Text Extraction from Big Data

Дмитрий Брюхов, к.т.н., с.н.с.

Институт Проблем Информатики РАН

Content

- Text extraction in big data
- Span Algebra
- Annotation Query Language
- The Eclipse IDE for text analytics
- Example of creating text analytics project in Eclipse

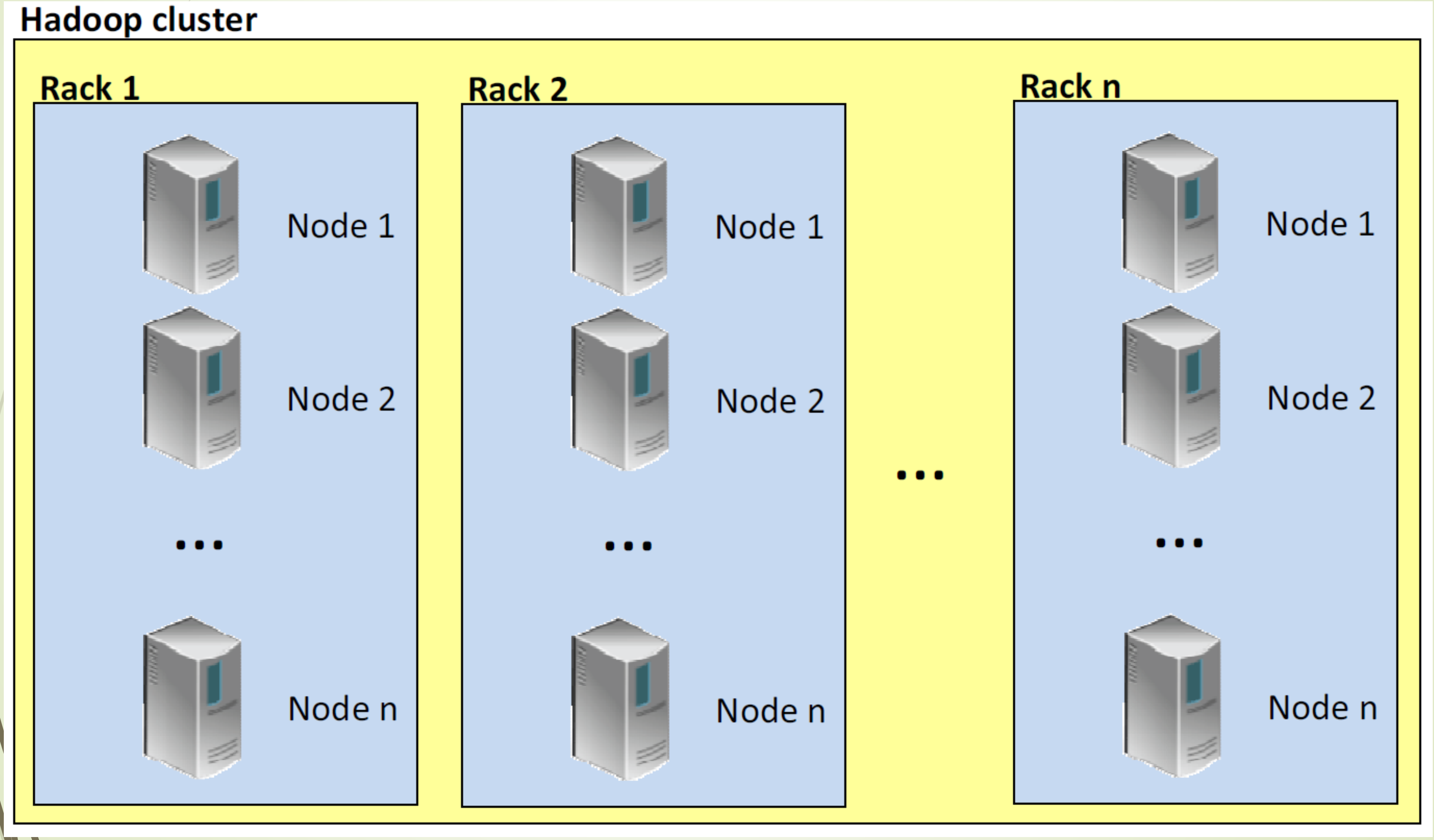
What is big data?

- Massive volumes of data stored across many machines
- Too big to back up
- Could be structured, semi-structured or unstructured data
- Using traditional OLTP or data warehouse solutions for data analysis are inadequate

What is Hadoop?

- Open source project
- Written in Java
- Optimized to handle
  - Massive amounts of data through parallelism
  - A variety of data (structured, unstructured, semi-structured)
  - Using inexpensive commodity hardware
- Great performance
- Reliability provided through replication
- Not for OLTP, not for OLAP/DSS, good for Big Data

Terminology review



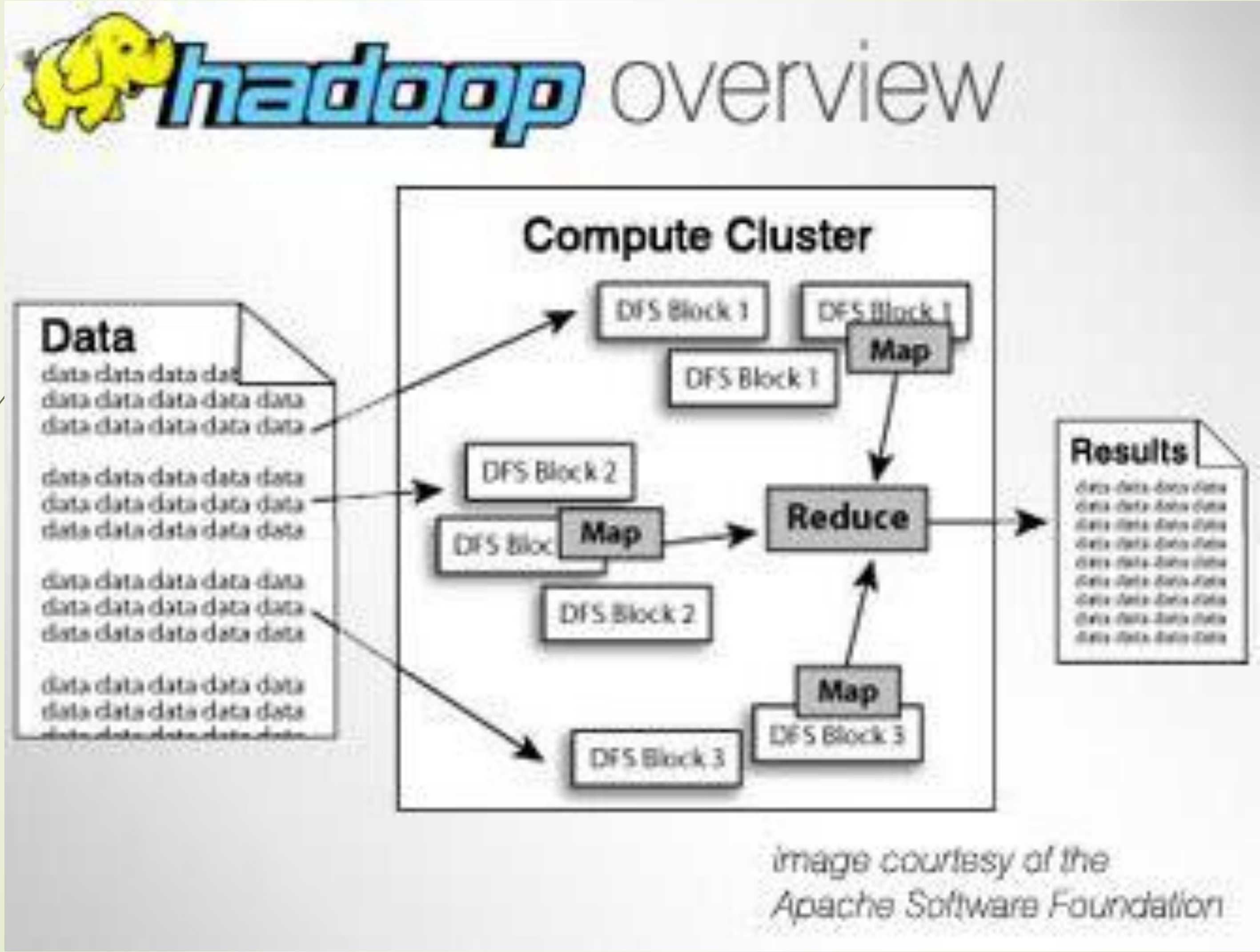
Hadoop architecture

- Two main components:
  - Hadoop Distributed File System (HDFS)
  - MapReduce Engine

Hadoop distributed file system (HDFS)

- Hadoop file system that runs on top of existing file system
- Designed to handle very large files with streaming data access patterns
- Uses blocks to store a file or parts of a file

8 Map Reduce


hadoop overview
Compute Cluster
Data
DFS Block 1
DFS Block 1
DFS Block 1
Map
DFS Block 2
DFS Block 2
Map
Reduce
DFS Block 3
DFS Block 3
Map
Results
image courtesy of the
Apache Software Foundation

Know where to look for the data

- Warehouse data about purchase history of a customer
- Site browsing history of the customer
- Comments on Websites
- Customer surveys

Integrate from various sources

- Different Data Source Platforms
  - Different hardware platforms
  - Different database systems
- Different Data Formats
  - Structured Data Sources
  - Semi Structured Data Sources
  - Unstructured Data Sources

Problem with unstructured data

- Structured data has
  - Known attribute types
    - Integer
    - Character
    - Decimal
  - Known usage
    - Represents salary versus zip code
- Unstructured data has no
  - Known attribute types nor usage
- Usage is based upon context
  - Tom Brown has brown eyes
- A computer program has to be able view a word in context to know its meaning

Need to harvest unstructured data

- Most data is unstructured
- Most data used for personal communication is unstructured
  - Email
  - Instant messages
  - Tweets
  - Blogs
  - Forums
- Opinions are expressed when people communicate
  - Beneficial for marketing
  - Give insight between you and your competitors

Need for structured data

- Business intelligence tools work with structured data
  - OLAP
  - Data mining
- To use unstructured data with business intelligence tools
  - Requires that structured data to be extracted from unstructured and semi-structured data
- InfoSphere BigInsights provides a language, Annotation Query Language (AQL)
  - Syntax is similar to that of Structured Query Language (SQL)
  - Builds extractors to extract structured data from
    - Unstructured data
    - Semi-structured data

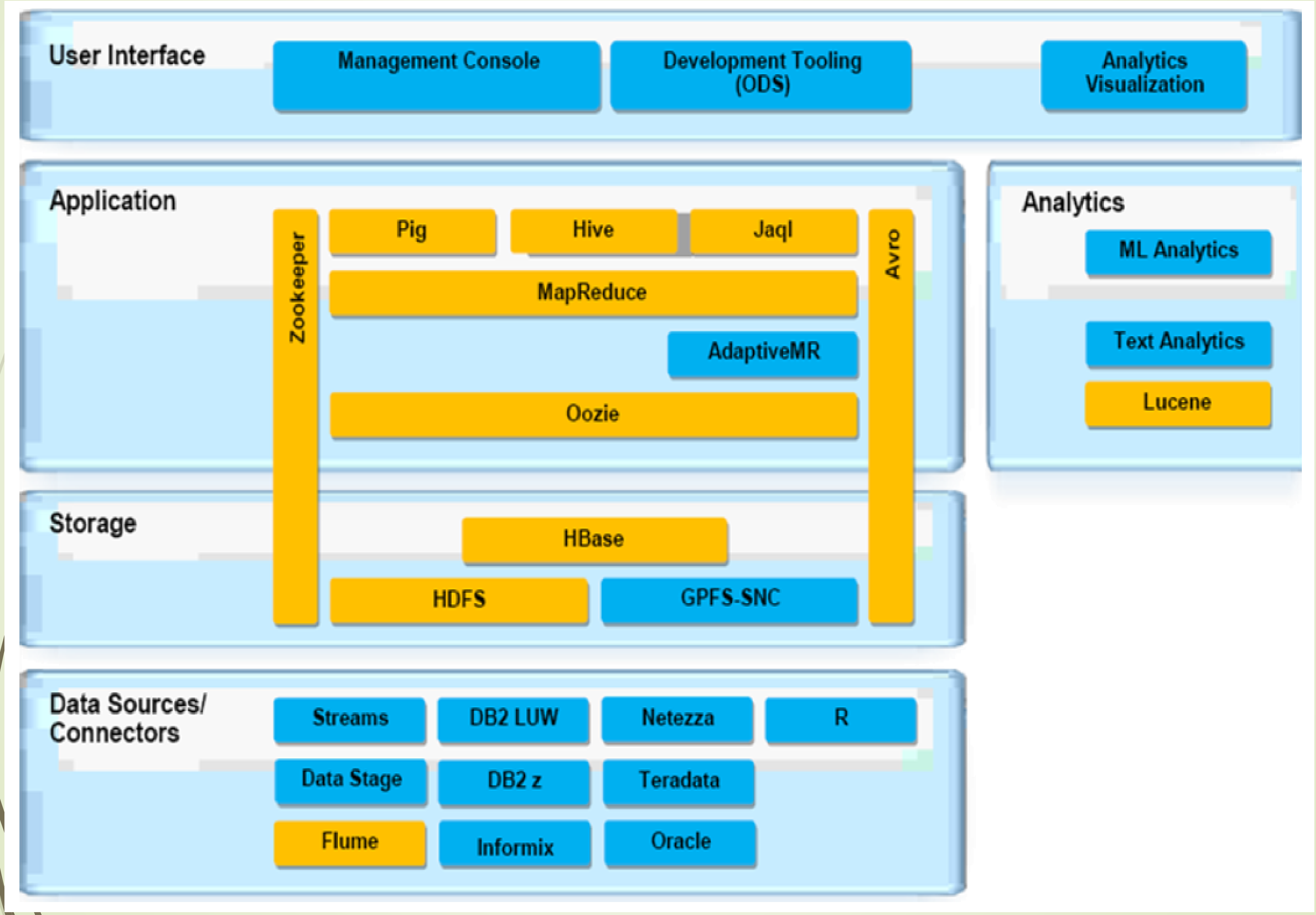
Content

- Text extraction in big data
- **Text Analytics in BigInsights**
  - Span Algebra
  - Annotation Query Language
- The Eclipse IDE for text analytics
- Example of creating text analytics project in Eclipse

InfoSphere BigInsights

- InfoSphere BigInsights – это программная платформа, призванная помочь организациям находить и анализировать бизнес-информацию, скрытую в больших объемах разнообразных данных, которые часто игнорируются или отвергаются, поскольку слишком неудобны или трудны для обработки традиционными средствами.
- Примерами таких данных являются данные социальных медиа, новостные ленты, журналы событий, данные о поведении потребителей (clickstream-данные), выходные данные электронных датчиков и даже некоторые традиционные транзакционные данные.

