Batch-Verarbeitung mit ActiveRecord

Andreas Gungl

OTTO Software Partner GmbH

OTTO Software Partner

- Mainframe (ASM / Cobol)
- C / C++
- Java SE und Java EE
- Seit 2005 Ruby und Rails

- Call Center Software
- Logistik-Steuerung in Lagern (bis 700.000 Sendungen/Tag)
- Fakturiersysteme (tgl. 1,5 Mio. Auftragspos. zu bewerten)
- Komplettsystem für Versandhandel (AMOS)

AMOS - Versandhandelssystem

- Module für:
  - Einkauf
  - Auftragsverwaltung (mit Erfassung im Call Center und Webshop-Online-Anbindung)
  - Fakturierung
  - Debitorenmanagement
  - Lagerverwaltung
- Bis 20.000 Aufträge pro Tag, ca. 2,5 Mio. Kundendaten
- Entstanden 1996-98 auf Basis von Linux (Oracle-DB, C/C++, Tcl/Tk, seit 2006 Rails sowie RubyQt)

Warum Rails?

- Einfachheit und Komfort von Rails (Convention over Configuration, Test First)
- Faszinierende Möglichkeiten von ActiveRecord
- Zeitsparende Implementierung von DB-Zugriffen und Webservices
- Migrationspfad im Call Center bedingte Einsatz von RubyQt
- Synergien bei den Modelklassen (Nutzung im Call Center, in Online-Schnittstellen und in Batches)

Batches in Ruby

- Beispiele für Batches:
  - Bestandszuteilung
  - Mahnwesen
  - Vorbereitung der Lagerentnahmen (Picking)
- Alternative Implementierungen:
  - PL/SQL
  - C/C++ mit OCI
- Beide Alternativen generieren erheblichen Mehraufwand!

ActiveRecord und Batches

- Hauptproblem:
  - Extremer Speicherverbrauch bei `ActiveRecord#find :all` mit tausenden Objekten
- Existierende Lösungen für Rails
  - Iterate over the records in groups
    http://snippets.dzone.com/posts/show/5461
  - Faking cursors in ActiveRecord
    http://weblog.jamisbuck.org/2007/4/6/faking-cursors-in-activerecord
- Nachteile:
  - Transaktionen werden u.U. gesplittet
  - Keine Read Consistency
- Suche nach einer Lösung, die die ResultSet-Iteration in ActiveRecord erlaubt

ActiveRecord-Iterator

- Schnittstelle:

```
MyTable.find_each(:conditions => …) do |record|
  # do something with record
end

MyTable.find_each_by_sql(...) do |record|
  # do something with record
end
```

- ActiveRecord::Base soll erweitert werden. Die Methodennamen deuten auf die Iteration hin.

ActiveRecord-Iterator

- Anpassung in ActiveRecord::Base

```
module ActiveRecord
  class Base
    class << self
    def find_each_by_sql(sql, &block)
      connection.select_each(sanitize_sql(sql),
          „#{name} Load“) do |record|
        yield instantiate(record)
      end
    end
  end
end
```

- Problem: `DatabaseStatements#select_all` funktioniert nicht wie nötig!

ActiveRecord-Iterator

- Anpassung in DatabaseStatements um den Block weiterzureichen:

```
module ConnectionAdapters
  module DatabaseStatements
    def select_each(sql, name = nil, &block)
      select_each_row(sql,
        name = nil) { |row_hash| yield row_hash }
    end
  end
end
```

- Anpassung der Datenbank-Adapter für die Verarbeitung des Blocks ist nötig!

Anpassung am MySQL-Adapter

- Vorbild:

```
class MySQLAdapter < AbstractAdapter
  def select_rows(sql, name = nil)
    @connection.query_with_result = true
    result = execute(sql, name)
    rows = []
    result.each_hash { |row| rows << row }
    result.free
    rows
  end
end
```

Anpassung am MySQL-Adapter

- Neue Iterator-Methode:

```
class MySQLAdapter < AbstractAdapter
  def select_each_row(sql, name = nil)
    @connection.query_with_result = true
    result = execute(sql, name)
    result.each_hash { |row| yield.row }
    result.free
    nil
  end
end
```

