INFO116

RDF and SPARQL

Querying semantic data

- Data can be in various forms
  - rdf files
  - triplestores
    - Virtuoso
    - Jena/Fuseki
- Query language: SPARQL
- SPARQL: select, construct, ask, describe

SPARQL

- SPARQL (select) returns results from RDF graphs based on pattern matching
  - variable binding
    - John -> likes -> Mary, John -> likes -> Cathy
      - ?x -> likes -> Mary
      - ?x -> likes -> ?y
      - John -> likes -> ?x

http://www.slideshare.net/LeeFeigenbaum/

URIs

Write full URIs:

```
<http://this.is.a/full/URI/written#out>
```

Abbreviate URIs with prefixes:

```
PREFIX foo: <http://this.is.a/URI/prefix#>
… foo:bar …
```

⇒ http://this.is.a/URI/prefix#bar

Shortcuts:

a ⇒ rdf:type

Variables

Variables:

```
?var1, ?anotherVar, ?and_one_more
```

Literals

Plain literals:

```
"a plain literal"
```

Plain literal with language tag:

```
"bonjour"@fr
```

Typed literal:

```
"13"^^xsd:integer
```

Shortcuts:

```
true ⇒ "true"^^xsd:boolean
3    ⇒    "3"^^xsd:integer
4.2  ⇒  "4.2"^^xsd:decimal
```

Comments

Comments:

```
# Comments start with a '#' and
# continue to the end of the line
```

Triple

Match an exact RDF triple:

```
ex:myWidget ex:partNumber "XY24Z1" .
```

Match one variable:

```
?person foaf:name "Lee Feigenbaum" .
```

Match multiple variables:

```
conf:SemTech2009 ?property ?value .
```

Common Prefixes

prefix...	...stands for
rdf:	***http://xmlns.com/foaf/0.1/***
rdfs:	http://www.w3.org/2000/01/rdf-schema#
owl:	***http://www.w3.org/2002/07/owl#***
xsd:	http://www.w3.org/2001/XMLSchema#
dc:	***http://purl.org/dc/elements/1.1/***
foaf:	http://xmlns.com/foaf/0.1/

More common prefixes at http://prefix.cc

Anatomy of a Query

Declare prefix shortcuts (*optional*)

```
PREFIX foo: <…>
PREFIX bar: <…>
…
SELECT …
FROM <…>
FROM NAMED <…>
WHERE {
    …
}
GROUP BY …
HAVING …
ORDER BY …
LIMIT …
OFFSET …
VALUES …
```

Define the dataset (*optional*)

Query result clause

Query pattern

Query modifiers (*optional*)

4 Types of SPARQL Queries

https://code.google.com/p/tdwg-rdf/wiki/Beginners6SPARQL

SELECT queries

Project out specific variables and expressions:

```
SELECT ?c ?cap (1000 * ?people AS ?pop)
```

Project out all variables:

```
SELECT *
```

Project out distinct combinations only:

```
SELECT DISTINCT ?country
```

?c	?cap	?pop
ex:France	*ex:Paris*	*63,500,000*
ex:Canada	*ex:Ottawa*	*32,900,000*
	ex:Rome	*58,900,000*

CONSTRUCT

Construct RDF triples/graphs:

```
CONSTRUCT {
   ?country a ex:HolidayDestination ;
       ex:arrive_at ?capital ;
       ex:population ?population .
}
```

Results in RDF triples (in any RDF serialization):

```
ex:France a ex:HolidayDestination ;
    ex:arrive_at ex:Paris ;
    ex:population 635000000 .
ex:Canada a ex:HolidayDestination ;
    ex:arrive_at ex:Ottawa ;
    ex:population 329000000 .
```

ASK queries

Ask whether or not there are any matches:

```
ASK
```

Result is either "true" or "false" (in XML or JSON):

```
true, false
```

DESCRIBE queries

Describe the resources matched by the given variables:

```
DESCRIBE ?country
```