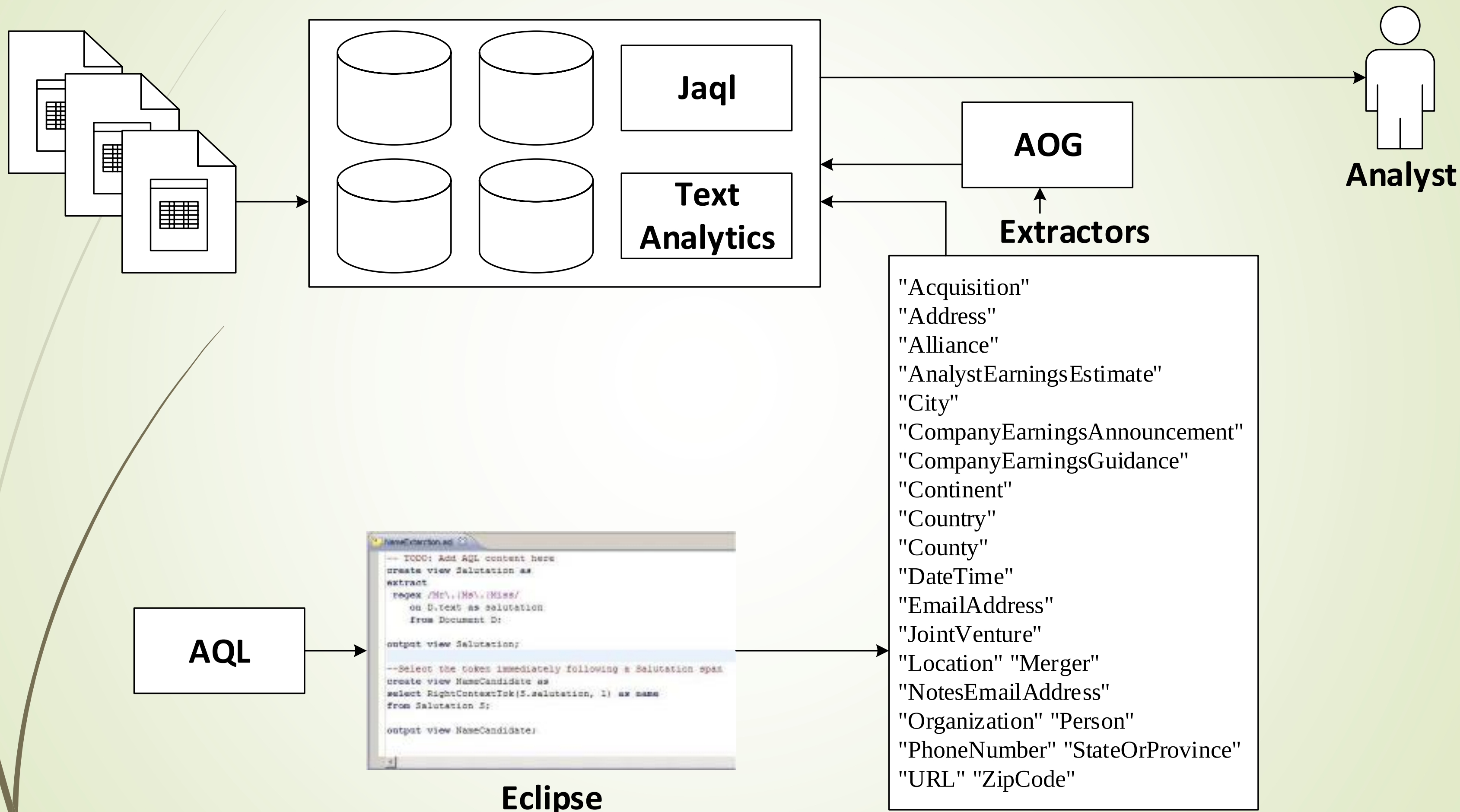
InfoSphereBigInsights–A Full HadoopStack (IBM)



Key aspects of text analytics in BigInsights

- A declarative language for identifying and extracting content from text data— The Annotation Query Language (AQL) enables programmers to create views (collections of records) that match specified rules.
- User-created or domain-specific dictionaries— Dictionaries can identify relevant context across input text to extract business insight from documents.
- User-created rules for text extraction— Pattern discovery and regular expression (regex) building tools enable programmers to specify how text should be analyzed to isolate data of interest.
- Provenance tracking and visualization— Text analysis is often iterative by nature, requiring rules (and dictionaries) to be built upon one another and refined over time.

Text Analytics Overview



Content

- Text extraction in big data
- Text Analytics in BigInsights
  - **Span Algebra**
  - Annotation Query Language
- The Eclipse IDE for text analytics
- Example of creating text analytics project in Eclipse

Data and Execution Model

- Algebra operates over a simple relational data model with three data types: span, tuple, and relation.
- A **span** is an ordered pair <*begin*, *end*> that denotes the region of text
- A **tuple** is a finite sequence of *w* spans $<s_1, ..., s_w>$;
  - we call *w* the width of the tuple.
- A **relation** is a multiset of tuples with the constraint that every tuple must be of the same width.
- Each operator in our algebra takes zero or more relations as input and produces a single relation as output.

Spans vs Strings

Moscow University from Moscow

- P<1,7> => “Moscow”
- P<24,30> => “Moscow”

- P<1,7> ≠ P<24,30>

Algebra Operators

- Relational Operators
- Span Extraction Operators
- Span Aggregation Operators

Relational Operators

- Since our data model is a minimal extension to the relational model, all of the standard relational operators apply without any change.
  - Select
  - Project
  - Join
  - …
- The main addition is that we use a few new selection predicates applicable only to spans
  - FollowsSpan

Span Extraction Operators

- A span extraction operator identifies segments of text that match a particular input pattern and produces spans corresponding to each such text segment.
- Regular expression matcher (E_{re})
  - Given a regular expression *r*, $E_{re}(r)$ identifies all non-overlapping matches when *r* is evaluated from left to right over the text represented by *s*. The output of $E_{re}(r)$ is the set of spans corresponding to these matches.
- Dictionary matcher (E_d)
  - Given a dictionary *dict* consisting of a set of words/phrases, the dictionary matcher $E_d(dict)$ produces an output span for each occurrence of some entry in *dict* within *doctext*.

Span Aggregation Operators

- Span aggregation operators take in a set of input spans and produce a set of output spans by performing certain aggregate operations over their entire input.
- Block
  - The block operator is used to identify regions of text where input spans occur with enough regularity.
- Consolidate
  - The block operator is used to identify regions of text where input spans occur with enough regularity.
- Containment consolidation
  - discard annotation spans that are wholly contained within other annotation spans.
- Overlap consolidation
  - produce new spans by merging overlapping spans

Span Aggregation Example

```
–– Define a dictionary of instrument names
create dictionary Instrument as ( ' flute ' , ' guitar ' , ... );
–– Use a regular expression to find names of band members
create view BandMember as
extract regex /[A−Z]\w+(\s+[A−Z]\w+)/
   on 1 to 3 tokens of D.text
   as name
from Document D;
–– A single ReviewInstance rule . Finds instances of BandMember followed
within 30 characters by an instrument name.
create view ReviewInstance as
select CombineSpans(B.name, I.inst) as instance
from
   BandMember B,
   (extract dictionary ' Instrument' on D.text as inst from Document D) I
where
   Follows(B.name, I . inst , 0, 30)
consolidate on CombineSpans(B.name, I.inst);
```

Content

- Text extraction in big data
- Text Analytics in BigInsights
  - Span Algebra
  - **Annotation Query Language**
- The Eclipse IDE for text analytics
- Example of creating text analytics project in Eclipse

Annotation Query Language (AQL) overview

- Annotation Query Language (AQL) is the language for developing text analytics extractors in the InfoSphere® BigInsights™ Text Analytics system. An extractor is a program written in AQL that extracts structured information from unstructured or semistructured text.
- AQL is a declarative language, with a syntax that is similar to that of the Structured Query Language (SQL).

AQL Data model

- Similar to the standard relational model
- Data is stored in tuples
  - Data records of one or more columns or fields
- A collection of tuples forms a relation
  - All tuples in a relation must have the same schema
    - That is names and field types must be the same
  - Scalar types
    - Integer – 32-bit signed integer
    - Float – Single precision floating-point number
    - Text – Unicode string
    - Span – Contiguous region of characters in a text object
    - List – Represents a bag of values of type (Integer, Float, Text, or Span)

AQL Execution model

- An AQL extractor consists of a collection of views
  - Each view defines a relation
- Special view called Document
  - Represents the document that is being annotated
    - Two fields in this view
      - text –Textual content of the document
      - label –Label for the document, usually the name of the document
- Support for include statements
  - Allows for reused of code

AQL components overview

- Create view statement
  - Creates a view name and defines the tuples inside the view
- Extract statement
  - Provides functionality for extracting features directly from text
- Select statement
  - Mechanism for constructing complex patterns out of simpler building blocks
- Detag statement
  - Pre-processing step to remove all HTML and XML tags from a document

AQL components overview (cont.)

- Create dictionary
  - Define dictionaries of words or phrases to use in extract statements
- Create table
  - Similar to the SQL create table statement
- Built-in functions
  - Functions for use in extraction rules
- User-defined functions
  - Allows user to define customer functions to be used in extraction rules

Create View Statement

- Creates a view and defines the tuples inside that view.

```
create view <viewname> as <select or extract statement>;
```

- The output view statement defines a view to be an output view.

```
output view <viewname>;
```

- Example

```
create view PhoneNumbers as
    extract regex /(\d{3})(-)(\d{3})(-)(\d{4})/ on d.text return
        group 1 as areacode and
        group 3 as exchange and
        group 5 as extension
    from Document d;
```

Regular expressions

- Regular expressions are used to extract textual values from a non-structured file
- AQL uses Perl syntax for regular expressions . The format is:

```
regex[es] /<regex1>/
        [and /<regex2>/ and ... and /<regex n>/]
        [with flags '<flags string>']
    on [<token spec>] <name>.<column>
    <grouping spec>
```

- Grouping specification determines how to handle capturing groups in the regular expression. Capturing groups are regions of the regular expression match, identified by parentheses in the original expression.
- Example

```
extract regex /(\d{3})(-)(\d{3})(-)(\d{4})/
    on d.text return
        group 1 as areacode and
        group 3 as exchange and
        group 5 as extension
```

Dictionaries

- Use the dictionary extraction specification to extract strings from input text that are contained in a dictionary of strings.
- Dictionaries are created either using the create dictionary stateme
- Dictionary extraction specification

```
dictionar[y|ies]
       '<dictionary>'
       [and '<dictionary>' and ... and '<dictionary>']
       [with flags '<flags string>']
```

- Example

```
create dictionary ConjunctionDict as
    ('and', 'or', 'but', 'yet');

create view Conjunction as
extract
    dictionary 'ConjunctionDict' on D.text as name
from Document D;
```

Splits

- Use the split extraction specification to split a large span into several smaller spans.
- The split extraction specification takes two arguments:
  - A column that contains longer target spans of text
  - A column that contains split points
- Example

```
create view Sentences as
extract
    split using B.boundary
        retain right split point
        on B.text
        as sentence
from (
    extract
        D.text as text,
        regex /(([\.\?!]+\s)|(\n\s*\n))/ on D.text as boundary
        from Document D
) B;
```

Blocks

- Use the blocks extraction specification to identify blocks of contiguous spans across input text.
- The block extraction specification takes two parameters:
  - count - Specifies how many spans can make up a block
  - separation - Specifies how much distance is allowed between spans before they are no longer considered to be contiguous
- Example

```
create view BlockCapitalWords as
    extract blocks
        with count between 2 and 3
        and separation between 0 and 100 characters
        on cw.words as captialwords
    from CapitalizedWords cw;
```

Part of speech

- Use the Part-of-speech extraction specification to identify locations of different parts of speech across the input text.
- Syntax:

```
part_of_speech
 '<part of speech spec>'
 [and '<part of speech spec>']*
 [with language '<language code>']
 [and mapping from <mapping table name>]
 on <input column> as <output column>
 from <input view>
```

- part of speech spec
  - a comma-delimited list of part-of-speech tags
  - A combination of an internal part-of-speech name and flags, as defined by the mapping table
- Example

```
create view Nouns as
    extract part_of_speech 'NN','NNS'
    on D.text as noun
    from Document D;
```

Multilingual support

Language	Code	Tokenization	Part-of-speech analysis
Afrikaans	af	Y	N
Arabic	ar	Y	Y
Catalan	ca	Y	N
Chinese (Simplified)	zh-CN	Y	Y
Chinese (Traditional)	zh-TW	Y	Y
Czech	cs	Y	N
Danish	da	Y	Y
Dutch	nl	Y	Y
English	en	Y	Y
Finnish	fi	Y	N
French	fr	Y	Y
German	de	Y	Y
Greek	el	Y	N
Italian	it	Y	Y
Japanese	ja	Y	Y
Korean	ko	Y	N
Norwegian (Bokmal)	nb	Y	N
Norwegian (Nynorsk)	nn	Y	N
Polish	pl	Y	N
Portuguese	pt	Y	Y
Russian	ru	Y	N
Spanish	es	Y	Y
Swedish	sv	Y	N

Sequence patterns

- Use the pattern extraction specification to perform pattern matching across an input document in addition to spans previously extracted from the input document.
- Allows you to write complex extraction patterns in a compact fashion involving
  - Alternation
  - Sequences
  - Repetitions
- Example

```
create view CountryLocation as
    extract
        pattern (<C.country>) <Token>{5,35} (<L.longlat>)
        return group 1 as country
        and group 2 as location
    from Countries C, Locations L;
```

The select statement

- Allows for the construction of more complex patterns from simpler building blocks
- The basic for is very similarly structured to an SQL select statement
- Format

```
select <selectlist>
from <fromlist>
[where <predicate>]
[consolidate on <column>]
[group by <group list>]
[order by <order list>]
[limit <maximum number of tuples for each document>]
```

AQL built-in functions

- AQL has three types of built-in functions
  - Predicate
  - Scalar
  - Aggregate
- Predicate functions- Building blocks for the where clause
  - Contains, Equals, Follows
- Scalar functions - Return integers, strings or new spans
  - CombineSpans, GetBegin, GetEnd, GetText
- Aggregate functions - Create a single value from a list of input values
  - Avg, Count, Max

User-defined functions

- AQL allows user to define custom functions used in extraction rules
- Only scalar user-defined functions (UDFs) are supported
- Currently AQL supports UDFs implemented in Java. Implement the UDF as a public method in a Java class.
- You make the user-defined functions (UDFs) available to AQL by using the create function statement.
- Scalar user-defined functions (UDFs) can be used in the select and where clauses in the same way as built-in scalar functions.
- User-defined functions that return Boolean can be used as predicates in where and having clauses.

