# Annotation Query Language

06/03/2011

CS717 - SRL

## Outline*

- Motivation
- SystemText background
- AQL

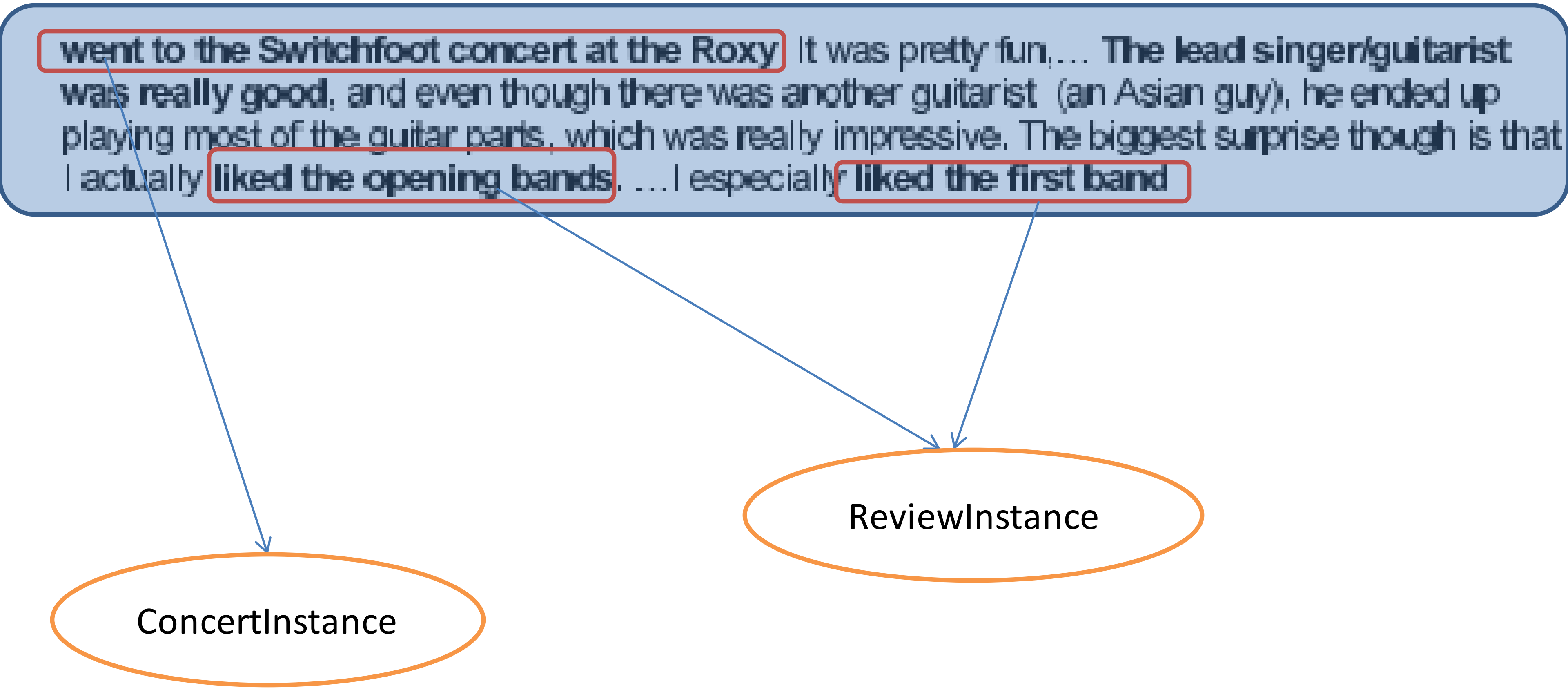
* Most examples are borrowed from SystemT literature (references at the end)

