Classifying $\mathcal{ELH}$ Ontologies in SQL Databases

Vincent Delaitre and Yevgeny Kazakov
(Presented by Rob Shearer)

Oxford University Computing Laboratory

October 24, 2009

Outline

1 Introduction

2 Procedure Outline

3 Problems and Solutions

4 Results

$\mathcal{ELH}$ and $\mathcal{OWL}$ 2 EL

OWL 2 Syntax	DL Syntax
Class expressions:	
ObjectIntersectionOf$(C\ D)$	$C \sqcap D$
ObjectSomeValuesFrom$(r\ C)$	$\exists r.C$
Axioms:	
SubClassOf$(C\ D)$	$C \sqsubseteq D$
EquivalentClasses$(C\ D)$	$C \equiv D$
SubObjectPropertyOf$(r\ s)$	$r \sqsubseteq s$

- $\mathcal{ELH}$ is a simple sub-fragment of $\mathcal{OWL}$ 2 EL
- Has a very simple polynomial-time classification procedure [Baader et al.,IJCAI 2003,2005]
- Sufficiently expressive for many ontologies such as SNOMED, FMA, NCI, GO and large part of GALEN
- Has a potential of scaling to even larger ontologies

Are we Ready for Ontologies with Millions of Classes?

- SNOMED CT contains over 300,000 classes—probably the largest ontology available so far
- Can be classified in minutes using many existing reasoners:

	Time
CEL	21min.42s.
FaCT++	16min.05s.
RACER	19min.30s.
SNOROCKET	1min.06s.
CB	0min.45s.

Are we Ready for Ontologies with Millions of Classes?

- SNOMED CT contains over 300,000 classes—probably the largest ontology available so far
- Can be classified in minutes using many existing reasoners:

	Time	Memory
CEL	21min.42s.	700MB*
FaCT++	16min.05s.	320MB*
RACER	19min.30s.	900MB
SNOROCKET	1min.06s.	2GB
CB	0min.45s.	400MB

- But memory consumption could be a problem for ontologies 10x larger.

Secondary Memory Onotlogy Reasoning

- The main idea: use a DBMS for processing of ontologies
- Advantages:
  1. Low main memory footprint
  2. Persistence: can save / restore computations
  3. Transactions and fault tolerance
  4. Possible to adapt to multi-user environments
- Disadvantage:
  1. Slow (because of the secondary memory characteristics)

Secondary Memory Onotlogy Reasoning

- The main idea: use a DBMS for processing of ontologies
- Advantages:
  1. Low main memory footprint
  2. Persistence: can save / restore computations
  3. Transactions and fault tolerance
  4. Possible to adapt to multi-user environments
- Disadvantage:
  1. Slow (because of the secondary memory characteristics)
- Our main results:
  - It is possible to classify $\mathcal{ELH}$ ontologies in SQL databases
  - Naive approach has poor performance
  - Optimizations (caching) improve performance significantly
  - Able to classify SNOMED CT in 20min using 32MB of RAM.

(UN)RELATED WORKS

- Conjunctive query answering in $\mathcal{EL}$ using relational databases [Lutz, Toman, Wolter,IJCAI 2009]
  - large instance data
  - medium-size schema (60,000 classes)
  - main focus is on query response
- "DB-backed" module in the IBM SHER system
  - Uses a Datalog engine
  - Presumably can work with $\mathcal{EL}^+$ ontologies
  - Cannot classify SNOMED CT(?)
- RDF databases:
  - Can query large triple stores
  - Can use custom rules
  - Cannot classify OWL ontologies(?)

Outline

1 Introduction

2 Procedure Outline

3 Problems and Solutions

4 Results

$\mathcal{ELH}$ Classification Procedure

1. Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

$\mathcal{ELH}$ Classification Procedure

1 Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow$

$\mathcal{ELH}$ Classification Procedure

1 Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.\underline{D} \quad \underline{D} \sqsubseteq B \sqcap C$

$\mathcal{ELH}$ Classification Procedure

1. Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \sqcap C$

$\mathcal{ELH}$ Classification Procedure

1. Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$

$\mathcal{ELH}$ Classification Procedure

1. Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

2. Deriving consequences using the rules [Brandt, ECAI 2004]:

$\mathcal{ELH}$ Classification Procedure

1 Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$

2 Deriving consequences using the rules [Brandt, ECAI 2004]:

IR1 $\dfrac{}{A \sqsubseteq A}$ **IR2** $\dfrac{}{A \sqsubseteq \top}$ (tautologies)

$\mathcal{ELH}$ Classification Procedure

1. Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

2. Deriving consequences using the rules [Brandt, ECAI 2004]:

IR1 $\dfrac{}{A \sqsubseteq A}$ **IR2** $\dfrac{}{A \sqsubseteq \top}$ (tautologies)

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$ **CR2** $\dfrac{A \sqsubseteq B \quad A \sqsubseteq C}{A \sqsubseteq D} : B \sqcap C \sqsubseteq D \in \mathcal{O}$

$\mathcal{ELH}$ Classification Procedure

1 Normalization to simple axioms of five forms:

(1) $A \sqsubseteq B$ (2) $A \sqcap B \sqsubseteq C$ (3) $A \sqsubseteq \exists r.B$ (4) $\exists r.B \sqsubseteq C$ (5) $r \sqsubseteq s$

Example

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \rightsquigarrow \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

2 Deriving consequences using the rules [Brandt, ECAI 2004]:

IR1 $\dfrac{}{A \sqsubseteq A}$ **IR2** $\dfrac{}{A \sqsubseteq \top}$ (tautologies)