Pre-built extractor libraries

- The Standard and Multilingual tokenizer JAR files export the following views:

Person
Organization
Location
Address
City
Continent
Country
County
DateTime
EmailAddress
NotesEmailAddress
PhoneNumber

StateOrProvince
Town
URL
ZipCode
CompanyEarningsAnnouncement
AnalystEarningsEstimate
CompanyEarningsGuidance
Alliance
Acquisition
JointVenture
Merger

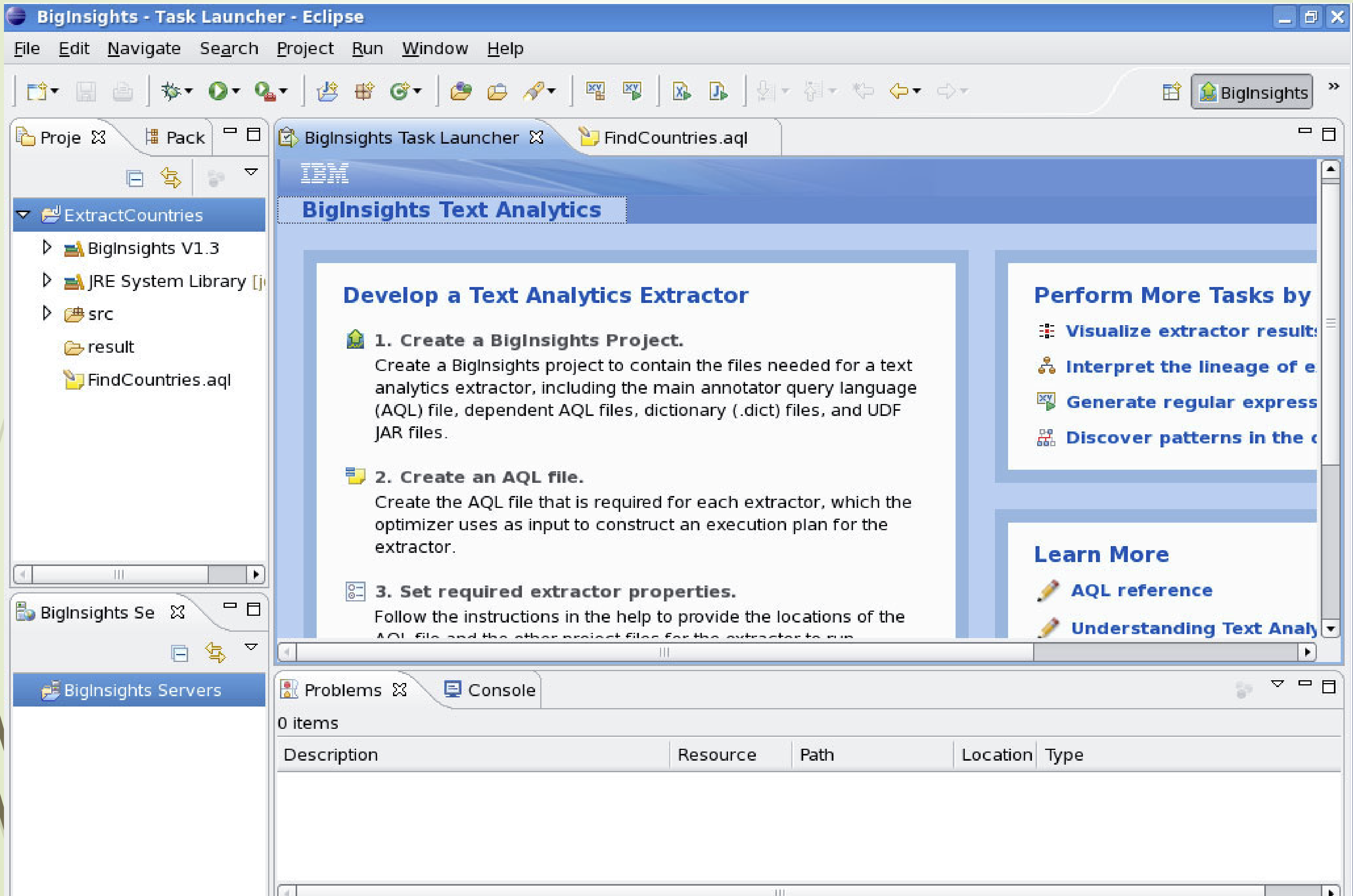
Content

- Text extraction in big data
- Text Analytics in BigInsights
  - Span Algebra
  - Annotation Query Language
- **The Eclipse IDE for text analytics**
- Example of creating text analytics project in Eclipse

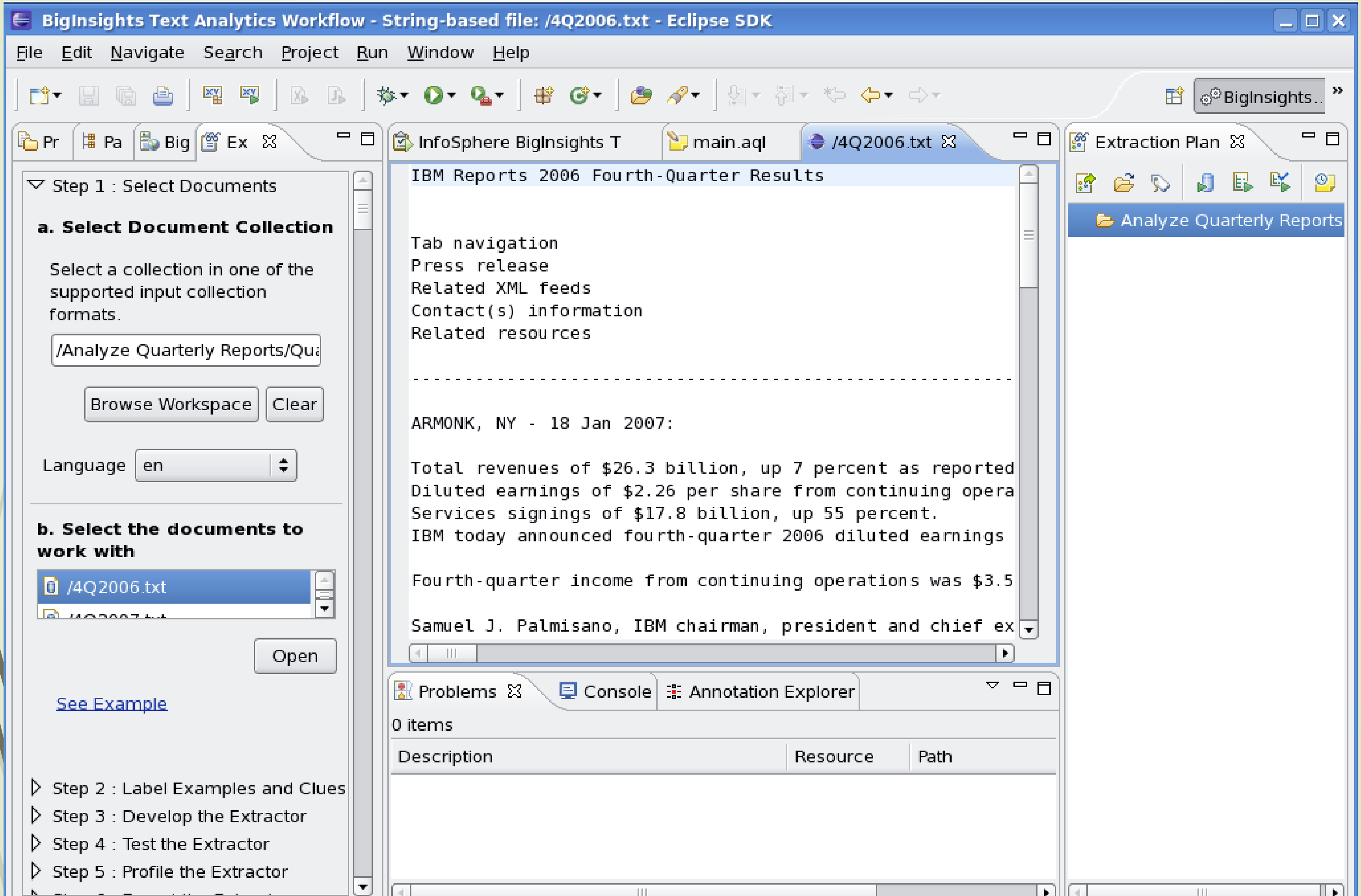
Developing text analytics extractors

- You can build and maintain extractors inside a full-fledged Eclipse development environment by installing the InfoSphere BigInsights tools for Eclipse.
- Inside the tools for Eclipse, there are two ways to develop an extractor:
  - Text Analytics Workflow perspective
    - This perspective displays several panes, one of which is the Extraction Tasks pane. This pane guides you through a six-step process to develop, test, and export extractors for deployment. You start by selecting your input document collection and conclude with exporting your working extractor to the cluster.
  - AQL Editor
    - This editor provides a framework to develop and test AQL extractors.

