UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and
Computer Science

Bachelor's Degree in
Computer Science

Final Dissertation

Titolo

Sottotitolo (alcune volte lungo - opzionale)

Supervisor
Prof. Gabriel Mark Kuper

Student
Daniele Parmeggiani

Academic year 2020/2021

Dedication

Contents

Abstract

This Theses contains the work I've carried out during the internship, part of the Bachelor's Degree in Computer Science curriculum, at SpazioDati Srl, a technology company based in Trento, Italy.

The work pertained the development of a Change Data Capture system.

My work will be based on a need for the company to move some part of their database to a new database, due to load constraints on the former database. The transfer of data would be carried out using Kafka streams, linked to changes in the main database.

The new database would serve primarily their sales team, which would use it for usage analytics for existing customers. This database will co-exist with the previous one, i.e. it will not replace the parts of it that it's replicating, but it would instead keep being updated with the change events triggering in the main database.

As the contents of this document pertains the work I've carried out within SpazioDati, all information relevant to their intellectual properties has been anonymized so as to not disclose it. The methods for such anonymization have been portrayed in the following Chapter 2. Nowhere in this document their customers' data is shown or cited, partly, or in whole.

In Appendix A the listings of source code are provided.

Chapter 1

Introduction

Chapter 2

Non Disclosure

In this document several information internal to SpazioDati is referenced.

In order to avoid disclosing information relevant to their intellectual property, the tables referenced in this Theses have been anonymized.

In particular the names of the eleven tables referenced have been replaced with the Greek letters α, β, γ, δ, ϵ, ζ, η, θ, ι, κ, and λ.

The contents of this Theses have been reviewed by SpazioDati and they have acknowledged this document does not contain information sensitive to either their customers or their intellectual properties.

Chapter 3

Technologies Employed

3.1 Apache Kafka

3.1.1 Kafka Connect

3.1.2 Kafka Streams

3.2 PostgreSQL

3.2.1 Write Ahead Log

Chapter 4

Time Travel Functionality

Every table has one representation in the source database and another representation in the destination database. To avoid ambiguity, let us introduce the following notation:

- $\text{source}[\omega]$ and $\text{dest}[\omega]$ refer to the representation of some table ω at the source and destination databases respectively;
- $\text{source}[*]$ and $\text{dest}[*]$ refer to the representations of all tables collectively (i.e. the whole databases), at the source and destination databases respectively.

We will assume the following, without proof:

Assumption 1. *$\text{source}[*]$ is inherently consistent.*

The consistency of the source database should be assumed because a replication of a database can only be as consistent as the original database, i.e. there is no way to have consistent data, starting from inconsistent data.

Additionally, such is safe to assume, since the database constraints are checked at the source by the PostgreSQL server.

Generally, $\text{dest}[\alpha] := \pi_S(\text{source}[\alpha])$ over some subset S of its columns, with one exception for table γ:

$$\text{dest}[\gamma] := \pi_{\text{dest}[\alpha].\text{id},\text{source}[\gamma].*}(\text{source}[\gamma] \bowtie_{\alpha.\text{email}=\gamma.\text{email}} \text{dest}[\alpha])$$

4.1 Empty Validity Ranges

When dealing with ranges, one should account for the fact that $\emptyset$ is indeed a valid range:

$$[t, t) = \emptyset, \forall t$$

This has the inherent meaning, in our domain, that two different records for the same table were issued at the exact same time t.

At first, this may seem an issue that could possibly be dismissed as very unlikely. Nevertheless, the presence of database transactions makes the significance of this scenario become evident. Within a transaction, different queries are evaluated and later performed as a single atomic[1] operation, thus resulting in several records being issued at the same instant.

[1] ??? explain atomic

Chapter 5

Future Work

5.1 Keeping up to date

With the source db changes

5.2 Metrics

5.3 Batching

Chapter 6

Conclusion

Bibliography

[1] Dollimore J. e Kindberg T Coulouris G. F. *Distributed Systems: concepts and Design*. second edition. Addison-Wesley, 1994.

[2] Triggs B. Dalal N. "Histograms of Oriented Gradients for Human Detection". In: *Computer Vision and Pattern Recognition (CVPR)*. San Diego, USA, June 2005, pp. 886–893.

[3] *ICT business*. URL: `http://www.ictbusiness.it/` (visited on June 15, 2015).

[4] Donoho D. L. "Compressed Sensing". In: *IEEE Trans. Inf. Theory* 52.4 (2006), pp. 1289–1306.

