*Reduce input groups* und *Reduce output records*).

3 Join

In dieser Aufgabe soll die folgende Operation mittels Map Reduce umgesetzt werden:

```
1 SELECT *
2 FROM registrierungen
3 JOIN mitglieder USING (mnr);
```

Um den Join umzusetzen sehen wir die folgende Aufgabenteilung zwischen Mapper und Reducer vor:

mapper Extrahiert den Join-Key (*mnr*) aus den Zeilen und gibt diesen als Key weiter. Als Value wird der restliche Inhalt der Zeile zusammen mit einem Hinweis weitergegeben. Der Hinweis sagt aus, aus welcher Tabelle der Datensatz stammt.

reducer Setzt die Zeilen des Results des Joins zusammen. Dabei muss für jeden Wert des Join-Keys jede Zeile aus *mitglieder* mit jeder Zeile aus *registrierungen* kombiniert werden. Dass *mnr* ein Primary-Key von *mitglieder* ist, und daher aus *mitglieder* nur eine einzige Zeile pro *mnr* geliefert wird, ignorieren wir, da die Information, dass *mnr* in *mitglieder* eindeutig ist, aus den CSV Files nicht direkt hervorgeht.

Für die beiden Input-Tabellen wurden separate Mapper-Klassen erstellt. Diese wurden mittels *MultipleInputs* dem Job hinzugefügt. Wir führen für die Angabe der Quellfiles der beiden Tabellen einen weiteren Parameter ein. Der erste Parameter gibt hier den Pfad zur Mitglieder-Datei, der zweite Parameter die Registrierungen-Datei und das dritte Argument das Ausgabeverzeichnis an.

```
1 join.jar <in mitglieder> <in registrierungen> <out>
```

Um dem reducer mitteilen zu können, aus welcher Tabelle eine Zeile stammt, haben wir eine zusätzliche Klasse (*MapPair*) definiert. Sie beinhaltet zwei Strings, der String *table* gibt dabei an, aus welcher Tabelle die Daten im String *record* stammen. Damit Instanzen eigener Klassen verwendet werden können, müssen diese von Hadoop serialisiert werden können. Dafür ist die Implementierung des Interfaces *Writable* notwendig. Die darin enthaltenen Methoden schreiben resp. lesen Daten in einen *DataOutput* resp. aus einem *DataInput*.

```
1 import java.io.DataInput;
2 import java.io.DataOutput;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import org.apache.hadoop.conf.Configuration;
8 import org.apache.hadoop.fs.Path;
9 import org.apache.hadoop.io.LongWritable;
10 import org.apache.hadoop.io.Text;
11 import org.apache.hadoop.io.Writable;
12 import org.apache.hadoop.mapreduce.Job;
13 import org.apache.hadoop.mapreduce.Mapper;
14 import org.apache.hadoop.mapreduce.Reducer;
15 import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
16 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18 import org.apache.hadoop.util.GenericOptionsParser;
19
20 public class Join {
21     private static final String tableM = "Mitglieder";
22     private static final String tableR = "Registrierungen";
23 }
```

```

24 public static void main(String[] args) throws ClassNotFoundException ,
    IOException, InterruptedException {
25     Configuration conf = new Configuration();
26     String[] otherArgs = new GenericOptionsParser(conf, args)
27         .getRemainingArgs();
28     if (otherArgs.length != 3) {
29         System.err.println(" Usage: join.jar <in mitglieder> <in
            registrierungen> <out>");
30         System.exit(2);
31     }
32     Job job = new Job(conf, "join");
33     MultipleInputs.addInputPath(job, new Path(otherArgs[0]),
        TextInputFormat.class, MitgliederMapper.class);
34     MultipleInputs.addInputPath(job, new Path(otherArgs[1]),
        TextInputFormat.class, RegistrierungenMapper.class);
35     job.setReducerClass(MRJoinReducer.class);
36     job.setMapOutputValueClass(MapPair.class);
37     job.setOutputKeyClass(Text.class);
38     job.setOutputValueClass(Text.class);
39     FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));
40     System.exit(job.waitForCompletion(true) ? 0 : 1);
41 }
42
43 public static class MitgliederMapper extends Mapper<LongWritable, Text, Text,
    MapPair> {
44     @Override
45     protected void map(LongWritable key, Text value,
46         org.apache.hadoop.mapreduce.Mapper<LongWritable,
47             Text, Text, MapPair>.Context context)
48         throws IOException, InterruptedException {
49         String[] fields = value.toString().split(";");
50
51         Text mnr = new Text(fields[0]);
52         String record = fields[1] + "\t" + fields[2] + "\t" +
            fields[3];
53
54         context.write(mnr, new MapPair(tableM, record));
55     }
56 }
57
58 public static class RegistrierungenMapper extends Mapper<LongWritable, Text,
    Text, MapPair> {
59     @Override
60     protected void map(LongWritable key, Text value,
61         org.apache.hadoop.mapreduce.Mapper<LongWritable,
62             Text, Text, MapPair>.Context context)
63         throws IOException, InterruptedException {
64         String[] fields = value.toString().split(";");
65
66         Text mnr = new Text(fields[0]);
67         String record = fields[1] + "\t" + fields[2] + "\t" +
            fields[3];
68
69         context.write(mnr, new MapPair(tableR, record));
70     }
71 }
72
73 public static class MRJoinReducer extends Reducer<Text, MapPair, Text, Text> {
74     @Override
75     protected void reduce(Text key, Iterable<MapPair> vals,
76         org.apache.hadoop.mapreduce.Reducer<Text, MapPair,
77             Text, Text>.Context ctx)
78         throws IOException, InterruptedException {
79         // trennung von Zeilen aus Mitglieder und Registrierungen
80         List<String> recordsM = new ArrayList<>();
81         List<String> recordsR = new ArrayList<>();
82
83         for (MapPair pair : vals) {