## Outline

- **Motivation**
- SystemText background
- AQL

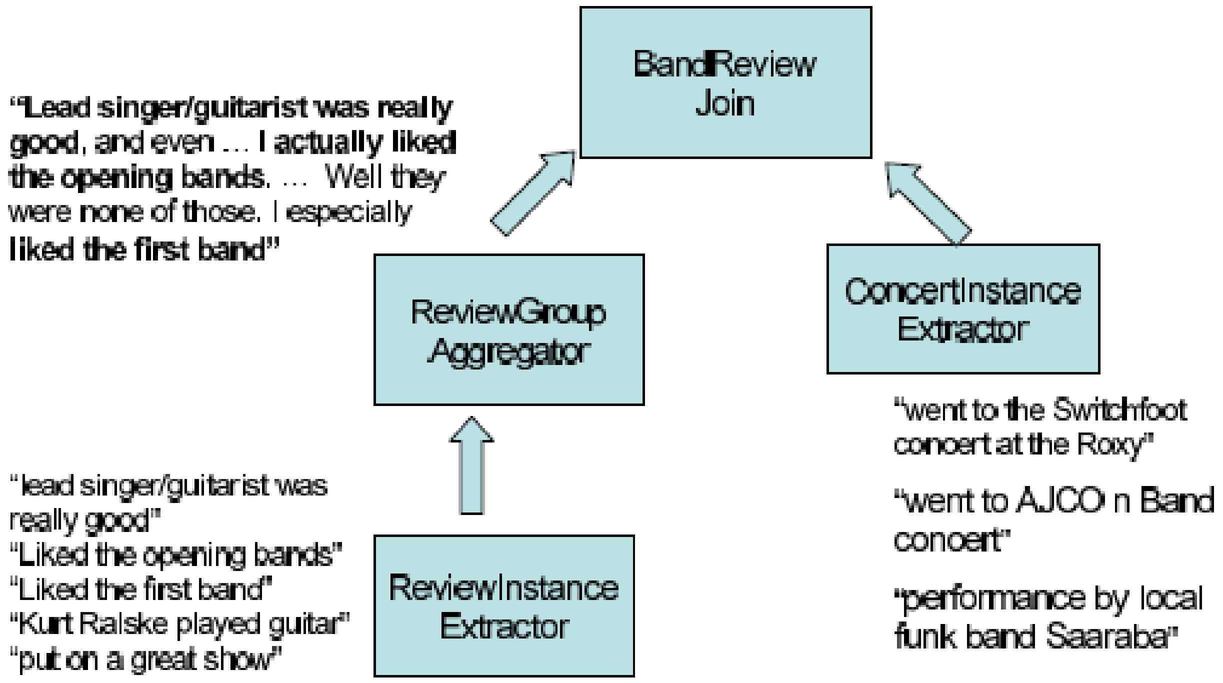
# Information Extraction task

## Extracting informal reviews from blogs



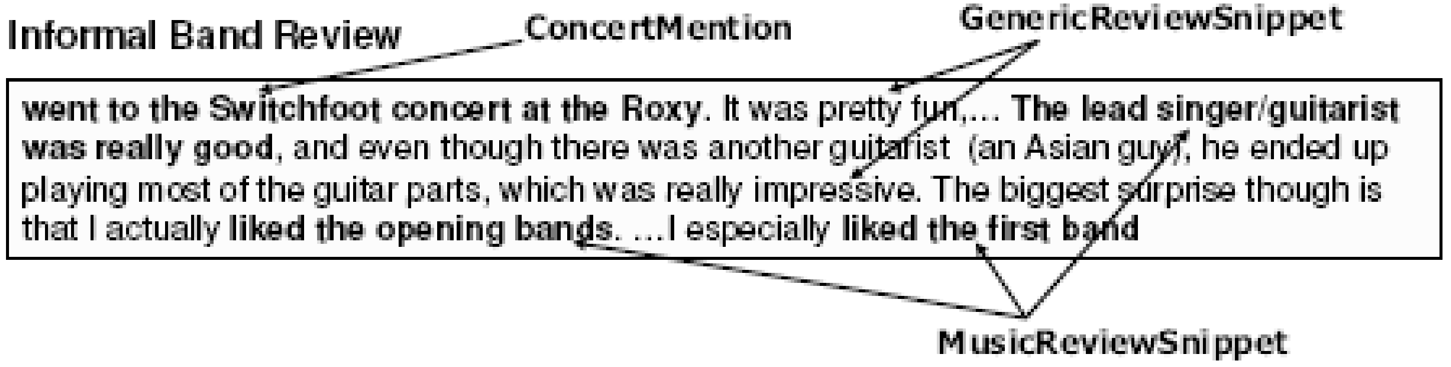
went to the Switchfoot concert at the Roxy. It was pretty fun,… The lead singer/guitarist was really good, and even though there was another guitarist (an Asian guy), he ended up playing most of the guitar parts, which was really impressive. The biggest surprise though is that I actually liked the opening bands. …I especially liked the first band

# Annotator: High level organization


BandReview Join
"Lead singer/guitarist was really good, and even … I actually liked the opening bands. … Well they were none of those. I especially liked the first band"
ReviewGroup Aggregator
ConcertInstance Extractor
"went to the Switchfoot concert at the Roxy"
"went to AJCO n Band concert"
"performance by local funk band Saaraba"
"lead singer/guitarist was really good"
"Liked the opening bands"
"Liked the first band"
"Kurt Ralske played guitar"
"put on a great show"
ReviewInstance Extractor