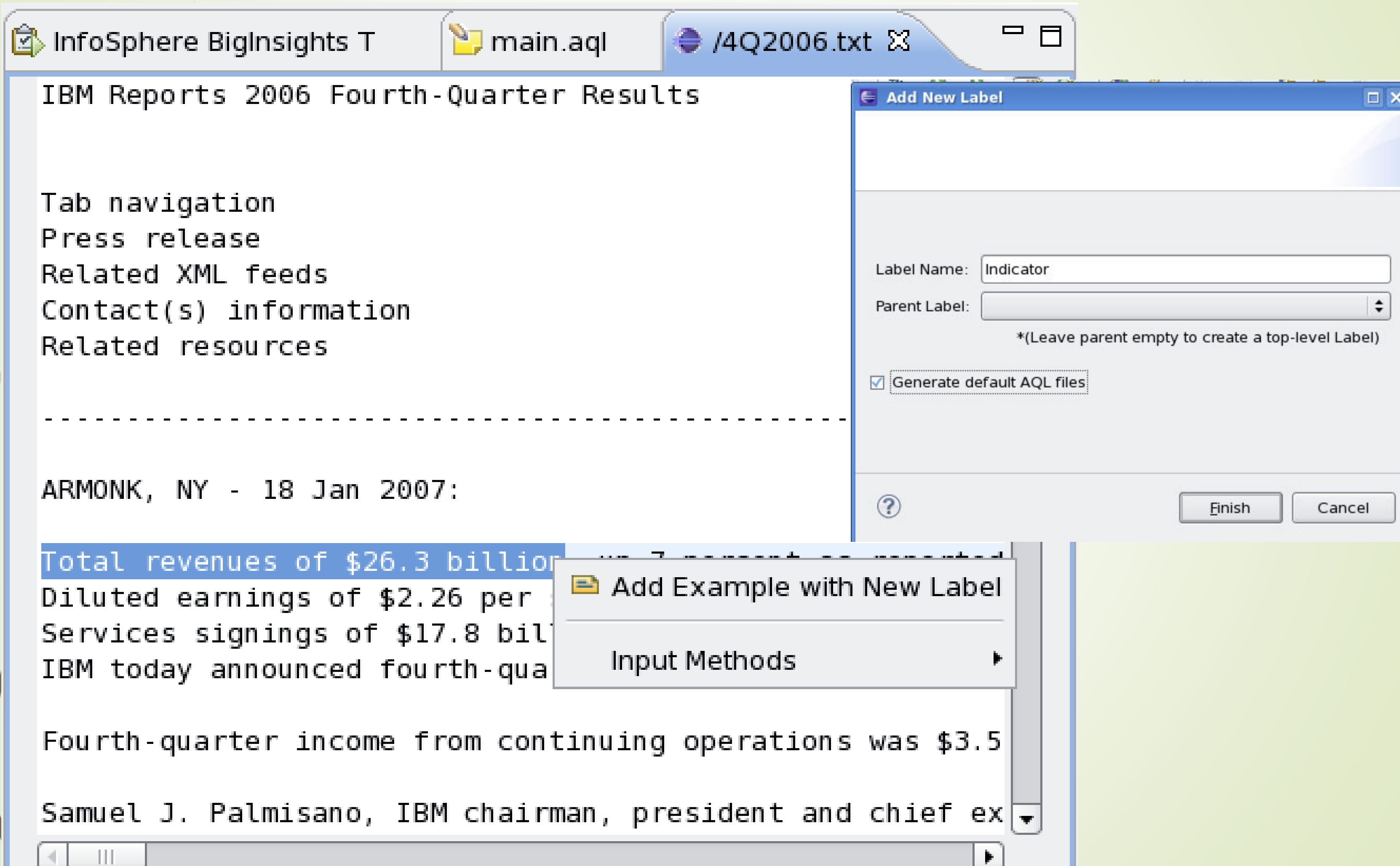
The Eclipse IDE for text analytics



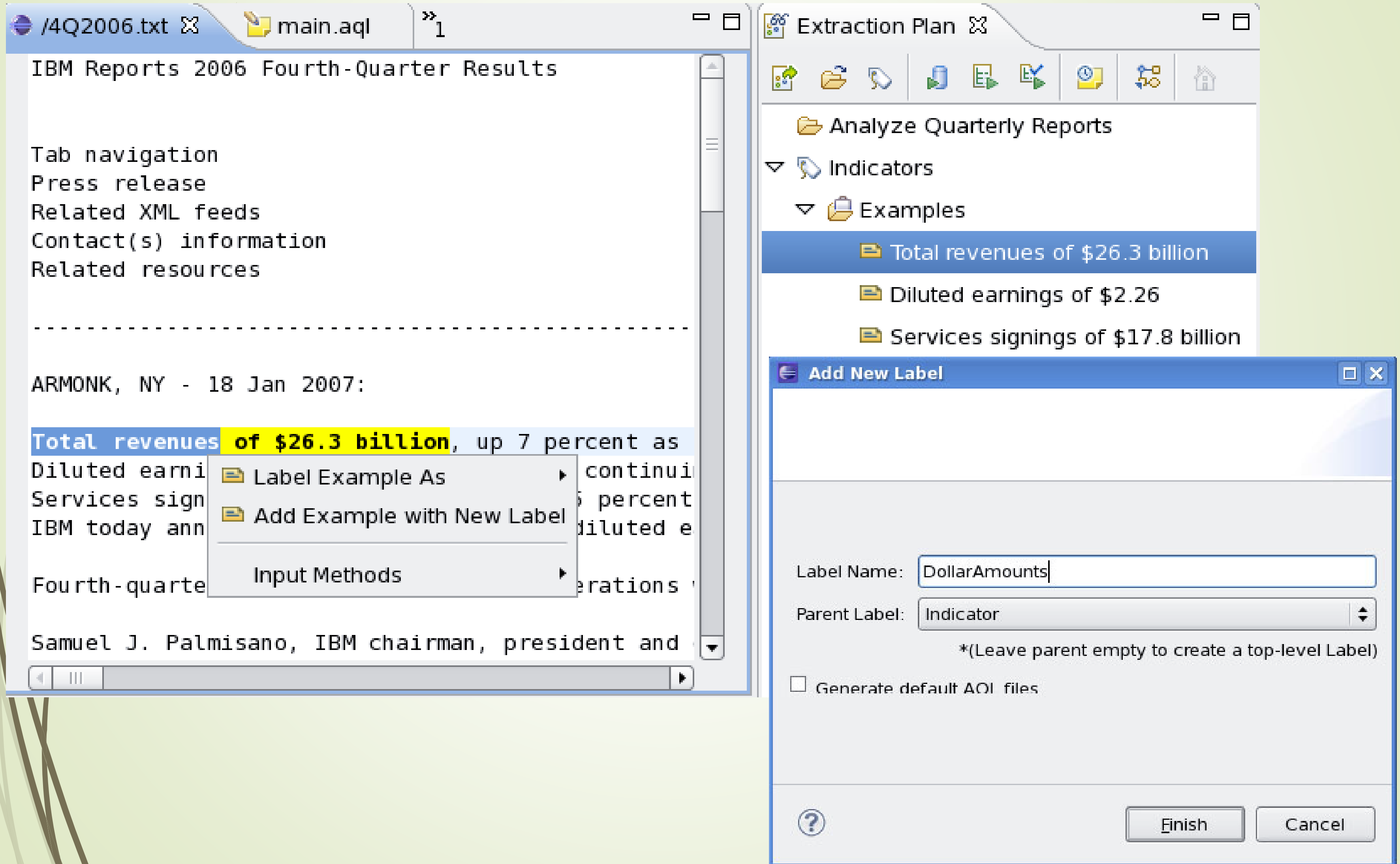
Eclipse extraction tasks and plan views



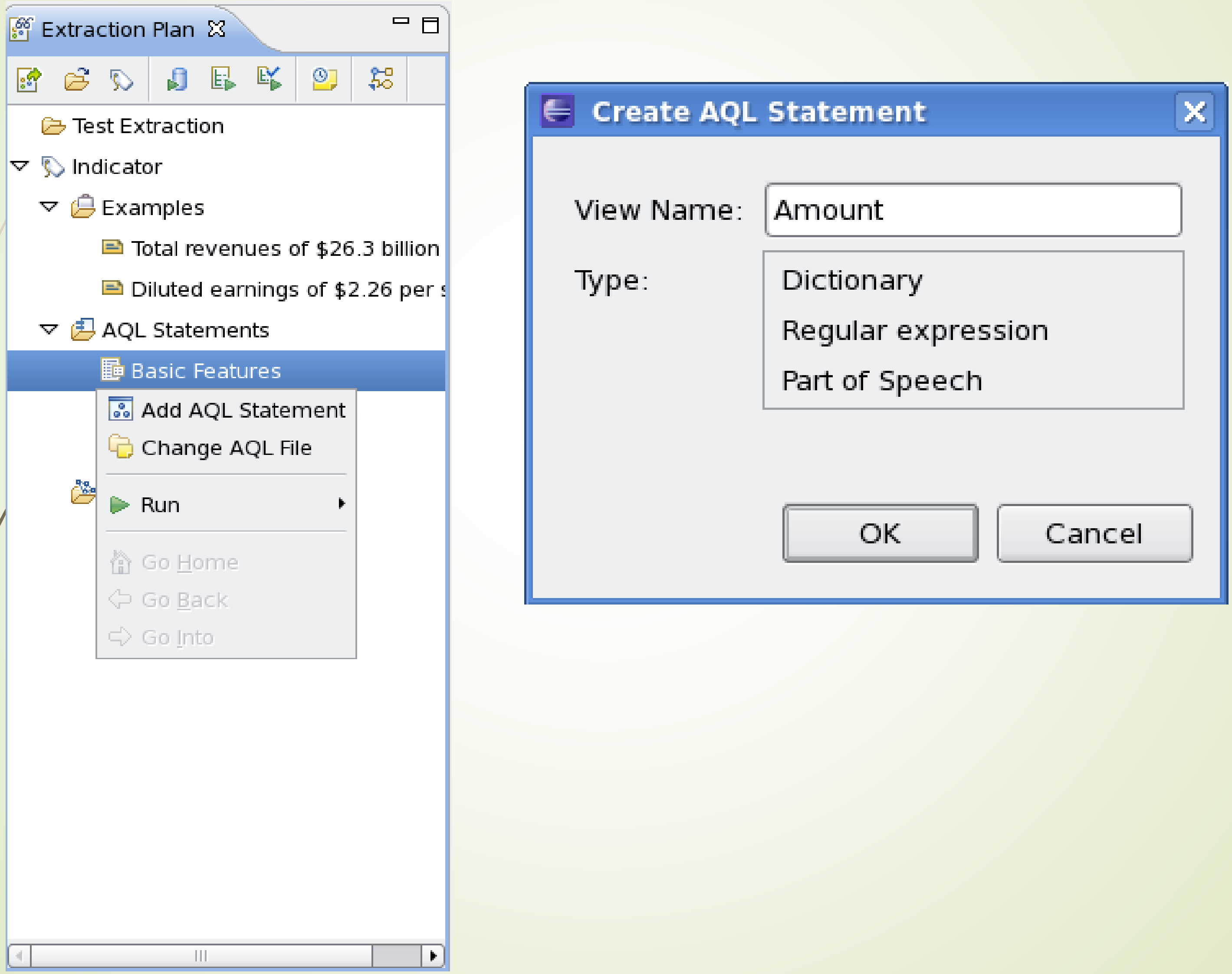
Label examples



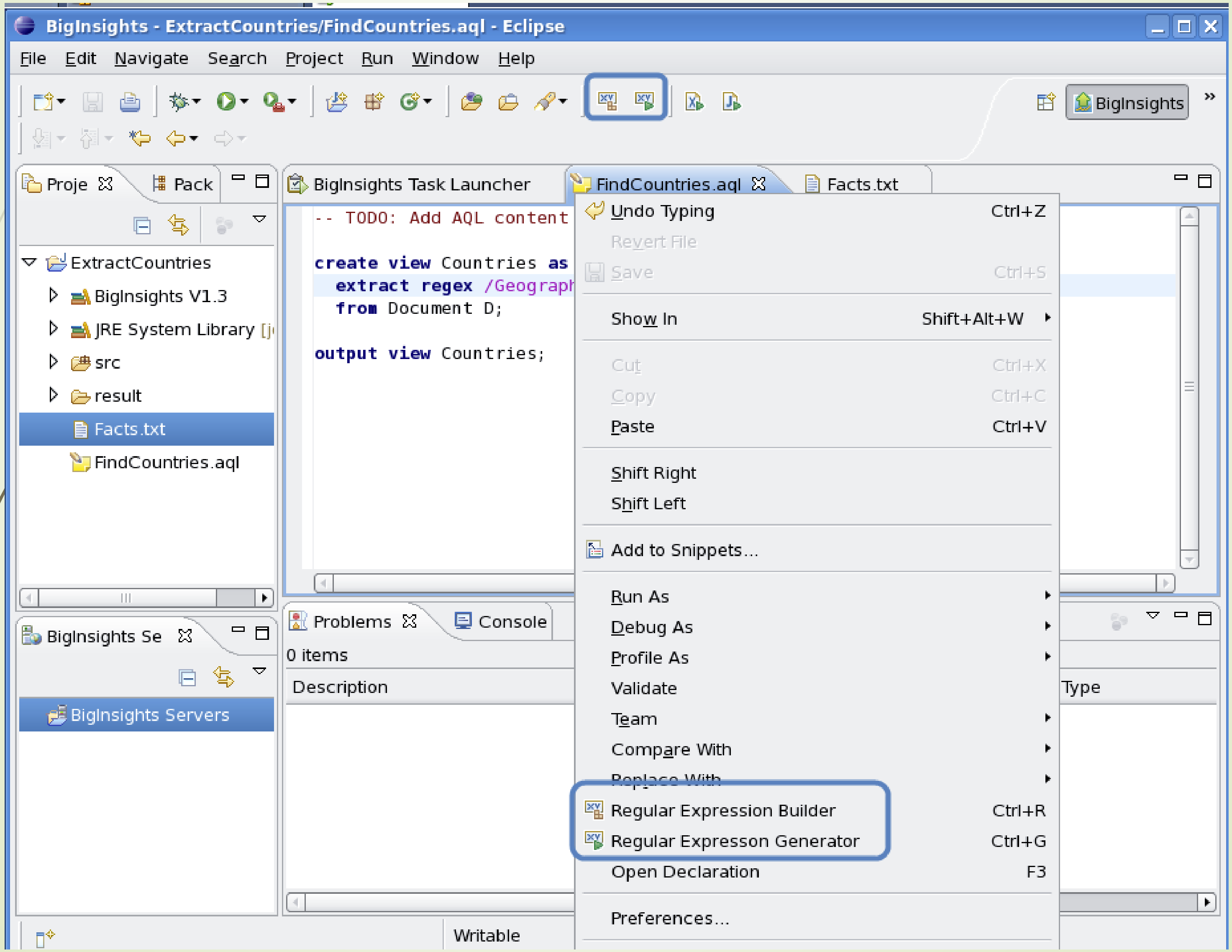
Adding clues to the puzzle



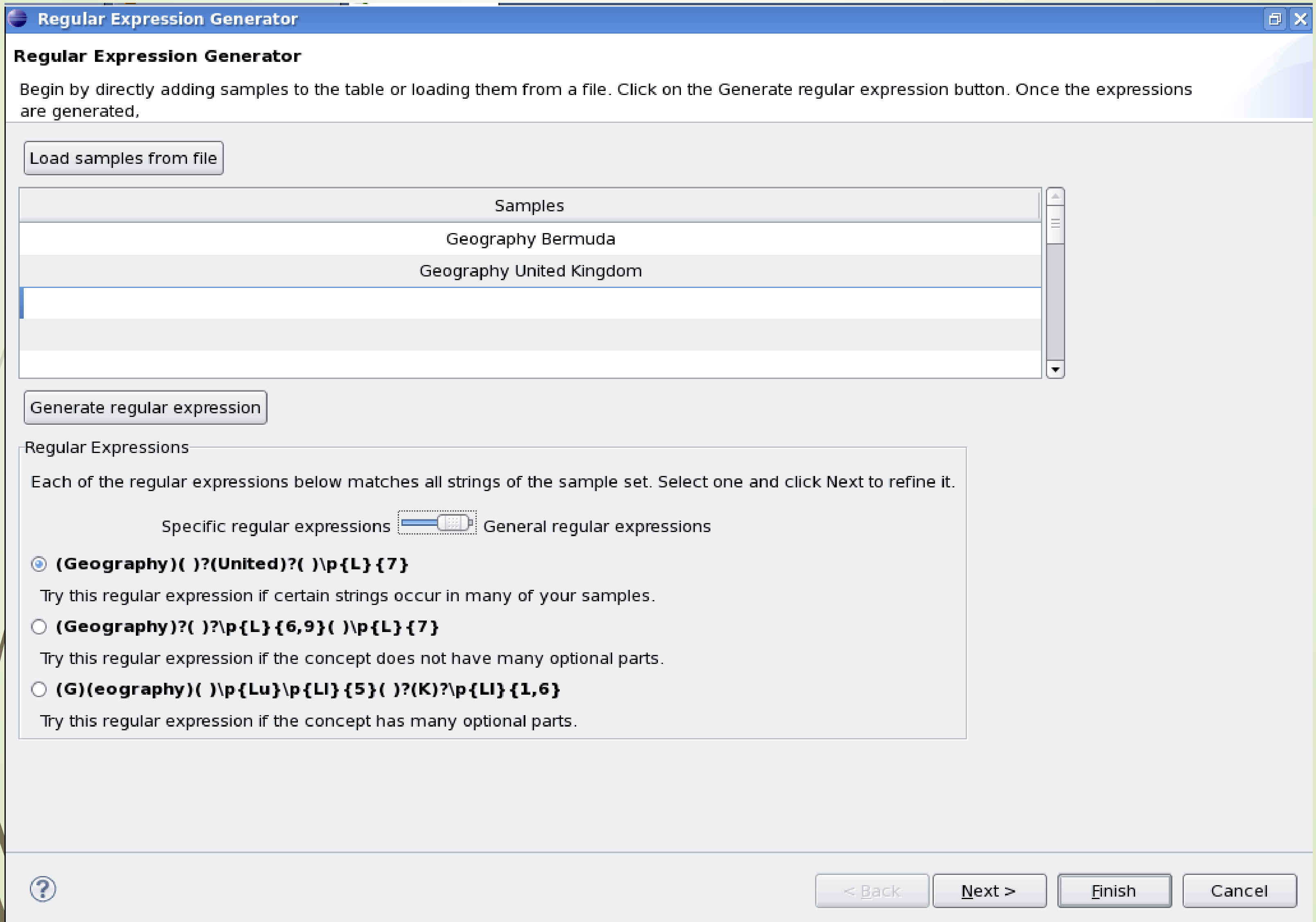
Create AQL statements