Result is RDF triples (in any RDF serialization) :

```
ex:France a geo:Country ;
  ex:continent geo:Europe ;
  ex:flag <http://…/flag-france.png> ;
  …
```

```
DESCRIBE ?X WHERE {
 ?X A ?Y.
 }
LIMIT 1
```

SPARQL Tutorial

http://jena.apache.org/tutorials/sparql.html

Blank nodes

- Weather forecast by the hour, for each city
  - time (+date?)
    - temp
- Bergen -> time -> 08:00; temp -> 10
- Bergen -> time -> 09:00; temp -> 11
- Bergen -> time -> 10:00; temp -> 11.5

Blank nodes to the rescue

- Create a new class, Forecast
- Bergen -> forecast -> a
  - a -> time -> 08:00; temp -> 10
  - a -> time -> 09:00; temp -> 11
- What to call “a”?
  - Unique name? (e.g. BRG08)
  - No name?

A first query

- SELECT ?x

  WHERE { ?x <http://www.w3.org/2001/vcard-rdf/3.0#FN>
  "John Smith" }

  | x |

  ===

  | <http://somewhere/JohnSmith/> |

Query solutions

- SELECT ?x ?fname

  WHERE {?x <http://www.w3.org/2001/vcard-rdf/3.0#FN> ?fname}

  | x | name |

  ==========

  | <http://somewhere/RebeccaSmith/> | "Becky Smith" |

  | <http://somewhere/SarahJones/> | "Sarah Jones" |

  | <http://somewhere/JohnSmith/> | "John Smith" |

  | <http://somewhere/MattJones/> | "Matt Jones" |

Basic patterns

- SELECT ?givenName

  WHERE

  { ?y <http://www.w3.org/2001/vcard-rdf/3.0#Family> "Smith" .

  ?y <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?givenName .

  }

  | givenName |

  ==========

  | "John" |

  | "Rebecca" |

- How would you list all FullNames and the corresponding Firstname, Secondname?

Qnames

- PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

```
SELECT ?givenName
WHERE
 { ?y vcard:Family "Smith" .
   ?y vcard:Given  ?givenName .
 }
```

Blank nodes

- PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

  SELECT ?y ?givenName

  WHERE

  { ?y vcard:Family "Smith" .

  ?y vcard:Given ?givenName .

  }

  | y | givenName |

  ===

  | _:b0 | "John" |

  | _:b1 | "Rebecca" |

SPARQL Filters

- *SPARQL* **`FILTER`***s eliminate solutions that do not cause an expression to evaluate to true.*
- *Place* **`FILTER`***s in a query inline within a basic graph pattern*

```
A . B . FILTER ( …expr… )
```

String matching

- FILTER regex(?x, "pattern" [, "flags"])
- PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

  SELECT ?g

  WHERE

  { ?y vcard:Given ?g .

    FILTER regex(?g, "r", "i") }

  | g |

  ==============

  | "Rebecca" |

  | "Sarah" |

Testing values (1/2)

- Add age info:

<http://somewhere/RebeccaSmith/>

info:age "23"^^xsd:integer ;

vCard:FN "Becky Smith" ;

vCard:N [vCard:Family "Smith" ;

vCard:Given "Rebecca"] .

<http://somewhere/RebeccaSmith/>

info:age "25"^^xsd:integer ;

Testing values (2/2)

- PREFIX info: <http://somewhere/peopleInfo#>

  SELECT ?resource

  WHERE

  {

  ?resource info:age ?age .

  FILTER (?age >= 24)