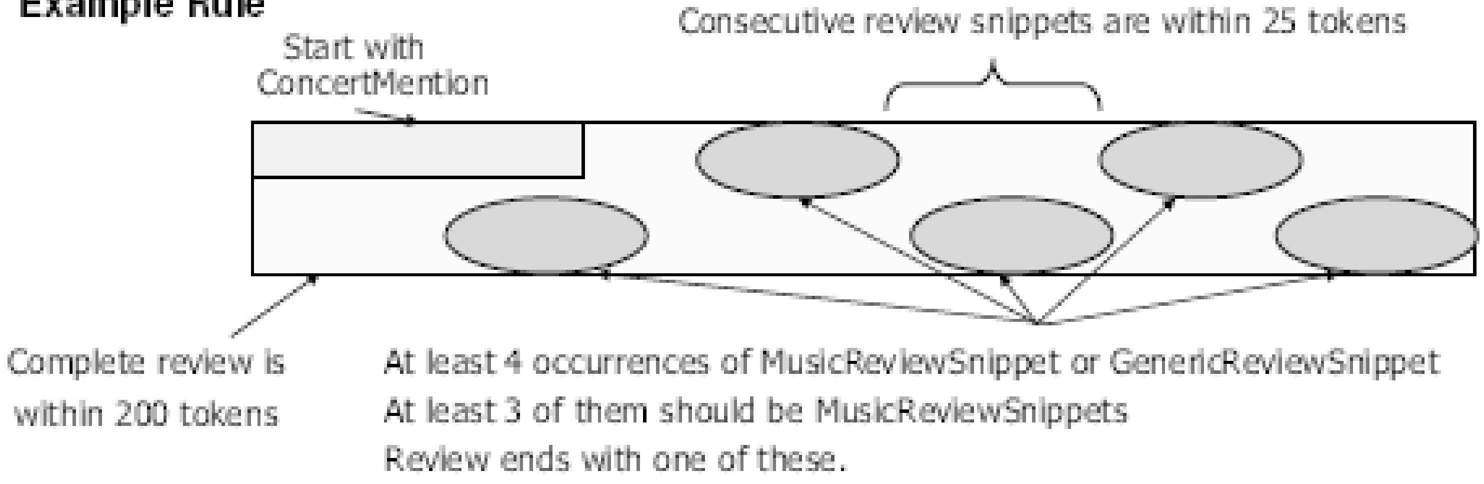
# An example rule

**Informal Band Review**



**went to the Switchfoot concert at the Roxy**. It was pretty fun,... **The lead singer/guitarist was really good**, and even though there was another guitarist (an Asian guy), he ended up playing most of the guitar parts, which was really impressive. The biggest surprise though is that I actually **liked the opening bands**. ...I especially **liked the first band**

**Example Rule**

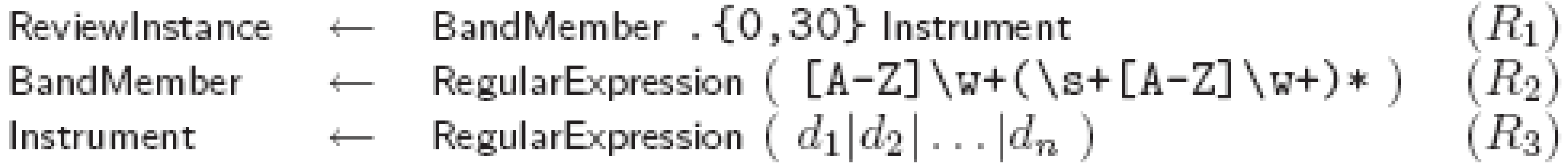

Start with ConcertMention
Consecutive review snippets are within 25 tokens
Complete review is within 200 tokens

At least 4 occurrences of MusicReviewSnippet or GenericReviewSnippet
At least 3 of them should be MusicReviewSnippets
Review ends with one of these.