Appendix A

Listings of Source Files

Listing A.1: source/python-example.py

```
1 import numpy as np
2
3 def incmatrix(genl1,genl2):
4     m = len(genl1)
5     n = len(genl2)
6     M = None #to become the incidence matrix
7     VT = np.zeros((n*m,1), int)  #dummy variable
8
9     # compute the bitwise xor matrix
10    M1 = bitxormatrix(genl1)
11    M2 = np.triu(bitxormatrix(genl2),1)
12
13    for i in range(m-1):
14        for j in range(i+1, m):
15            [r,c] = np.where(M2 == M1[i,j])
16            for k in range(len(r)):
17                VT[(i)*n + r[k]] = 1;
18                VT[(i)*n + c[k]] = 1;
19                VT[(j)*n + r[k]] = 1;
20                VT[(j)*n + c[k]] = 1;
21
22                if M is None:
23                    M = np.copy(VT)
24                else:
25                    M = np.concatenate((M, VT), 1)
26
27                VT = np.zeros((n*m,1), int)
28
29    return M
```

Listing A.2: source/dir/scala-example.scala

```
1 package examples
2
```

```
3 object Persons {
4
5   /** A list of persons. To create a list, we use Predef.
     List
6    *  which takes a variable number of arguments and
     constructs
7    *  a list out of them.
8    */
9   val persons = List(
10    new Person("Bob", 17),
11    new Person("John", 40),
12    new Person("Richard", 68)
13  )
14
15  /** A Person class. 'val' constructor parameters become
16   *  public members of the class.
17   */
18  class Person(val name: String, val age: Int)
19
20  /** Return an iterator over persons that are older than
      20.
21   */
22  def olderThan20(xs: Seq[Person]): Iterator[String] =
23    olderThan20(xs.elements)
24
25  /** Return an iterator over persons older than 20,
     given
26   *  an iterator over persons.
27   */
28  def olderThan20(xs: Iterator[Person]): Iterator[String]
      = {
29
30    // The first expression is called a 'generator' and
     makes
31    // 'p' take values from 'xs'. The second expression
     is
32    // called a 'filter' and it is a boolean expression
     which
33    // selects only persons older than 20. There can be
     more than
34    // one generator and filter. The 'yield' expression
     is evaluated
35    // for each 'p' which satisfies the filters and used
     to assemble
36    // the resulting iterator
37    for (p <- xs if p.age > 20) yield p.name
38  }
39 }
40
41
```

```
42 /** Some functions over lists of numbers which
       demonstrate
43  *  the use of for comprehensions.
44  */
45 object Numeric {
46
47   /** Return the divisors of n. */
48   def divisors(n: Int): List[Int] =
49     for (i <- List.range(1, n+1) if n % i == 0) yield i
50
51   /** Is 'n' a prime number? */
52   def isPrime(n: Int) = divisors(n).length == 2
53
54   /** Return pairs of numbers whose sum is prime. */
55   def findNums(n: Int): Iterable[(Int, Int)] = {
56
57     // a for comprehension using two generators
58     for (i <- 1 until n;
59          j <- 1 until (i-1);
60          if isPrime(i + j)) yield (i, j)
61   }
62
63   /** Return the sum of the elements of 'xs'. */
64   def sum(xs: List[Double]): Double =
65     xs.foldLeft(0.0) { (x, y) => x + y }
66
67   /** Return the sum of pairwise product of the two lists
       . */
68   def scalProd(xs: List[Double], ys: List[Double]) =
69     sum(for((x, y) <- xs zip ys) yield x * y);
70
71   /** Remove duplicate elements in 'xs'. */
72   def removeDuplicates[A](xs: List[A]): List[A] =
73     if (xs.isEmpty)
74       xs
75     else
76       xs.head :: removeDuplicates(for (x <- xs.tail if x
      != xs.head) yield x)
77 }
78
79
80 /** The main class, the entry point of this program.
81  */
82 object Fors {
83
84   def main(args: Array[String]) {
85     // import all members of object 'persons' in the
      current scope
86     import Persons._
87
```

```
88      print("Persons over 20:")
89      olderThan20(persons) foreach { x => print(" " + x) }
90      println
91
92      import Numeric._
93
94      println("divisors(34) = " + divisors(34))
95
96      print("findNums(15) =")
97      findNums(15) foreach { x => print(" " + x) }
98      println
99
100     val xs = List(3.5, 5.0, 4.5)
101     println("average(" + xs + ") = " + sum(xs) / xs.
        length)
102
103     val ys = List(2.0, 1.0, 3.0)
104     println("scalProd(" + xs + ", " + ys +") = " +
        scalProd(xs, ys))
105   }
106
107 }
```