  }

| resource |

==========

| <http://somewhere/JohnSmith/> |

Category	Functions / Operators	Examples
Logical & Comparisons	`!, &&, ||, =, !=, <, <=, >, >=, IN, NOT IN`	`?hasPermit || ?age < 25`
Conditionals *(SPARQL 1.1)*	`EXISTS, NOT EXISTS, IF, COALESCE`	`NOT EXISTS { ?p foaf:mbox ?email }`
Math	`+, -, *, /, abs, round, ceil, floor, RAND`	`?decimal * 10 > ?minPercent`
Strings *(SPARQL 1.1)*	`STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, CONCAT, STRENDS, CONTAINS, STRBEFORE, STRAFTER`	`STRLEN(?description) < 255`
Date/time *(SPARQL 1.1)*	`now, year, month, day, hours, minutes, seconds, timezone, tz`	`month(now()) < 4`
SPARQL tests	`isURI, isBlank, isLiteral, isNumeric, bound`	`isURI(?person) || !bound(?person)`
Constructors *(SPARQL 1.1)*	`URI, BNODE, STRDT, STRLANG, UUID, STRUUID`	`STRLANG(?text, "en") = "hello"@en`
Accessors	`str, lang, datatype`	`lang(?title) = "en"`
Hashing *(1.1)*	`MD5, SHA1, SHA256, SHA512`	`BIND(SHA256(?email) AS ?hash)`
Miscellaneous	`sameTerm, langMatches, regex, REPLACE`	`regex(?ssn, "\\d{3}-\\d{2}-\\d{4}")`

Optionals (1/2)

- PREFIX info: <http://somewhere/peopleInfo#>

  PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

  SELECT ?name ?age

  WHERE

  {

  ?person vcard:FN ?name .

  OPTIONAL { ?person info:age ?age }

  }

  | name | age |

  ==========

  | "Becky Smith" | 23 |

  | "Sarah Jones" | |

  | "John Smith" | 25 |

  | "Matt Jones" | |

Optionals (2/2)

- PREFIX info: <http://somewhere/peopleInfo#>

  PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

  SELECT ?name ?age

  WHERE

  {

  ?person vcard:FN ?name .

  ?person info:age ?age .

  }

  | name | age |

  ==========

  | "Becky Smith" | 23 |

  | "John Smith" | 25 |

Union (1/3)

- Suppose you had the following data:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:a foaf:name "Matt Jones" .

_:b foaf:name "Sarah Jones" .

_:c vcard:FN "Becky Smith" .

_:d vcard:FN "John Smith" .

Union (2/3)

- PREFIX foaf: <http://xmlns.com/foaf/0.1/>

  PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

  SELECT ?name

  WHERE

  {

    { [] foaf:name ?name } UNION { [] vCard:FN ?name }

  }

  | name |

  ==================

  | "Matt Jones" |

  | "Sarah Jones" |

  | "Becky Smith" |

  | "John Smith" |

Union (3/3)

- Remembering where the data comes from

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name1 ?name2
WHERE
{
   { [] foaf:name ?name1 } UNION { [] vCard:FN ?name2 }
}
```

name1	name2
"Matt Jones"	
"Sarah Jones"	
	"Becky Smith"
	"John Smith"

UNION and OPTIONAL

- Can often give same results … but not always

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name1 ?name2
WHERE
{
  OPTIONAL { ?x  foaf:name  ?name1 }
  OPTIONAL { ?x  vCard:FN   ?name2 }
}
```

```
-------------------------------
| name1                       | name2        |
=================================================
| "Matt Jones"                |              |
| "Sarah Jones"               |              |
|                             | "Becky Smith" |
|                             | "John Smith"  |
```

Combining SPARQL Graph Patterns

*Consider **A** and **B** as graph patterns.*

A Basic Graph Pattern – one or more triple patterns

```
A . B
```

⇒ Conjunction. Join together the results of solving A and B by matching the values of any variables in common.

Optional Graph Patterns

```
A OPTIONAL { B }
```

⇒ Left join. Join together the results of solving A and B by matching the values of any variables in common, if possible. Keep all solutions from A whether or not there's a matching solution in B

Combining SPARQL Graph Patterns

*Consider **A** and **B** as graph patterns.*

Either-or Graph Patterns

```
{ A } UNION { B }
```

⇒ Disjunction. Include both the results of solving A and the results of solving B.

"Subtracted" Graph Patterns (SPARQL 1.1)

```
A MINUS { B }
```

⇒ Negation. Solve A. Solve B. Include only those results from solving A that are *not compatible* with any of the results from B.