```

```

82         if (tableM.equals(pair.table)) {
83             recordsM.add(pair.record);
84         } else if (tableR.equals(pair.table)) {
85             recordsR.add(pair.record);
86         }
87     }
88
89     // alle Zeilen aus Mitgliedern mit allen Zeilen aus
90     // Registrierungen kombinieren
91     for (String recordM : recordsM) {
92         for (String recordR : recordsR) {
93             ctx.write(key, new Text(recordM + "\t" +
94                                     recordR));
95         }
96     }
97 }
98
99 /**
100  * Hilfsklasse um sowohl eine Table-ID als auch einen Record
101  * von map an reduce weiterreichen zu koennen
102  */
103 public static class MapPair implements Writable {
104     public String table, record;
105     public MapPair(String table, String record) {
106         this.table = table; this.record = record;
107     }
108
109     // default constr wird wegen serialisierbarkeit benoetigt
110     public MapPair() {}
111
112     @Override
113     public void readFields(DataInput arg0) throws IOException {
114         table = arg0.readUTF(); record = arg0.readUTF();
115     }
116     @Override
117     public void write(DataOutput arg0) throws IOException {
118         arg0.writeUTF(table); arg0.writeUTF(record);
119     }
120     public static MapPair read(DataInput in) throws IOException {
121         final MapPair mp = new MapPair();
122         mp.readFields(in); return mp;
123     }
124 }
125 }