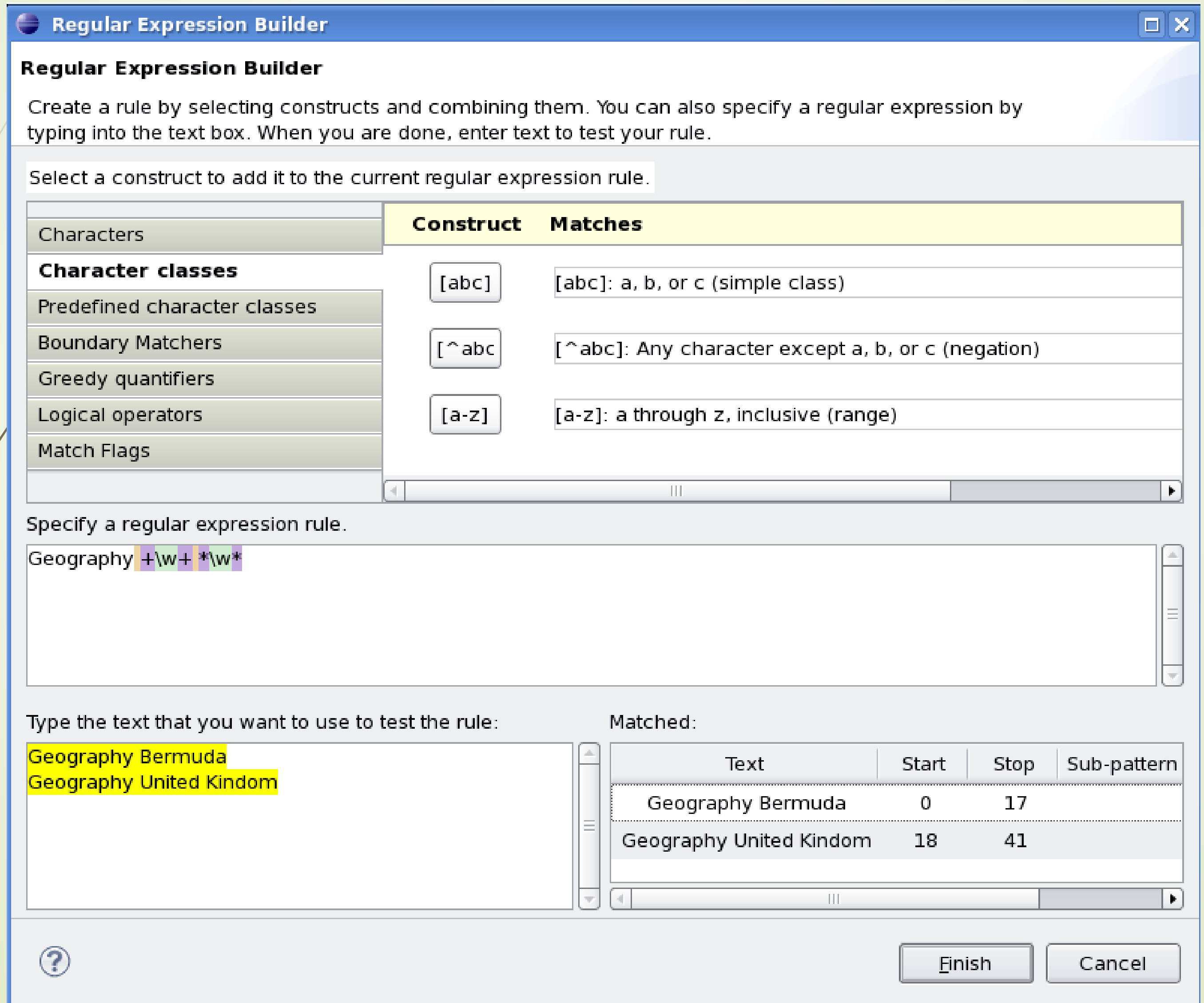
Regular expression helpers



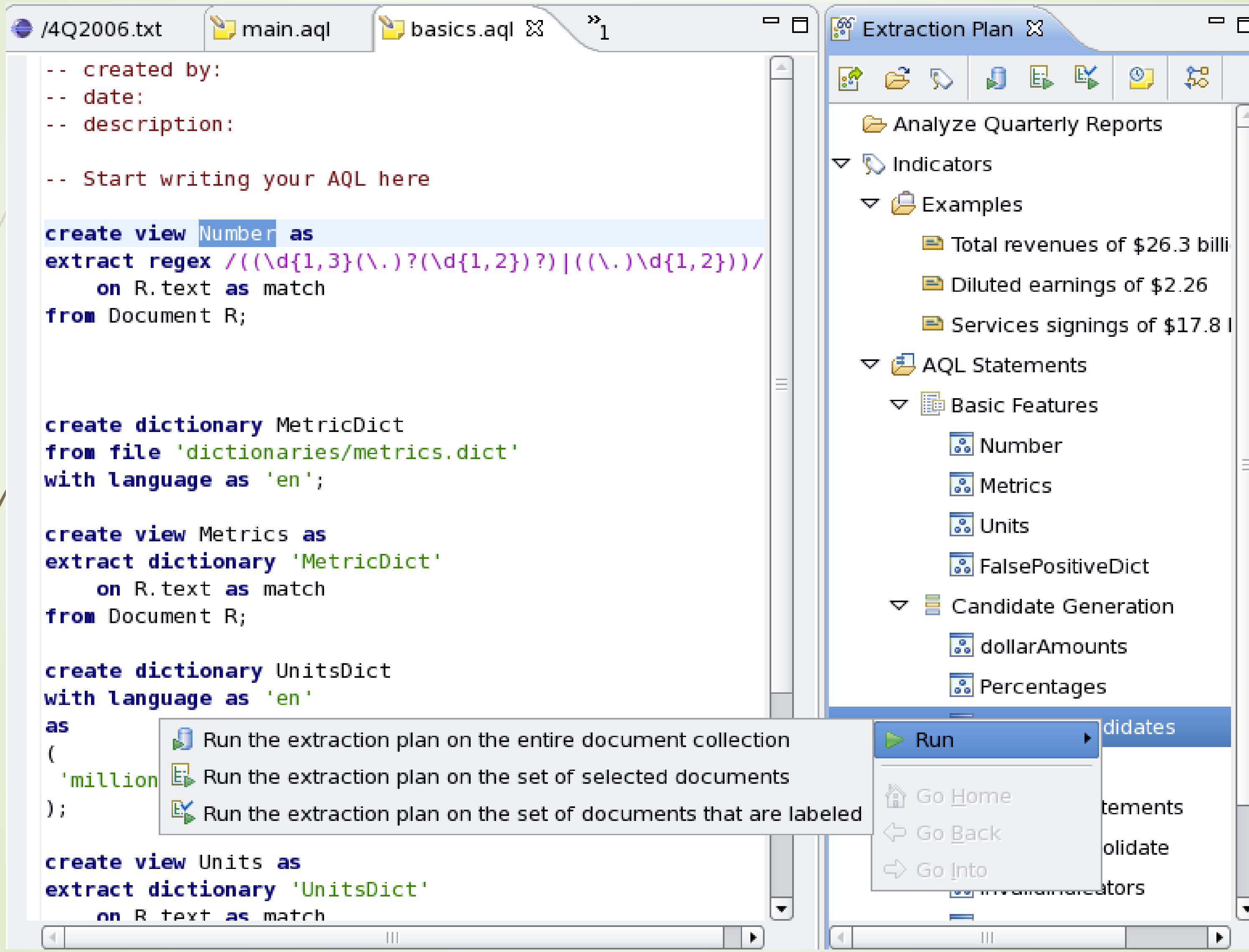
Regular Expression Generator



Regular Expression Builder

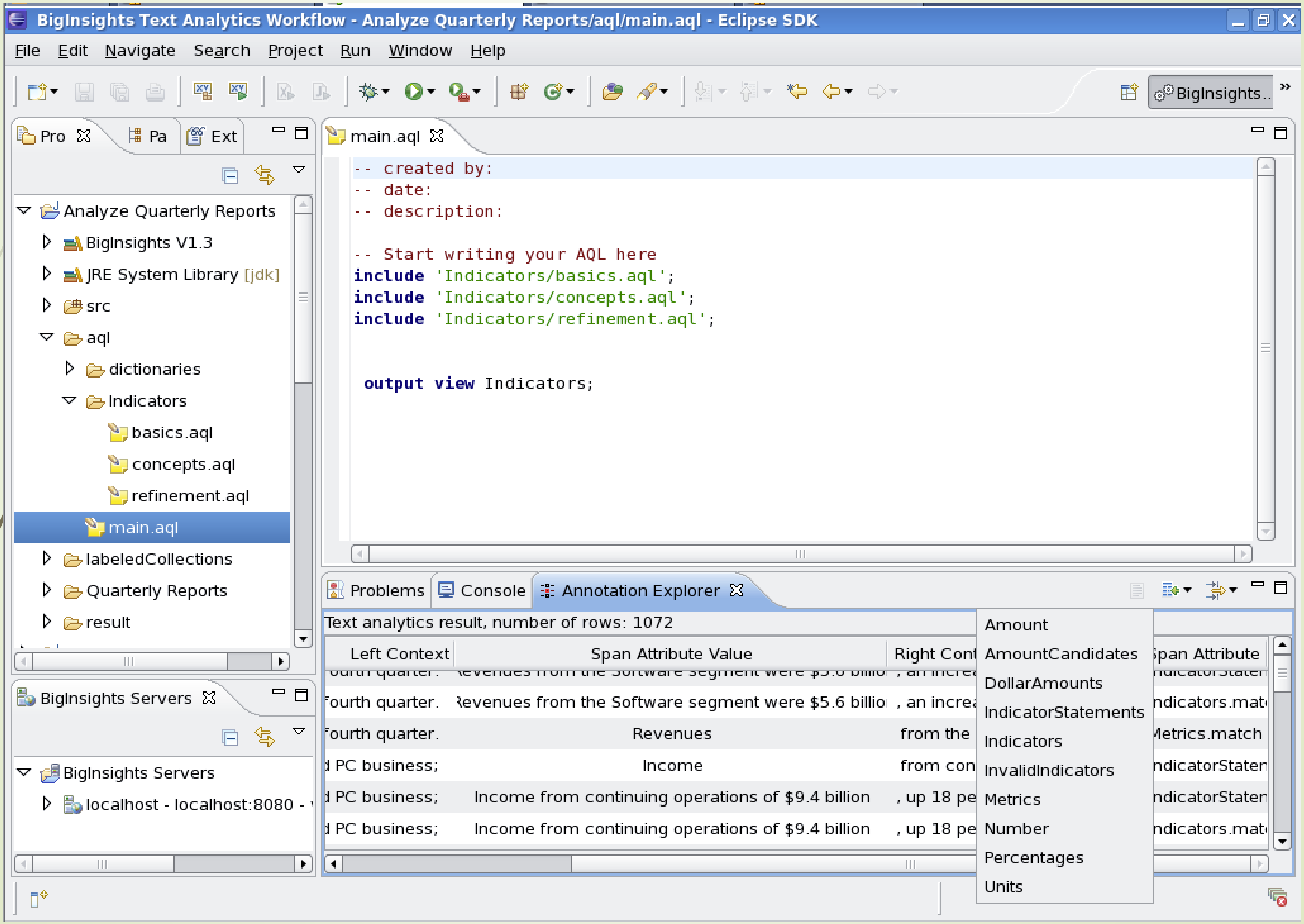


Testing your code

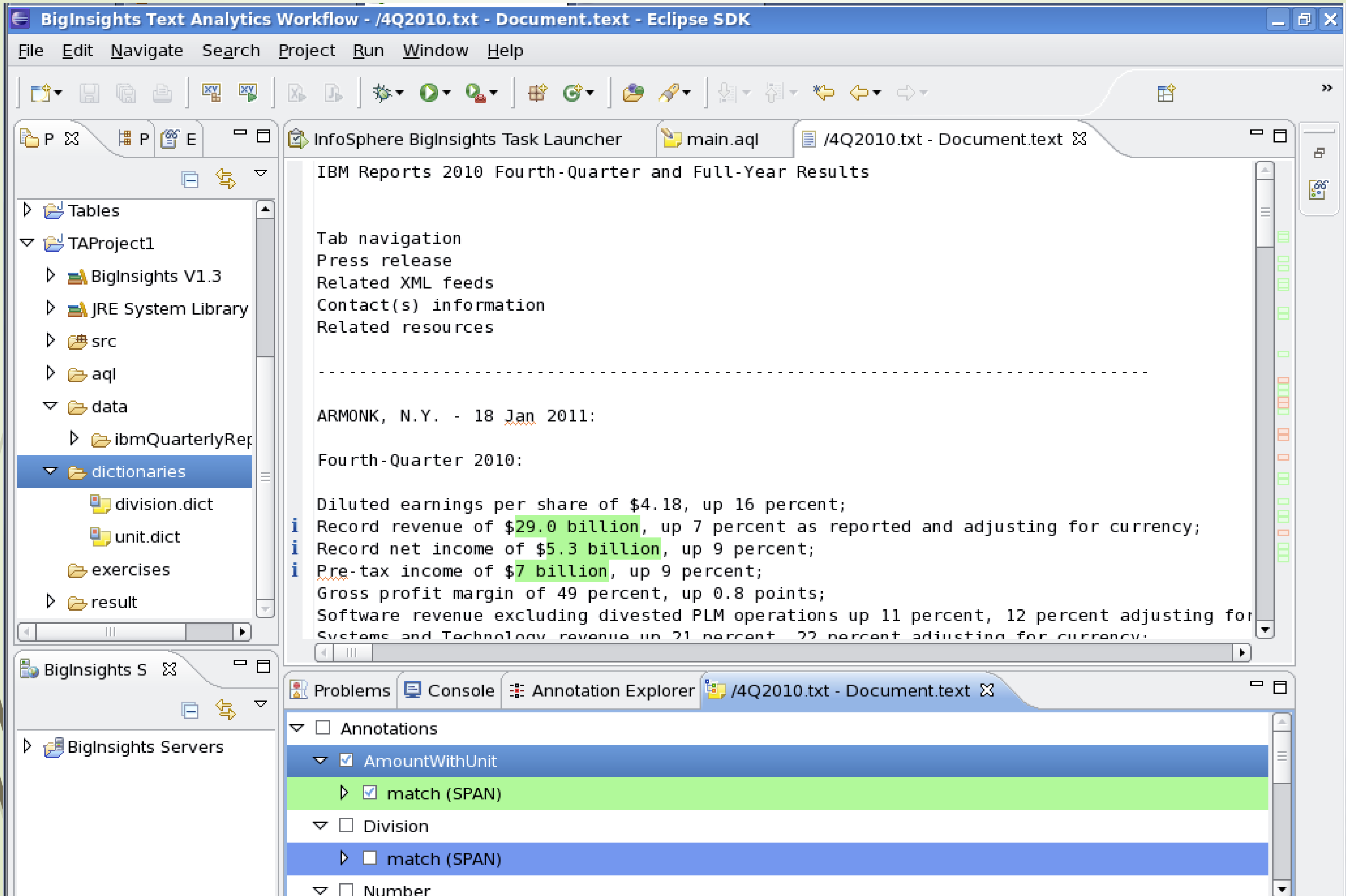


56

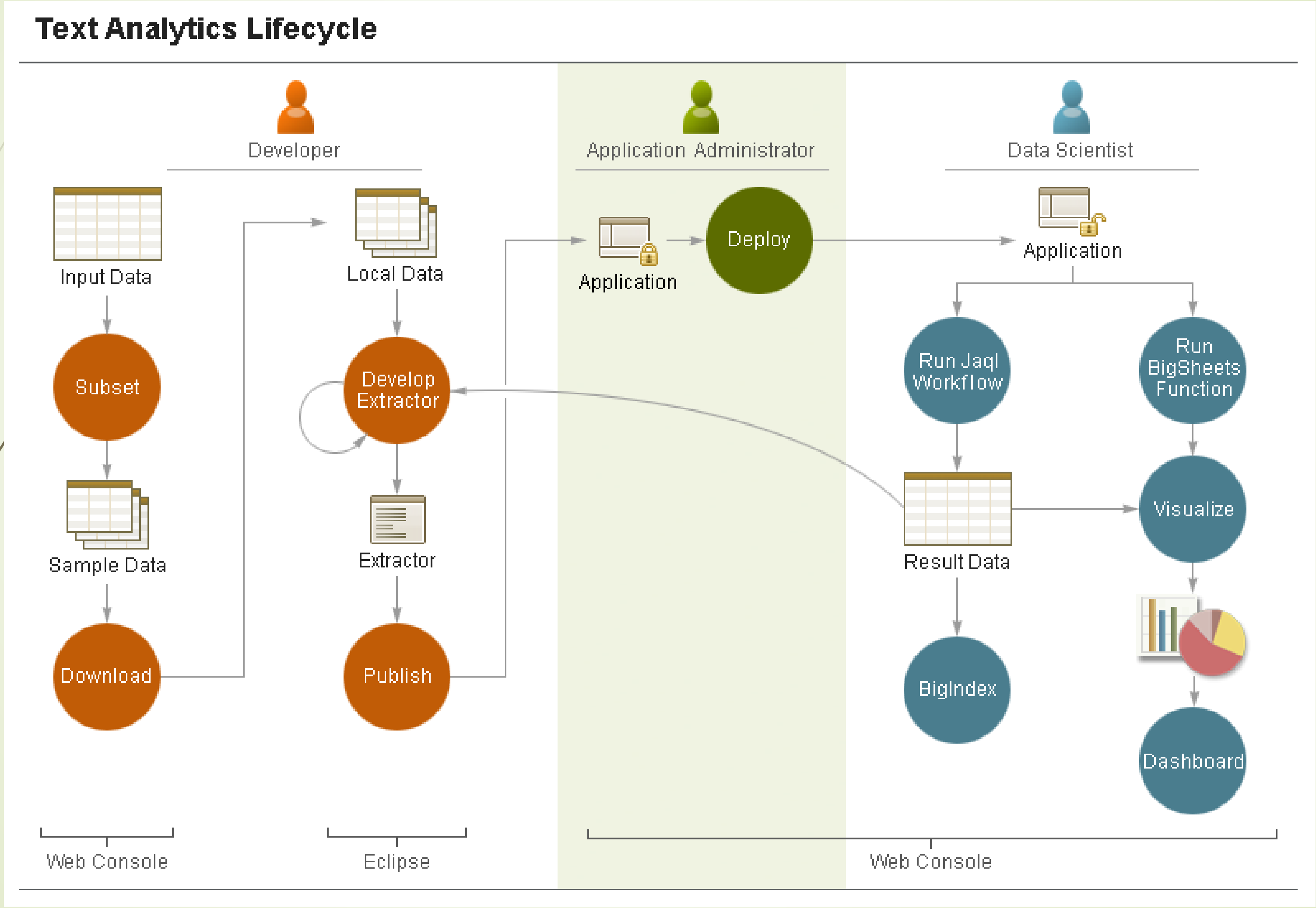
Extractor run results