# Cascading Grammar

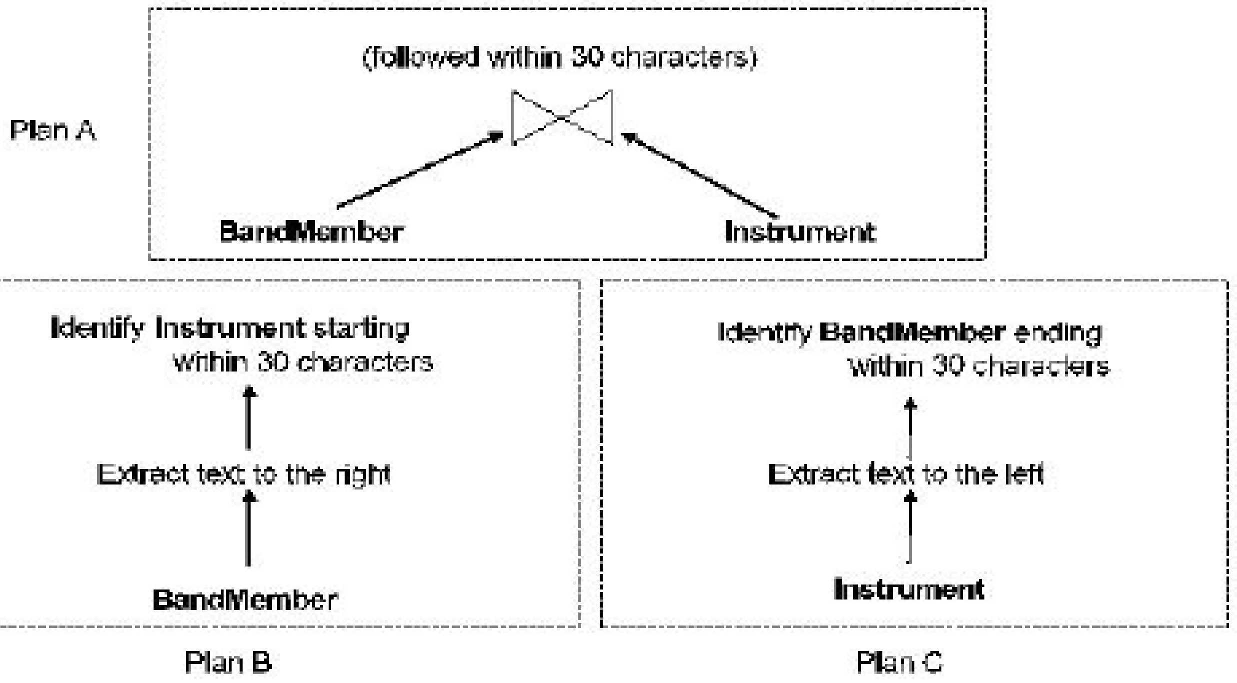
- A rule for identifying *ReviewInstance*
- In words:

***“BandMember followed within 30 characters by Instrument”***



| | | | |
|---|---|---|---|
| ReviewInstance | $\longleftarrow$ | BandMember `.{0,30}` Instrument | $(R_1)$ |
| BandMember | $\longleftarrow$ | RegularExpression ( `[A-Z]\w+(\s+[A-Z]\w+)*` ) | $(R_2)$ |
| Instrument | $\longleftarrow$ | RegularExpression ( $d_1 \| d_2 \| \ldots \| d_n$ ) | $(R_3)$ |

# Shortcomings [1/2]



(followed within 30 characters)
Plan A
BandMember
Instrument
Identify Instrument starting within 30 characters
Extract text to the right
BandMember
Plan B
Identify BandMember ending within 30 characters
Extract text to the left
Instrument
Plan C

- **Overlapping annotations**
- **Multiple execution plans**

## Shortcomings [2/2]

- More drawbacks in expressivity
  - Lossy sequencing
  - Rigid matching priority
- Need for new IE paradigm
  - Expressivity
  - Scalability