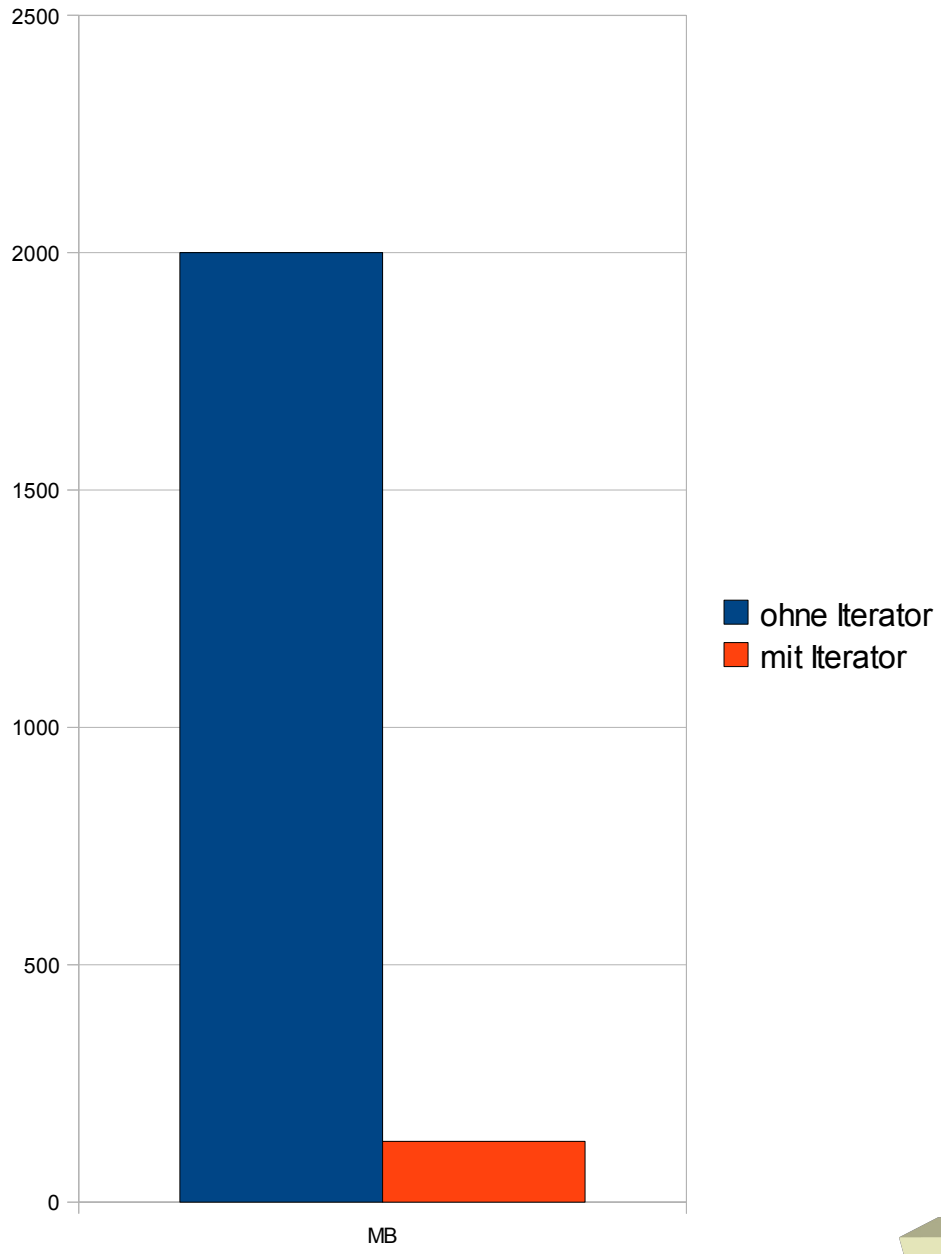
- Der Aufruf aus `DatabaseStatements#select_each_row` erfolgt mit Block, der nun für jede Zeile im ResultSet ausgeführt wird.

ActiveRecord-Iterator

- Unterstützte DB-Adapter:
  - Oracle
  - MySQL
  - JDBC (bis 0.9 und für 0.9.1)
- Für den JDBC-Adapter sind Patches nötig, um die in Java liegende Implementierung anzupassen. Ab Version 0.9.1 des Adpaters hat dieser eine veränderte innere Struktur. Daher war der alte Patch nicht mehr verwendbar.
- Die Anpassung weiterer Adapter ist möglich...

Speicherverbrauch

- Verarbeiten von 80.000 Kundendaten
- mit JRuby 1.1.4
- 2 GB ohne Iterator (mit massiver Garbage Collection),
  128 MB mit Iterator



Schwachstellen

- Nicht alle Datenbanken sind unterstützt.
- Problem beim iterativen Verarbeiten von Master- mit Detailtabellen im JOIN
  - :include Option lädt nicht wie eventuell erwartet automatisch alle Details des Masters
- Ausgehend von Detailtabellen mit dem Master im JOIN besteht das Problem nicht.

Ideal wäre sicherlich die Aufnahme der Iterator-Funktionalität in das ActiveRecord-Framework mit dann gesicherter Unterstützung für alle Datenbanken.

Fragen?

Danke für Ihr Interesse!