```

Nach dem Exportieren der Klasse in ein JAR File und dem Bereitstellen der Eingabedaten können wir Hadoop mit diesem Job aufrufen:

```

1 [iso@iso-t530arch hadoop-0.22.0]$ bin/hadoop jar join.jar InJoin/mitglieder.csv
2   InJoin/registrierungen.csv OutJoin
3 13/05/15 17:27:34 INFO jvm.JvmMetrics: Initializing JVM Metrics with
4   processName=JobTracker, sessionId=
5 13/05/15 17:27:34 INFO util.NativeCodeLoader: Loaded the native-hadoop library
6 13/05/15 17:27:34 WARN conf.Configuration: mapred.used.genericoptionsparser is
7   deprecated. Instead, use mapreduce.client.genericoptionsparser.used
8 13/05/15 17:27:34 WARN mapreduce.JobSubmitter: No job jar file set. User classes may
9   not be found. See Job or Job#setJar(String).
10 13/05/15 17:27:34 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with
11   processName=JobTracker, sessionId= - already initialized
12 13/05/15 17:27:34 INFO input.FileInputFormat: Total input paths to process : 1
13 13/05/15 17:27:34 INFO input.FileInputFormat: Total input paths to process : 1
14 13/05/15 17:27:34 INFO mapreduce.JobSubmitter: number of splits:2
15 13/05/15 17:27:34 INFO mapreduce.Job: Running job: job_local-0001
16 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Waiting for map tasks
17 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Starting task:
18   attempt_local-0001_m-000000.0
19 13/05/15 17:27:34 INFO util.ProcessTree: setsid exited with exit code 0

```

```

14 13/05/15 17:27:34 INFO mapred.Task: Using ResourceCalculatorPlugin :
    org.apache.hadoop.mapreduce.util.LinuxResourceCalculatorPlugin@49ec9f34
15 13/05/15 17:27:34 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
16 13/05/15 17:27:34 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
17 13/05/15 17:27:34 INFO mapred.MapTask: soft limit at 83886080
18 13/05/15 17:27:34 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
19 13/05/15 17:27:34 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
20 13/05/15 17:27:34 INFO mapred.LocalJobRunner:
21 13/05/15 17:27:34 INFO mapred.MapTask: Starting flush of map output
22 13/05/15 17:27:34 INFO mapred.MapTask: Spilling map output
23 13/05/15 17:27:34 INFO mapred.MapTask: bufstart = 0; bufend = 369; bufvoid = 104857600
24 13/05/15 17:27:34 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend =
    26214364(104857456); length = 33/6553600
25 13/05/15 17:27:34 INFO mapred.MapTask: Finished spill 0
26 13/05/15 17:27:34 INFO mapred.Task: Task:attempt_local-0001-m-000000-0 is done. And
    is in the process of committing
27 13/05/15 17:27:34 INFO mapred.LocalJobRunner: map > sort
28 13/05/15 17:27:34 INFO mapred.Task: Task 'attempt_local-0001-m-000000-0' done.
29 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Finishing task:
    attempt_local-0001-m-000000-0
30 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Starting task:
    attempt_local-0001-m-000001-0
31 13/05/15 17:27:34 INFO mapred.Task: Using ResourceCalculatorPlugin :
    org.apache.hadoop.mapreduce.util.LinuxResourceCalculatorPlugin@47c44bd4
32 13/05/15 17:27:34 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
33 13/05/15 17:27:34 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
34 13/05/15 17:27:34 INFO mapred.MapTask: soft limit at 83886080
35 13/05/15 17:27:34 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
36 13/05/15 17:27:34 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
37 13/05/15 17:27:34 INFO mapred.LocalJobRunner:
38 13/05/15 17:27:34 INFO mapred.MapTask: Starting flush of map output
39 13/05/15 17:27:34 INFO mapred.MapTask: Spilling map output
40 13/05/15 17:27:34 INFO mapred.MapTask: bufstart = 0; bufend = 277; bufvoid = 104857600
41 13/05/15 17:27:34 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend =
    26214376(104857504); length = 21/6553600
42 13/05/15 17:27:34 INFO mapred.MapTask: Finished spill 0
43 13/05/15 17:27:34 INFO mapred.Task: Task:attempt_local-0001-m-000001-0 is done. And
    is in the process of committing
44 13/05/15 17:27:34 INFO mapred.LocalJobRunner: map > sort
45 13/05/15 17:27:34 INFO mapred.Task: Task 'attempt_local-0001-m-000001-0' done.
46 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Finishing task:
    attempt_local-0001-m-000001-0
47 13/05/15 17:27:34 INFO mapred.LocalJobRunner: Map task executor complete.
48 13/05/15 17:27:34 INFO mapred.Task: Using ResourceCalculatorPlugin :
    org.apache.hadoop.mapreduce.util.LinuxResourceCalculatorPlugin@6953dc4b
49 13/05/15 17:27:34 INFO mapred.Merger: Merging 2 sorted segments
50 13/05/15 17:27:34 INFO mapred.Merger: Down to the last merge-pass, with 2 segments
    left of total size: 666 bytes
51 13/05/15 17:27:34 INFO mapred.LocalJobRunner:
52 13/05/15 17:27:34 WARN conf.Configuration: mapred.skip.on is deprecated. Instead, use
    mapreduce.job.skiprecords
53 13/05/15 17:27:34 INFO mapred.Task: Task:attempt_local-0001-r-000000-0 is done. And
    is in the process of committing
54 13/05/15 17:27:34 INFO mapred.LocalJobRunner:
55 13/05/15 17:27:34 INFO mapred.Task: Task attempt_local-0001-r-000000-0 is allowed to
    commit now
56 13/05/15 17:27:34 INFO output.FileOutputCommitter: Saved output of task
    'attempt_local-0001-r-000000-0' to OutJoin
57 13/05/15 17:27:34 INFO mapred.LocalJobRunner: reduce > sort
58 13/05/15 17:27:34 INFO mapred.Task: Task 'attempt_local-0001-r-000000-0' done.
59 13/05/15 17:27:35 INFO mapreduce.Job: map 100% reduce 100%
60 13/05/15 17:27:35 INFO mapreduce.Job: Job complete: job_local-0001
61 13/05/15 17:27:35 INFO mapreduce.Job: Counters: 21
62     FileInputFormatCounters
63         BYTES_READ=402
64     FileSystemCounters
65         FILE_BYTES_READ=4690
66         FILE_BYTES_WRITTEN=209280
67 Map-Reduce Framework