# Outline

- Motivation
- **SystemText background**
- AQL

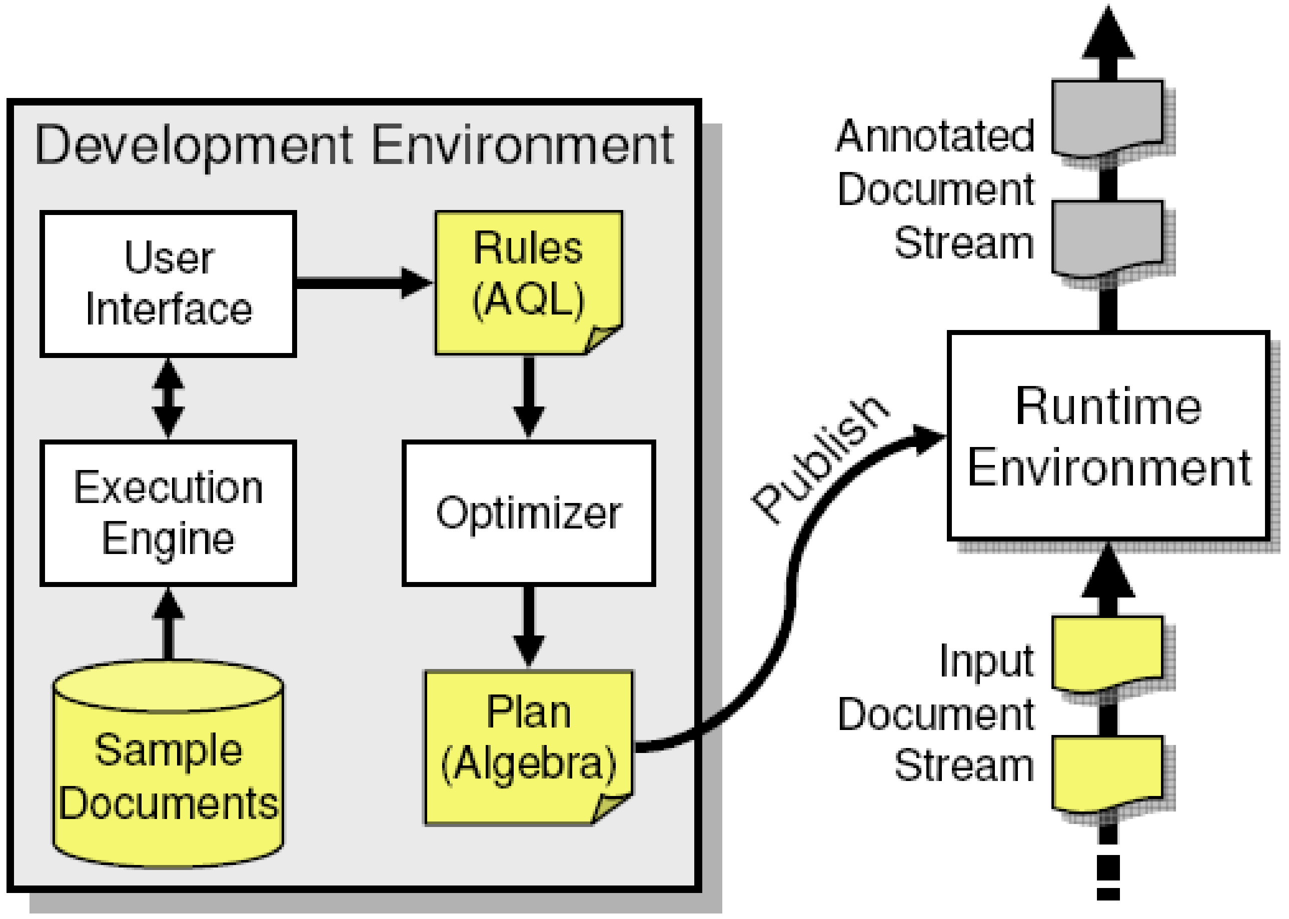
# SystemText

- Rule-based Information Extraction system
  - Rules expressed in SQL-like declarative language – Annotation Query Language (AQL)
  - Annotation engine architecture is inspired from database systems
- Overcomes the shortcomings of traditional IE
  - Expressivity: declarative query language
  - Scalability: query optimization from databases

# Algebra and Data model

- Simple relational data model
  - **Span**: <begin, end>
  - **Tuple**: $<s_1,s_2 \ldots s_m>$ ; **m** is called the **width** of tuple
  - **Relation**: same width tuples
- Operators
  - **Relational** – select, project, join, minus
  - **Span Extraction** – dictionary matcher, regex matcher
  - **Span Aggregation** – containment and overlap consolidation

# SystemT Architecture



# Optimizations in SystemT [1/2]

- Conditions that govern the design of optimization strategies
  - Processing one document at a time
  - Spans are at the centre of the system and obey the conditions of interval algebra
  - Span extraction operators (regex and dict match) are most CPU intensive operations and have to be optimized
- Optimization strategies try to exploit the first two conditions to reduce / avoid span extraction operations

# Optimizations in SystemT [2/2]

- Rule rewriting strategies
  - Regex strength reduction
  - Shared Dictionary Matching
- Cost-based optimization
  - Conditional Evaluation
  - Restricted Span Evaluation

# Outline

- Motivation
- SystemText background
- **AQL**

## AQL: high level overview

- **View**
- Basic AQL constructs
  - Extract statement (dictionary , regex matcher)
  - Select statement (select, project, join)
- Built-in functions
  - Scalar functions
  - Predicate functions
  - ….

# Views

- Basic building block of AQL queries
- Types
  - Output view
  - Non-output view
- Create view statement
  - Single select / extract statement
  - Multiple select / extract statements: combined
    - union all
    - minus

# Extract statement [1/3]

- Specific to Information extraction
- Basic character level extraction primitives
- Types
  - Regular expression extraction
  - Dictionary extraction
  - Block extraction
  - …..