Property Paths *(SPARQL 1.1)*

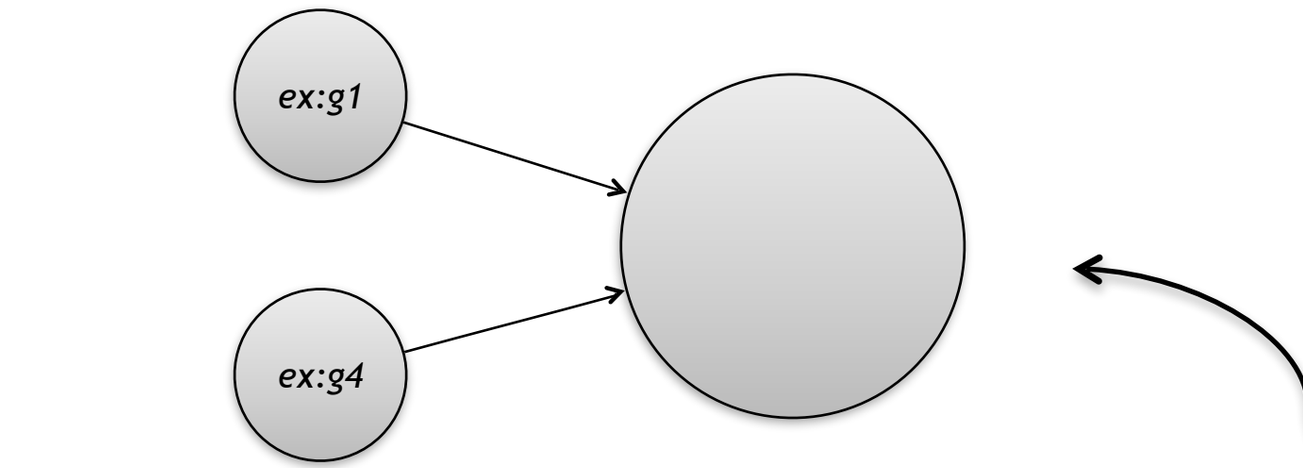
- Property paths allow triple patterns to match arbitrary-length paths through a graph
- Predicates are combined with regular-expression-like operators:

Construct	Meaning
`path1/path2`	*Forwards path (`path1` followed by `path2`)*
`^path1`	*Backwards path (object to subject)*
`path1|path2`	*Either `path1` or `path2`*
`path1*`	*`path1`, repeated zero or more times*
`path1+`	*`path1`, repeated one or more times*
`path1?`	*`path1`, optionally*
`!uri`	*Any predicate except `uri`*
`!^uri`	*Any backwards (object to subject) predicate except `uri`*