Document tree view



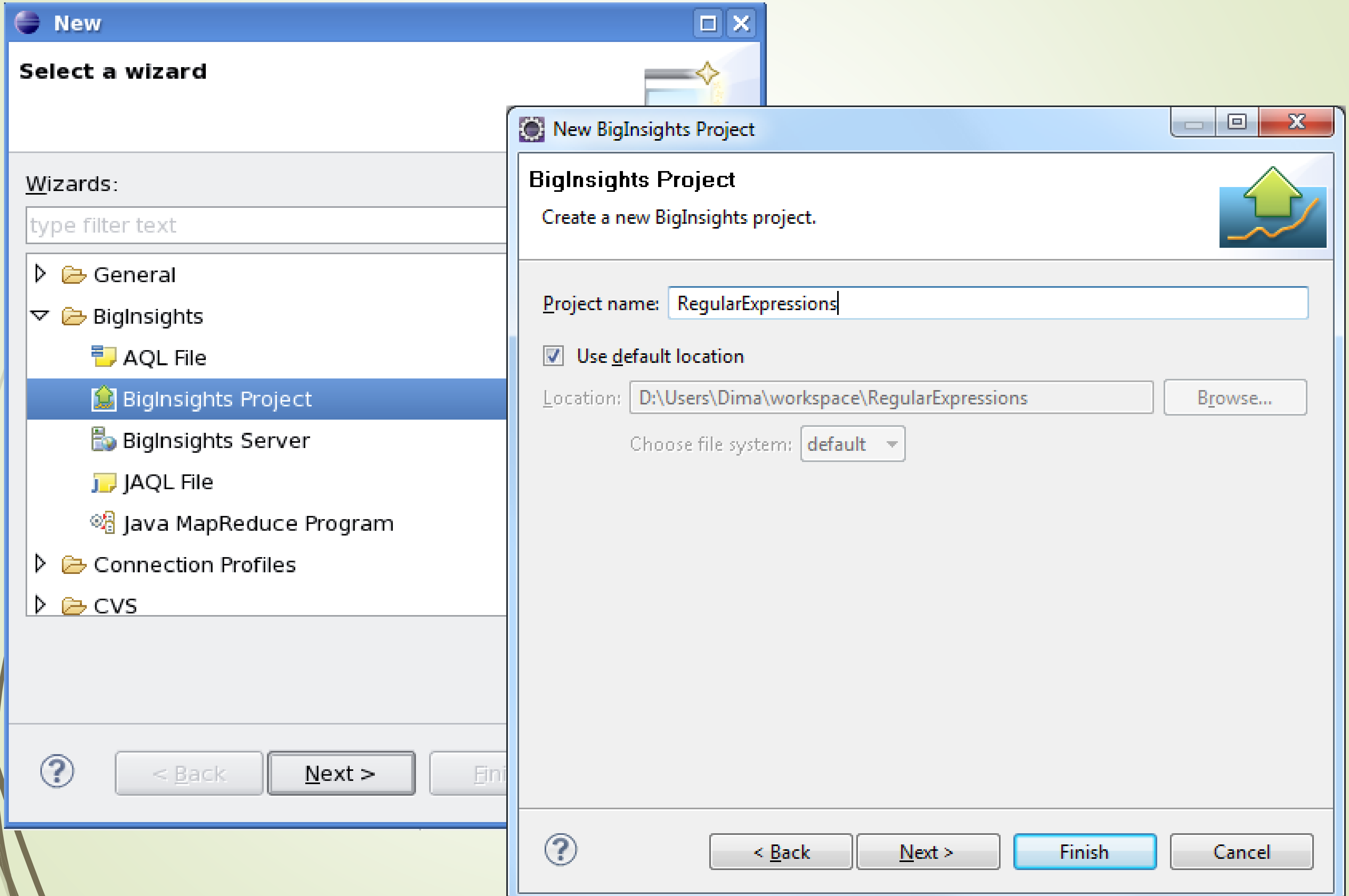
Text Analytics Lifecycle



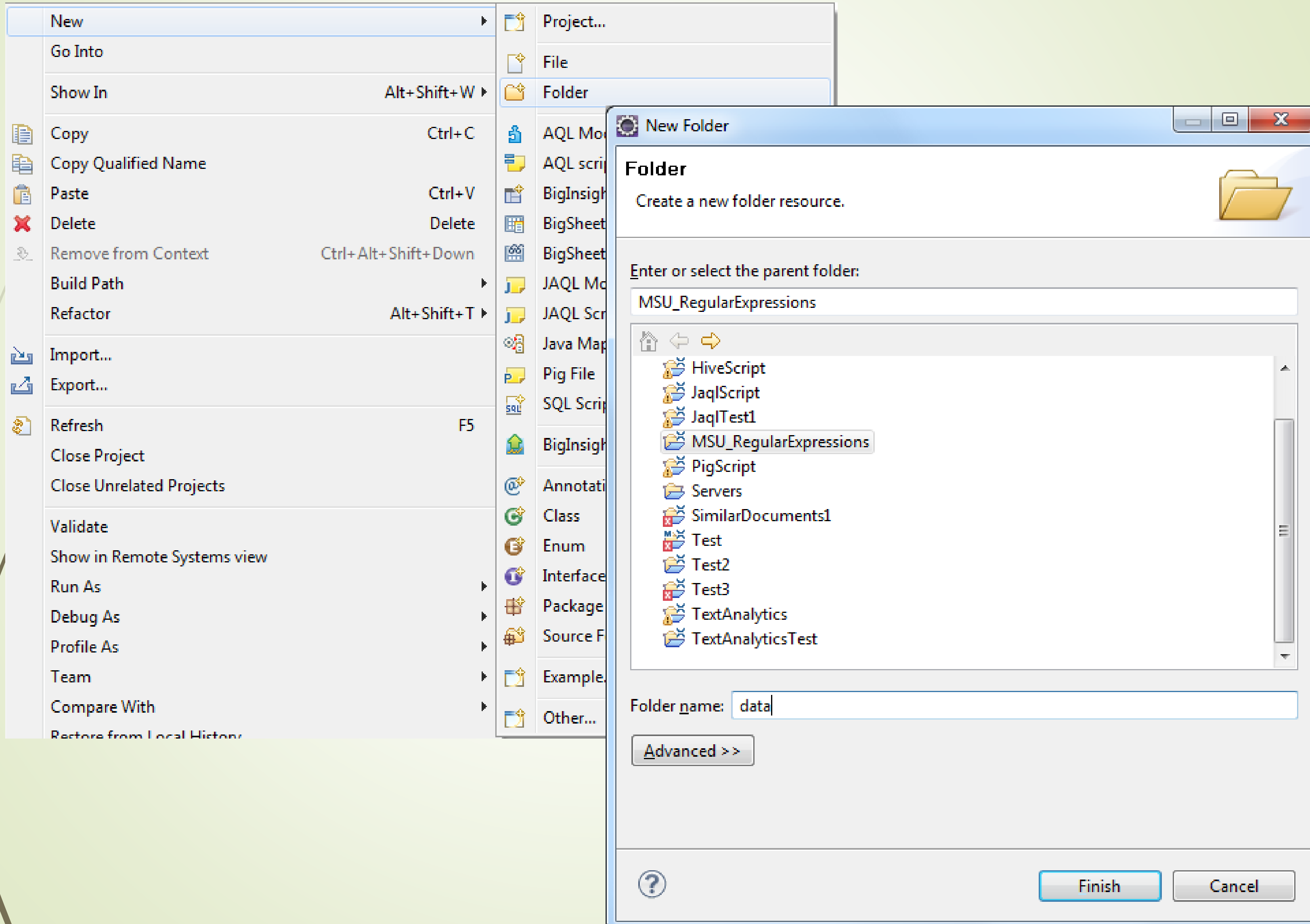
Content

- Text extraction in big data
- Text Analytics in BigInsights
  - Span Algebra
  - Annotation Query Language
- The Eclipse IDE for text analytics
- **Example of creating text analytics project in Eclipse**