## Extract statement [2/3]

- Examples

```
extract
    E.sender as emailsender,
    regex /\d{3}-\d{3}-\d{4}/ on E.body as num
from Email E
having MatchesRegex(/.*@enron.com/, emailsender);
```

```
extract
     dictionaries 'first.dict'  and 'last.dict'
      with flags 'Exact‘ on D.text as name
from Document D;
```

# Extract Statement [3/3]

- Block extract specification

```
blocks
    with count [between <min> and] <max>
    and separation [between <min> and] <max> (tokens| characters)
    on <column> as <output name>
```

- Block extract example

```
create view TwoToThreeCapitalizedWords as
extract blocks
    with count between 2 and 3
    and separation between 0 and 100 characters
    on CW.word as capswords
from CapitalizedWord CW;
```

## Select statement [1/3]

- Similar to SQL select statement
- Provides mechanism for constructing complex patterns out of simpler building blocks
- Syntax

```
select <select list>
from <from list>
[where <where clause>]
[consolidate on <column> [using '<policy>']]
[group by <group by list>]
[order by <order by list>]
[limit <maximum number of output tuples for each document>];
```

## Select statement [2/3]

- Select list
  - Comma-separated list of output columns
  - Select * → similar to SQL
  - Can involve scalar functions
- From list
  - List of input views or nested AQL statements

```
select *
from
   (extract dictionary 'first.dict' on D.text as name from Document D) as FN,
   LastName as "Last Name"
```

## Select statement [3/3]

- Where clause
  - Defined over the cross-product of input relations
  - Conjunction of predicate functions
- Consolidate clause
  - For handling spans that overlap
  - ***consolidate on <target> [using <policy>]***
  - ***consolidate on P.name using ContainedWithin***

# Built-in Functions

- Types
  - **Predicate functions**
  - **Scalar functions**
  - Aggregate functions
  - Table functions
- Many functions are specific to information extraction

# Predicate Functions

- Predicates – used in the where clause
- *Contains, ContainsDict, ContainsRegex*
- *MatchesDict, MatchesRegex*
- *Follows, FollowsTok*
- *Overlaps*
- *Or, Not, And, Equals*

## Scalar functions

- Scalar – used in select list or input to predicates
- *CombineSpans*
- *SpanBetween*
- *GetText*
- *SpanIntersection*
- *LeftContext, LeftContextTok, RightContext, RightContextTok*

# Complicated rules

Detects all false overlaps to eliminate

Unintended match

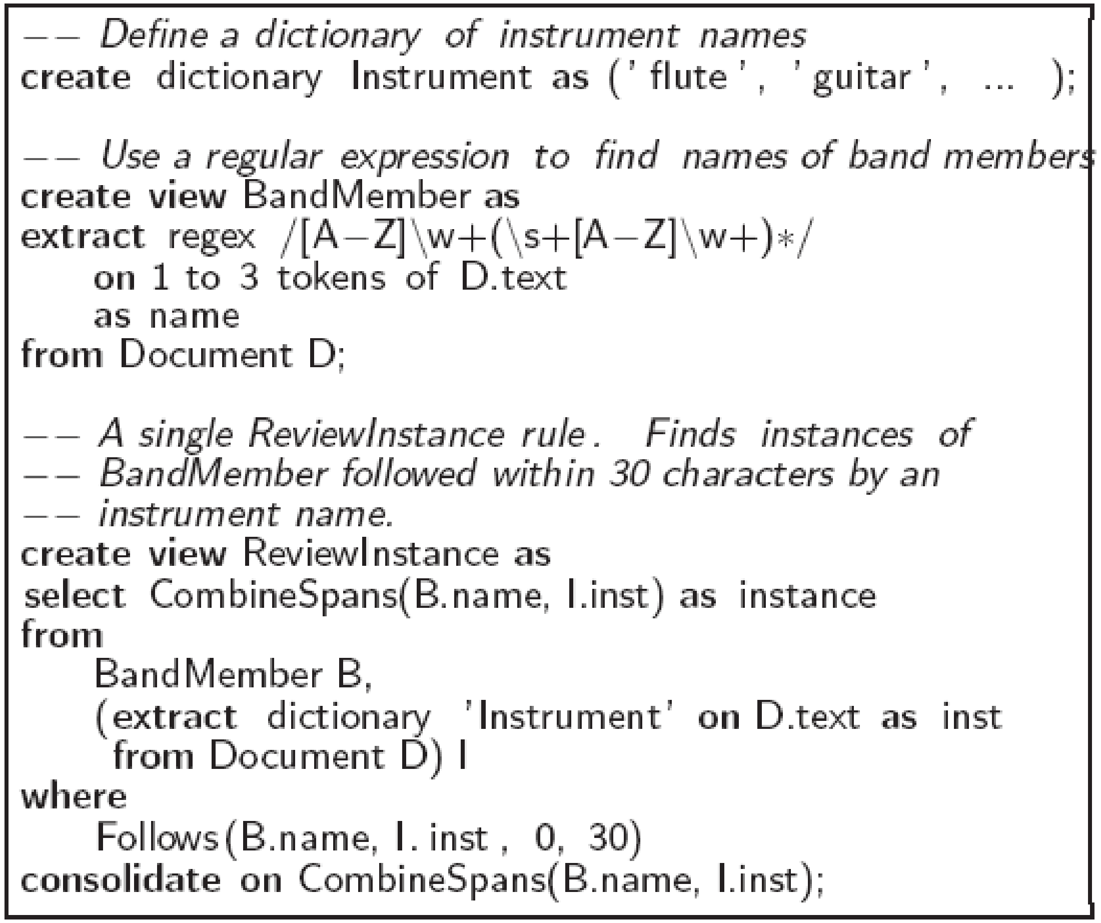
- Filteri
  - E.g: “… *Sachin Tendulkar, Vijay Hazare*, …. “

```
create view LastCommaFirstToDelete as
select        LCF.name as name
from          FirstLast FL, LastCommaFirst LCF
where         Overlaps(LCF.name, FL.name);

create view LastCommaFirstValid as
(select R.name as name from LastCommaFirst R)
minus
(select R.name as name
 from LastCommaFirstToDelete R);
```