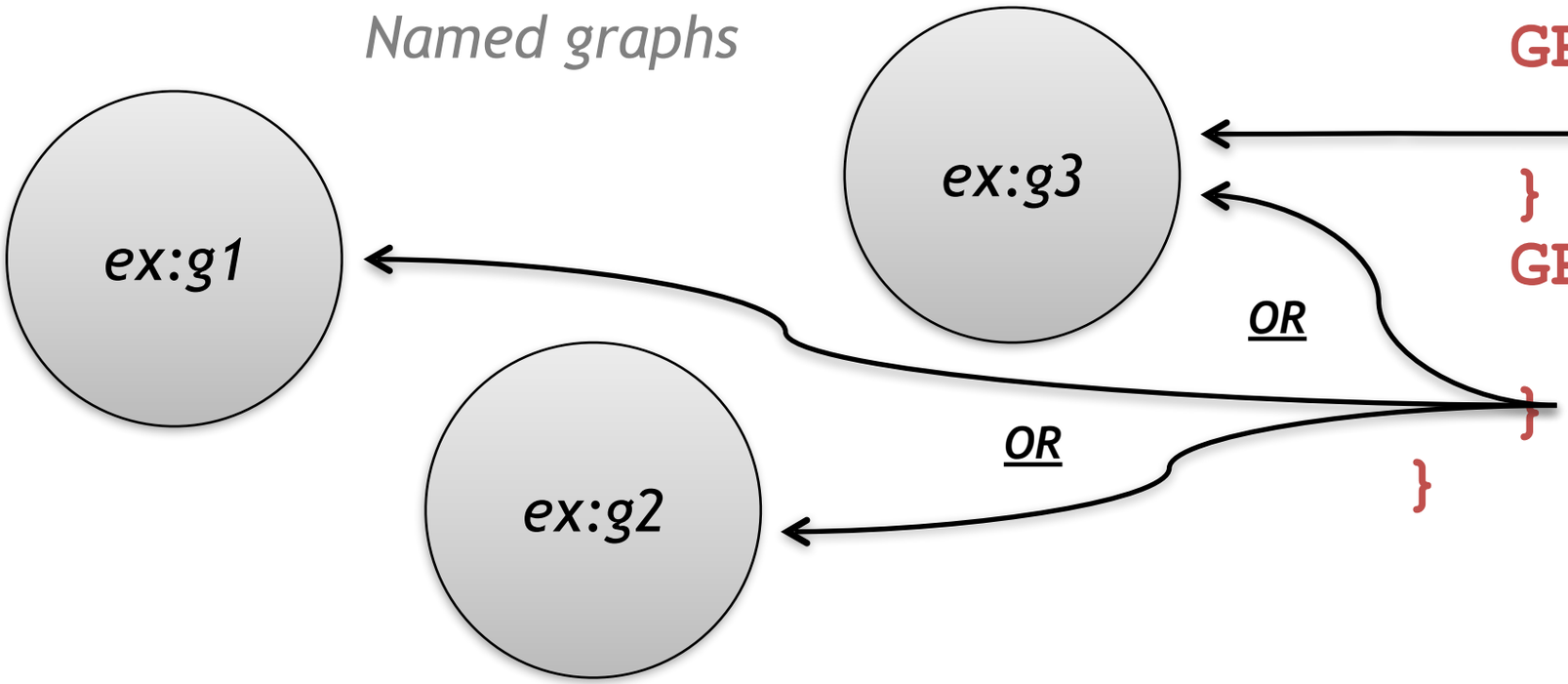
RDF Datasets

A SPARQL queries a *default graph* (normally) and zero or more *named graphs* (when inside a **GRAPH** clause).

Default graph
(the merge of zero or more graphs)



Named graphs




```
PREFIX ex: <…>
SELECT …
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
   … A …
   GRAPH ex:g3 {
      … B …
   }
   GRAPH ?graph {
      … C …
   }
}
```

SPARQL Over HTTP (the SPARQL Protocol)

`http://host.domain.com/sparql/endpoint?`<*parameters*>

where <*parameters*> can include:

`query=`<*encoded query string*>

e.g. `SELECT+*%0DWHERE+{…`

`default-graph-uri=`<*encoded graph URI*>

e.g. `http%3A%2F%2Fexmaple.com%2Ffoo…`

n.b. zero of more occurrences of `default-graph-uri`

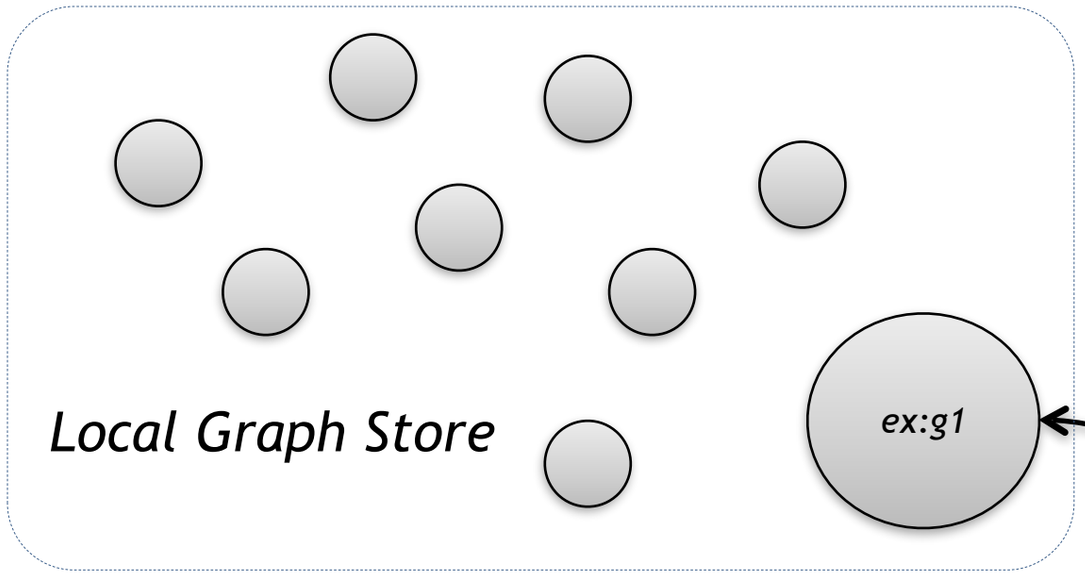
`named-graph-uri=`<*encoded graph URI*>

e.g. `http%3A%2F%2Fexmaple.com%2Fbar…`

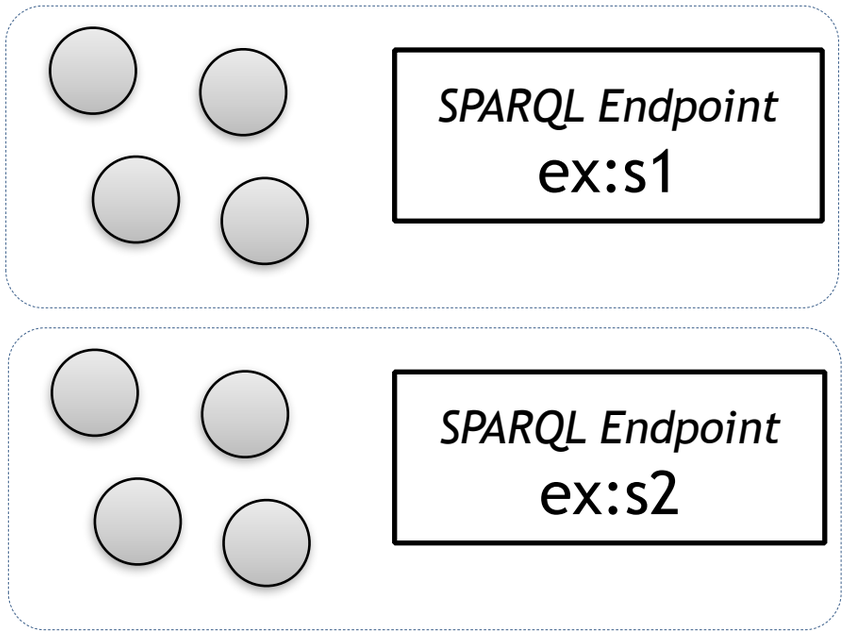
n.b. zero of more occurrences of `named-graph-uri`

HTTP `GET` or `POST`. Graphs given in the protocol override graphs given in the query.

Federated Query *(SPARQL 1.1)*




```
PREFIX ex: <…>
SELECT …
FROM ex:g1
WHERE {
    … A …
    SERVICE ex:s1 {
        … B …
    }
    SERVICE ex:s2 {
        … C …
    }
}
```



Some Public SPARQL Endpoints

Name	URL	What's there?
SPARQLer	*http://sparql.org/sparql.html*	*General-purpose query endpoint for Web-accessible data*
DBPedia	*http://dbpedia.org/sparql*	*Extensive RDF data from Wikipedia*
DBLP	*http://www4.wiwiss.fu-berlin.de/dblp/snorql/*	*Bibliographic data from computer science journals and conferences*
LinkedMDB	*http://data.linkedmdb.org/sparql*	*Films, actors, directors, writers, producers, etc.*
World Factbook	*http://www4.wiwiss.fu-berlin.de/factbook/snorql/*	*Country statistics from the CIA World Factbook*
bio2rdf	*http://bio2rdf.org/sparql*	*Bioinformatics data from around 40 public databases*