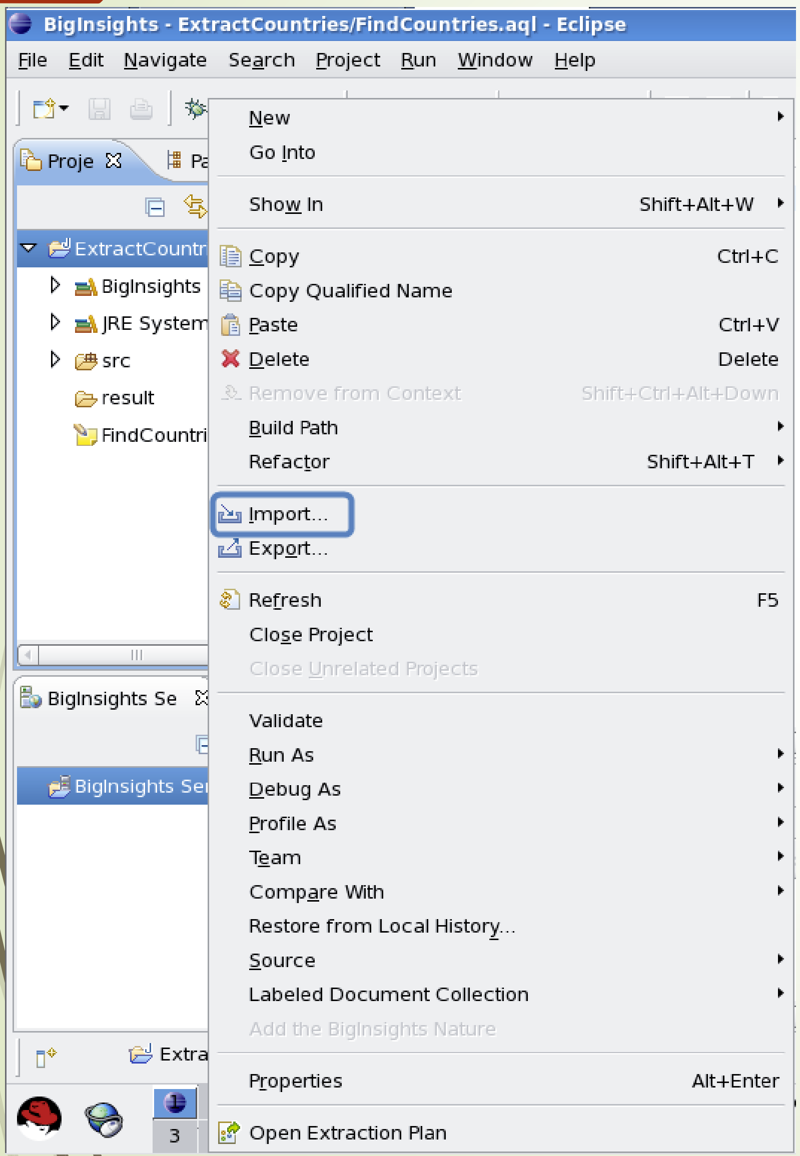
Создание проекта BigInsights

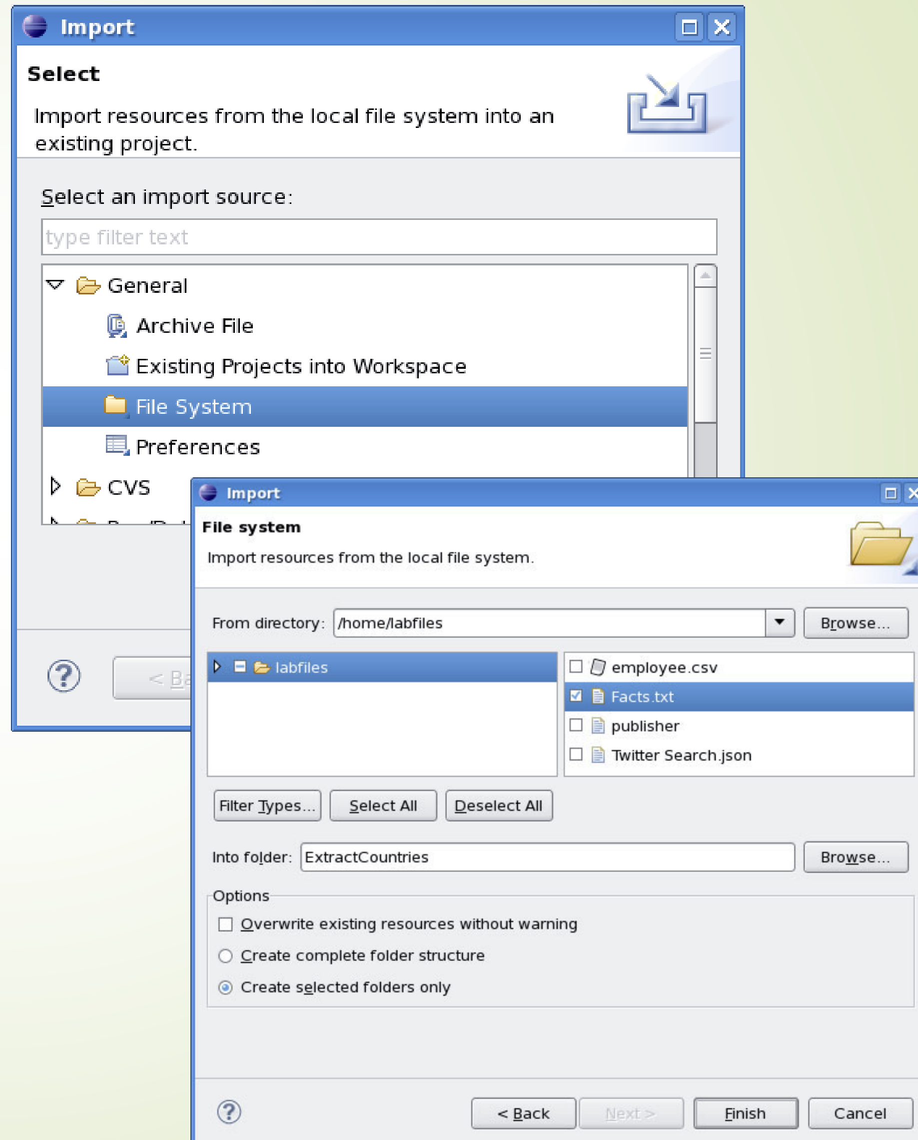


Создание директории data



Импорт данных





Входной файл Facts.txt

"==

Afghanistan

Introduction

Afghanistan

Background: Afghanistan's recent history is characterized by war and civil strife, with intermittent periods of relative calm and …

Geography Afghanistan

Location: Southern Asia, north and west of Pakistan, east of Iran

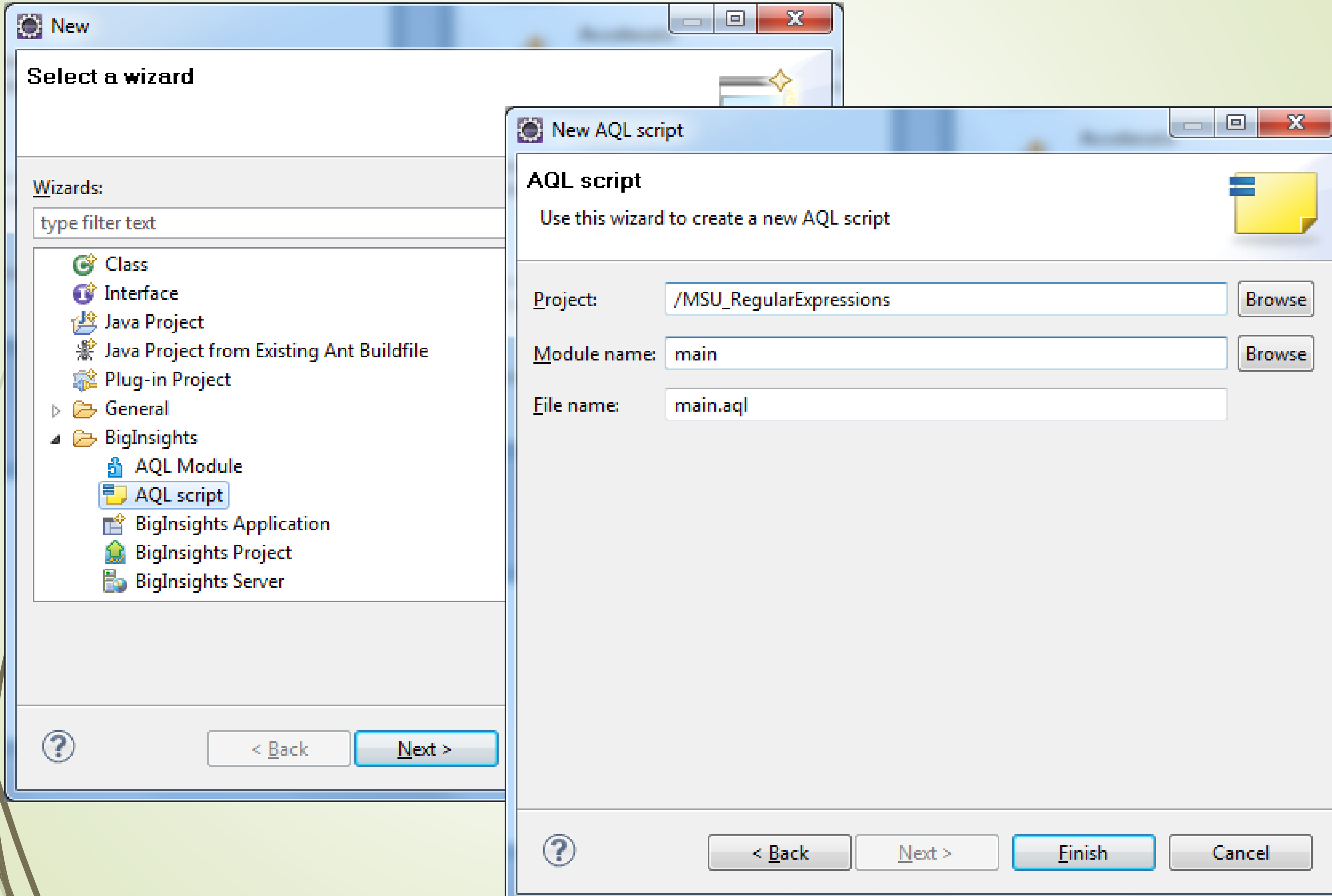
Geographic coordinates: 33 00 N, 65 00 E

Map references: Asia

Area: total: 647,500 sq km water: 0 sq km land: 647,500 sq km

Area - comparative: slightly smaller than Texas

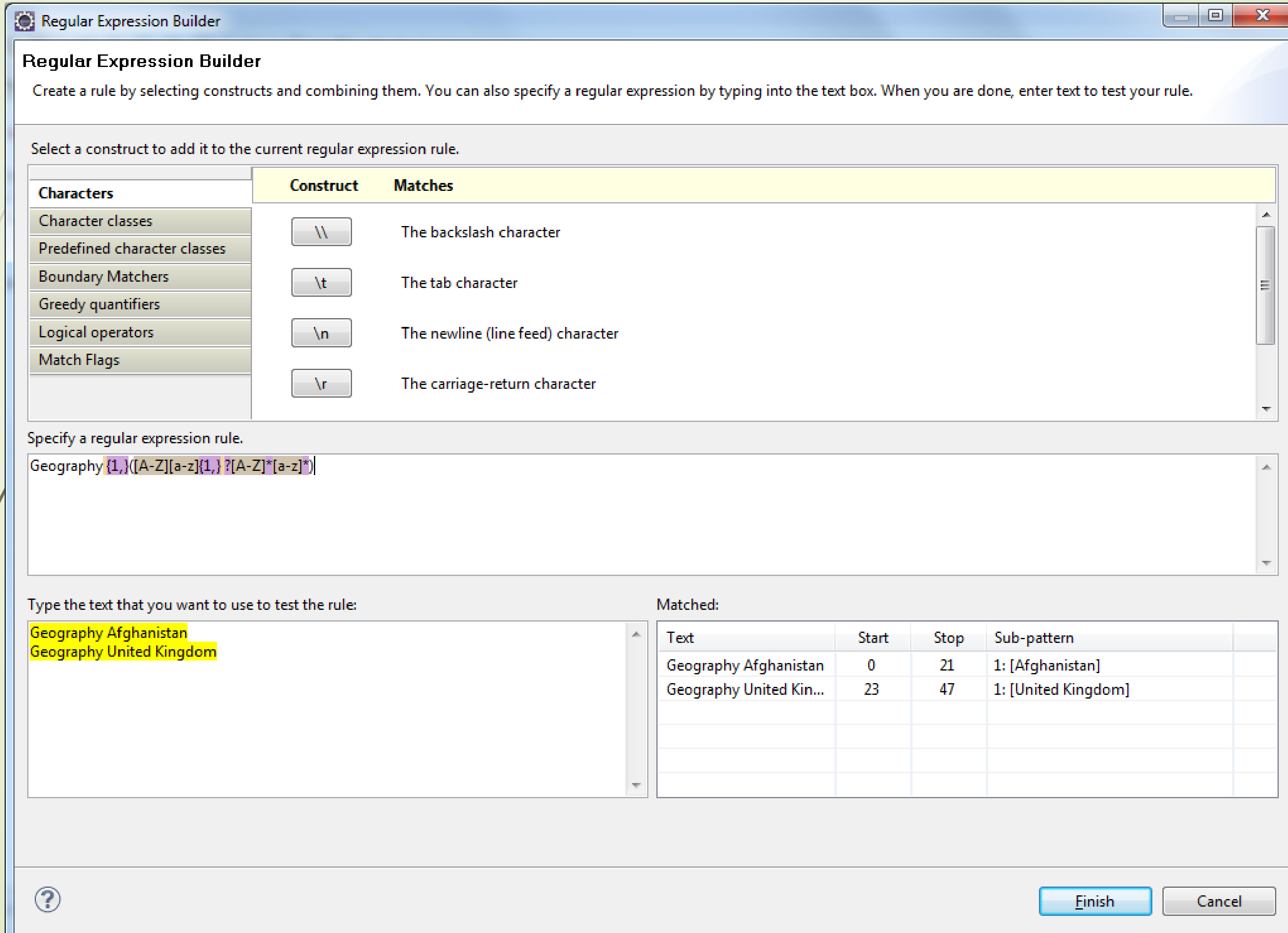
Создание AQL файла



Создание представления Countries

```
module main;

create view Countries as
extract regex // on D.text
return group 1 as country
from Document D;

output view Countries;
```