Filter operation

# *ReviewInstance* example

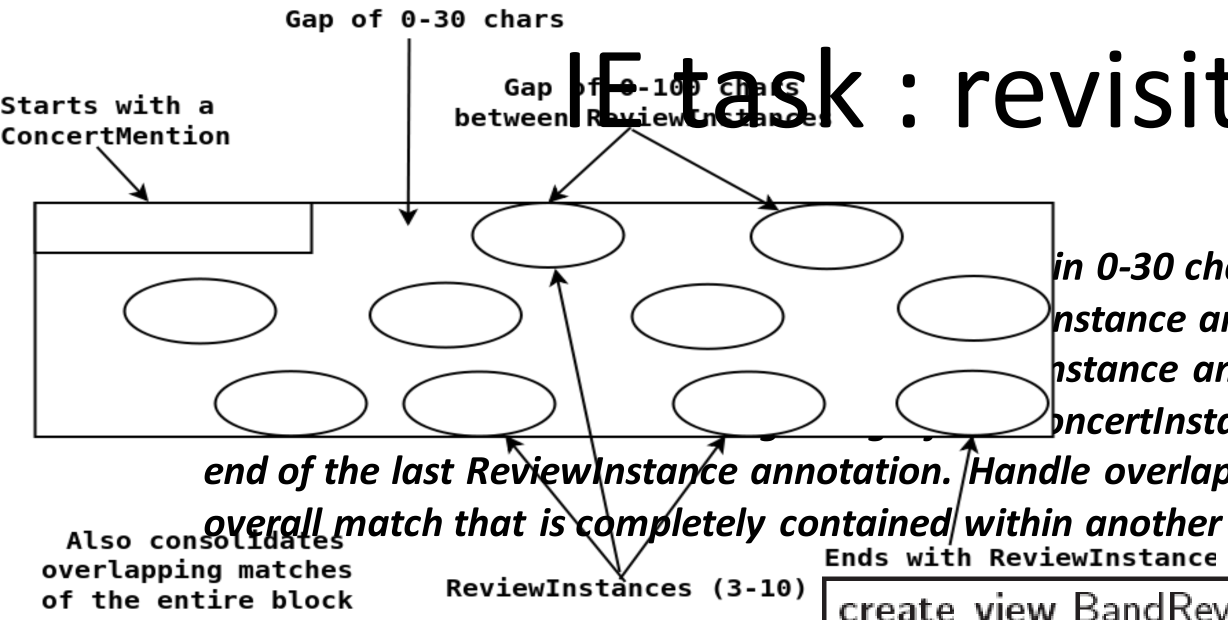
***"BandMember followed within 30 characters by Instrument"***