```



```

68      Combine input records=0
69      Combine output records=0
70      CPU_MILLISECONDS=0
71      Failed Shuffles=0
72      GC time elapsed (ms)=0
73      Map input records=15
74      Map output bytes=646
75      Map output records=15
76      Merged Map outputs=0
77      PHYSICAL_MEMORY_BYTES=0
78      Reduce input groups=6
79      Reduce input records=15
80      Reduce output records=9
81      Reduce shuffle bytes=0
82      Shuffled Maps =0
83      Spilled Records=30
84      SPLIT_RAW_BYTES=544
85      VIRTUAL_MEMORY_BYTES=0

```

Die Ausgabe des Map Reduce Jobs entspricht unseren Erwartungen:

```

1  [iso@iso-t530arch hadoop-0.22.0]$ cat OutJoin/part-r-00000
2  M001    A. Huber      Basel      15.05.1978    F2        A04        12.10.2007
3  M001    A. Huber      Basel      15.05.1978    F1        A01        07.11.2007
4  M002    E. Mueller    Bern       30.07.1985    F3        A02        17.05.2007
5  M003    K. Buser       Riehen     13.04.1972    F1        A03        07.11.2007
6  M004    S. Baumann      Bern       21.03.1982    F3        A01        29.06.2007
7  M004    S. Baumann      Bern       21.03.1982    F1        A01        29.06.2007
8  M005    U. Schoch       Basel      01.09.1975    F3        A02        01.12.2007
9  M005    U. Schoch       Basel      01.09.1975    F1        A01        04.07.2007
10 M006    E. Mueller      Reinach BL 28.10.1980    F4        A05        16.05.2007

```