Конструктор регулярных выражений

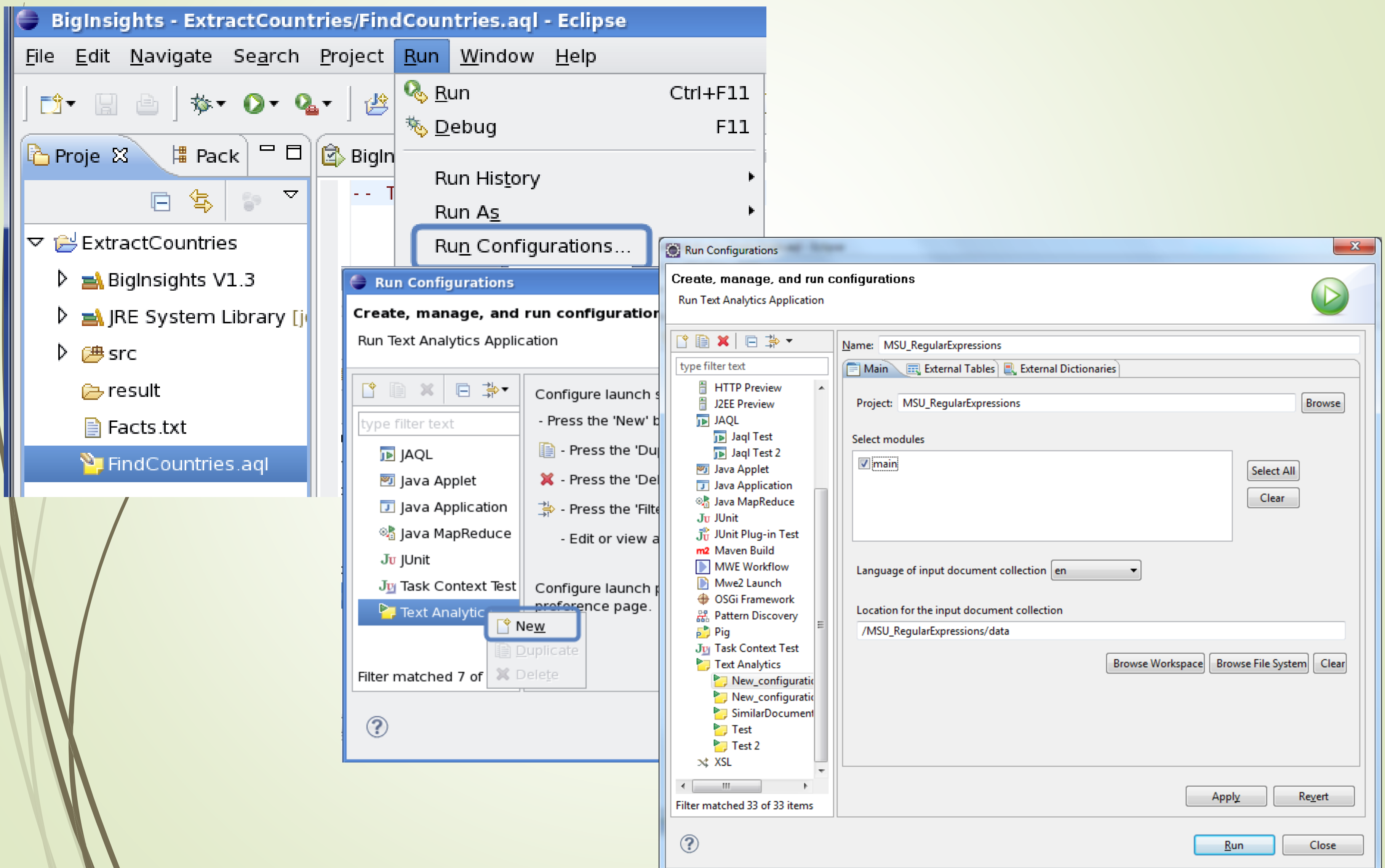


Добавление регулярного выражения в представление Countries

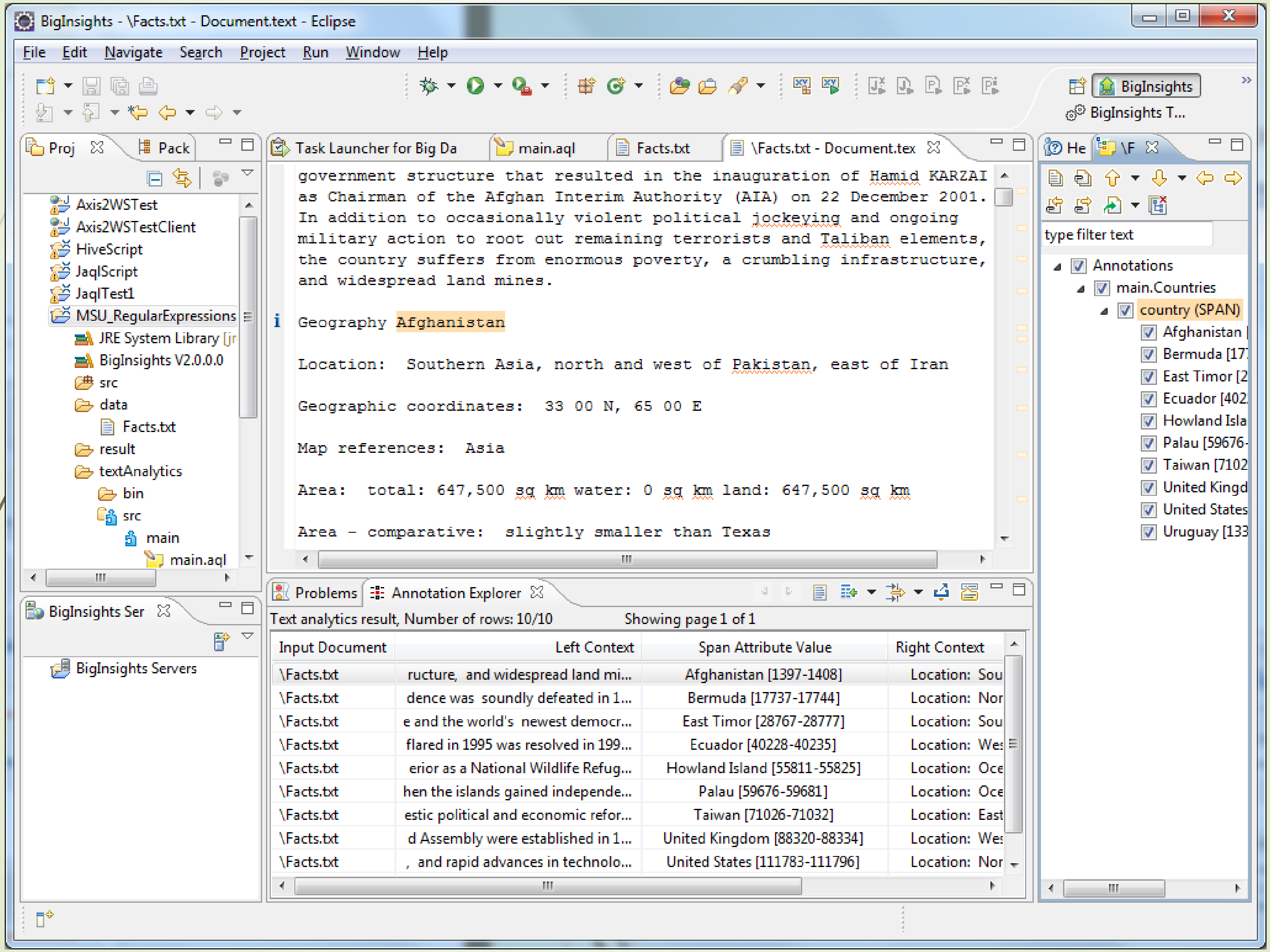
Geography Afghanistan

```
module main;

create view Countries as
extract regex /Geography {1,}([A-Z][a-z]{1,} ?[A-Z]*[a-z]*)/ on D.text
return group 1 as country
from Document D;

output view Countries;
```

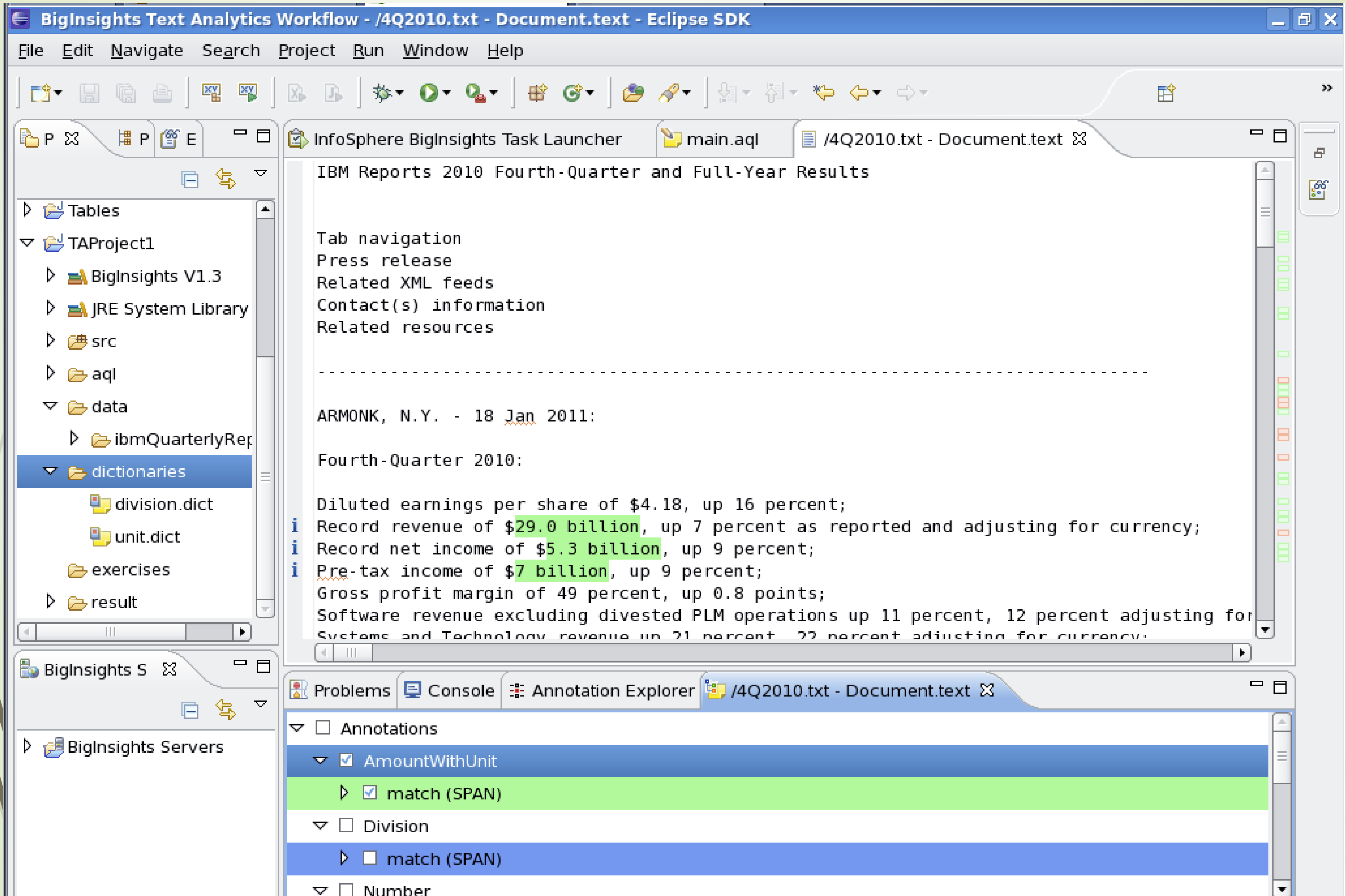
Конфигурация выполнения



Результат выполнения экстракторов



Представление результата в виде дерева документа



Извлечение координат

```
Geographic coordinates:   0 48 N, 176 38 W
```

```
create view Locations as
extract regex /Geographic coordinates: {1,}((\d{1,2} \d{2} [NS]),
(\d{1,3} \d{2} [EW]))/ on D.text
return group 1 as longlat
and group 2 as longitude
and group 3 as latitude
from Document D;
```

Работа со словарями

Exchange rates: afghanis per US dollar - 4,700 (January 2000), 4,750 (February 1999), 17,000 (December 1996), 7,000 (January 1995), 1,900 (January 1994), 1,019 (March 1993), 850 (1991); note - these rates reflect the free market exchange rates rather than the official exchange rate, which was fixed at 50.600 afghanis to the dollar until 1996, when it rose to 2,262.65 per dollar, and finally became fixed again at 3,000.00 per dollar in April 1996

Fiscal year: 21 March - 20 March

```
create dictionary MonthsDict as
(
'January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December'
);
```

```
create view Months as
extract
dictionary 'MonthsDict'
on D.text as month
from Document D;
```

Работа с частями речи

Introduction
Afghanistan
Background: Afghanistan's recent history is characterized by war and civil strife, with intermittent periods of relative calm and stability. The Soviet Union invaded in 1979 but was forced to withdraw 10

```
create view Nouns as

extract part_of_speech 'NN' and 'NNS' with language 'en'

on D.text as noun

from Document D;
```

Работа с блоками

Introduction
Afghanistan
Background: Afghanistan's recent history is characterized by war and civil strife, with intermittent periods of relative calm and

```
create view BlockNouns as
extract blocks
with count between 2 and 3
and separation between 0 and 50 characters
on n.noun
as BlockedNouns
from Nouns n;
```

Работа с паттернами

Geography Afghanistan

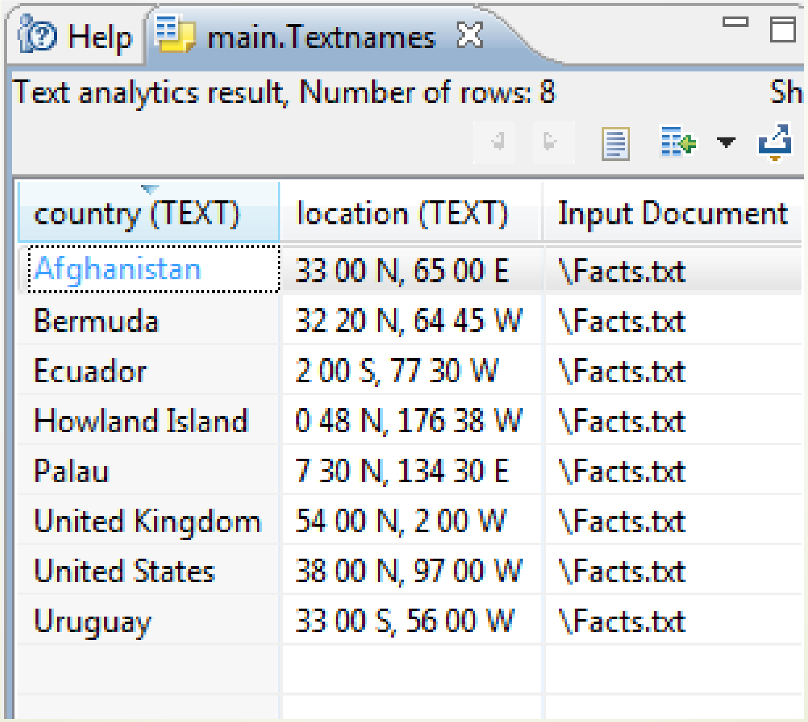
Location: Southern Asia, north and west of Pakistan, east of Iran

Geographic coordinates: 33 00 N, 65 00 E

```
create view CountryLocation as
extract
pattern (<C.country>) <Token>{5,35} (<L.longlat>)
return group 1 as country
and group 2 as location
from Countries C, Locations L;
output view CountryLocation;
```

Работа с Select

```
create view Textnames as
select GetText(C.country) as country, GetText(C.location) as location
from CountryLocation C;
```



country (TEXT)	location (TEXT)	Input Document
Afghanistan	33 00 N, 65 00 E	\Facts.txt
Bermuda	32 20 N, 64 45 W	\Facts.txt
Ecuador	2 00 S, 77 30 W	\Facts.txt
Howland Island	0 48 N, 176 38 W	\Facts.txt
Palau	7 30 N, 134 30 E	\Facts.txt
United Kingdom	54 00 N, 2 00 W	\Facts.txt
United States	38 00 N, 97 00 W	\Facts.txt
Uruguay	33 00 S, 56 00 W	\Facts.txt

Questions ?

Анонс

- Практикум
  - 27 марта, 18:30
  - Машинный зал № 4