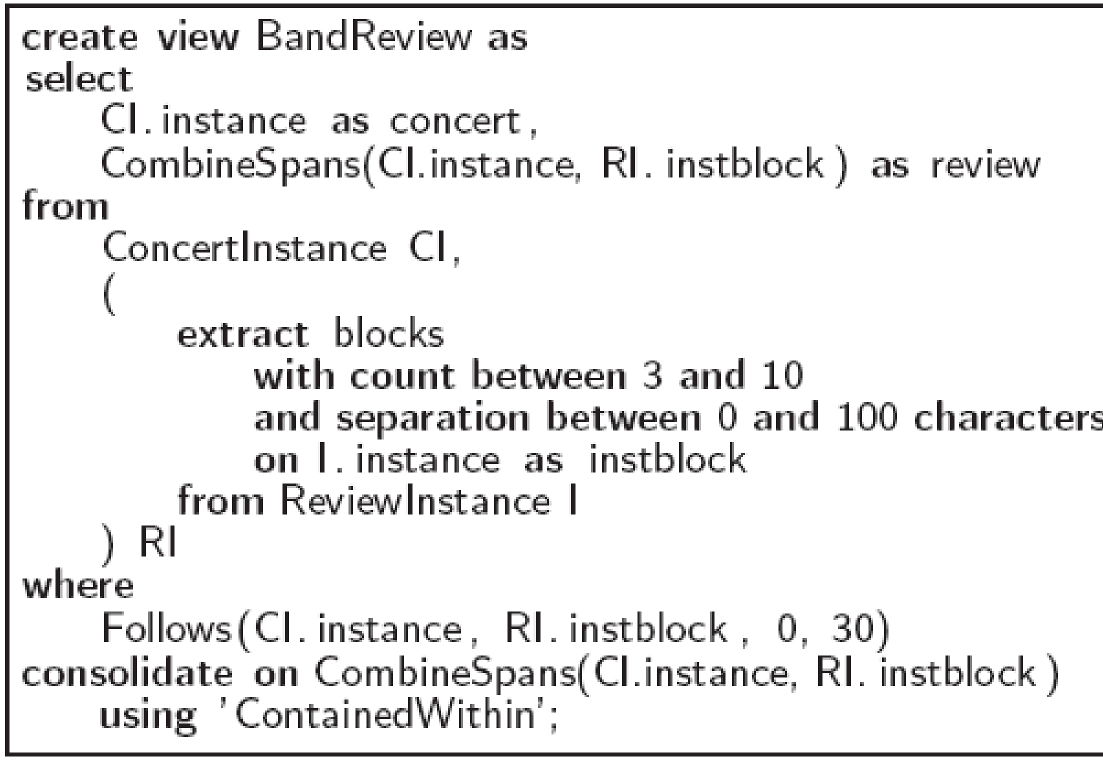
```
-- Define a dictionary of instrument names
create dictionary Instrument as ('flute', 'guitar', ... );

-- 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);
```

# IE task : revisited

*…in 0-30 characters by a block of 3 to 10 … nstance annotations must be within 100 … nstance annotation, create a new output … ncertInstance annotation and runs to the end of the last ReviewInstance annotation. Handle overlapping matches by removing any overall match that is completely contained within another match.”*



Gap of 0-30 chars
Starts with a ConcertMention
Gap of 0-100 chars between ReviewInstances
Also consolidates overlapping matches of the entire block
ReviewInstances (3-10)
Ends with ReviewInstance



```
create view BandReview as
select
    CI.instance as concert,
    CombineSpans(CI.instance, RI.instblock) as review
from
    ConcertInstance CI,
    (
        extract blocks
            with count between 3 and 10
            and separation between 0 and 100 characters
            on I.instance as instblock
        from ReviewInstance I
    ) RI
where
    Follows(CI.instance, RI.instblock, 0, 30)
consolidate on CombineSpans(CI.instance, RI.instblock)
    using 'ContainedWithin';
```

# Summary

- Need for a new IE paradigm
  - **Expressivity**
  - Scalability
- SystemT
  - Architecture
  - Optimizations
- AQL
  - Main constructs
  - Examples of non-trivial rules

# References

1. Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. **Systemt: An algebraic approach to declarative information extraction**. In ACL , July 2010.
2. Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. **An algebraic approach to rule-based information extraction**. In ICDE, 2008.
3. Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan,Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu .**SystemT: A System for Declarative Information Extraction.** In SIGMOD, 2008.
4. Laura Chiticariu Rajasekar Krishnamurthy Yunyao Li Frederick Reiss Shivakumar Vaithyanathan, **Domain Adaptation of Rule-Based Annotators for Named-Entity Recognition Tasks**, in EMNLP 2010.
5. SystemT. 2010, AQL manual. **http://www.alphaworks.ibm.com/tech/systemt**