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$ **CR2** $\dfrac{A \sqsubseteq B \quad A \sqsubseteq C}{A \sqsubseteq D} : B \sqcap C \sqsubseteq D \in \mathcal{O}$

CR3 $\dfrac{A \sqsubseteq B}{A \sqsubseteq \exists r.C} : B \sqsubseteq \exists r.C \in \mathcal{O}$ **CR4** $\dfrac{A \sqsubseteq \exists r.B}{A \sqsubseteq \exists s.B} : r \sqsubseteq s \in \mathcal{O}$

CR5 $\dfrac{A \sqsubseteq \exists r.B \quad B \sqsubseteq C}{A \sqsubseteq D} : \exists r.C \sqsubseteq D \in \mathcal{O}$

Database Organization

Example

$Heart \sqsubseteq MuscularOrgan$ (type 1)

$Heart \sqsubseteq \exists isPartOf.CirculatorySystem$ (type 3)

- Use two tables to assign ids to classes and object properties

class	id
Heart	1
MuscularOrgan	2
CirculatorySystem	3

object property	id
isPartOf	1

- Use five tables to store normalized axioms of each type

ax_t1

A	$\sqsubseteq$	B
1		2

ax_t2

A	$\sqcap B$	$\sqsubseteq C$
.	.	.

ax_t3

A	$\sqsubseteq \exists r$	$.B$
1	1	3

ax_t4

$\exists r$	$.B$	$\sqsubseteq C$
.	.	.

ax_t5

r	$\sqsubseteq s$
.	.

COMPLETION USING SQL QUERIES

ax_t1

$A \sqsubseteq B$	
1	2

ax_t2

$A \sqcap B \sqsubseteq C$		
2	5	1

ax_t3

$A \sqsubseteq \exists r.B$		
1	1	3

ax_t4

$\exists r.B \sqsubseteq C$		
3	4	5

ax_t5

$r \sqsubseteq s$	
1	2

- Use two tables to output the results of inferences:

s_t1

$A \sqsubseteq B$	
1	1

s_t2

$A \sqsubseteq \exists r.B$		
1	1	3

- Use SQL commands to perform inferences:

IR1 $\dfrac{}{A \sqsubseteq A}$

```
INSERT INTO s_t1
SELECT class.id, class.id;
```

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

Outline

1 Introduction

2 Procedure Outline

3 Problems and Solutions

4 Results

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

class	id

object property	id

$A \sqsubseteq B$

$A \sqcap B \sqsubseteq C$

$A \sqsubseteq \exists r.B$

$\exists r.B \sqsubseteq C$

$r \sqsubseteq s$

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

class	id
Heart	1

object property	id

$A \sqsubseteq B$	$A \sqcap B \sqsubseteq C$	$A \sqsubseteq \exists r.B$	$\exists r.B \sqsubseteq C$	$r \sqsubseteq s$
1				

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqcap B \sqsubseteq C$

$A \sqsubseteq \exists r.B$

$\exists r.B \sqsubseteq C$

$r \sqsubseteq s$

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqcap B \sqsubseteq C$

$A \sqsubseteq \exists r.B$
1

$\exists r.B \sqsubseteq C$

$r \sqsubseteq s$

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ ∃isPartOf.CirculatorySystem

class	id
Heart	1
MuscularOrgan	2

object property	id
isPartOf	1

$A \sqsubseteq B$	
1	2

$A \sqcap B \sqsubseteq C$

$A \sqsubseteq \exists r.B$	
1	1

$\exists r.B \sqsubseteq C$

$r \sqsubseteq s$

Problem 1: Assignment of the IDs

Example

Heart ⊑ MuscularOrgan   Heart ⊑ ∃isPartOf.CirculatorySystem

class	id
Heart	1
MuscularOrgan	2
CirculatorySystem	3

object property	id
isPartOf	1

$A \sqsubseteq B$	
1	2

$A \sqcap B \sqsubseteq C$		

$A \sqsubseteq \exists r.B$		
1	1	3

$\exists r.B \sqsubseteq C$		

$r \sqsubseteq s$	

Problem 1: Assignment of the IDs

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

class	id
Heart	1
MuscularOrgan	2
CirculatorySystem	3

object property	id
isPartOf	1

$A \sqsubseteq B$	
1	2

$A \sqcap B \sqsubseteq C$

$A \sqsubseteq \exists r.B$		
1	1	3

$\exists r.B \sqsubseteq C$

$r \sqsubseteq s$

- On-disc table lookup is too slow!
- Making a query for every occurrence of a class is impractical due to overheads (connection + parsing + transaction)

SOLUTION: IN-MEMORY CACHING

EXAMPLE

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

On-Disk

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqsubseteq \exists r.B$

In-Memory

class	id

object property	id

$A \sqsubseteq B$

$A \sqsubseteq \exists r.B$

Solution: In-memory Caching

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

On-Disk

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqsubseteq \exists r.B$

In-Memory

class	id
Heart	3

object property	id

$A \sqsubseteq B$

$A \sqsubseteq \exists r.B$
3

Solution: In-memory Caching

Example

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

On-Disk

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqsubseteq \exists r.B$	

In-Memory

class	id
Heart	3

object property	id
isPartOf	4

$A \sqsubseteq B$	

$A \sqsubseteq \exists r.B$	
3	4

SOLUTION: IN-MEMORY CACHING

EXAMPLE

Heart $\sqsubseteq$ MuscularOrgan   Heart $\sqsubseteq$ $\exists$isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

On-Disk

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqsubseteq \exists r.B$		

In-Memory

class	id
Heart	3
CirculatorySystem	5

object property	id
isPartOf	4

$A \sqsubseteq B$	

$A \sqsubseteq \exists r.B$		
3	4	5

SOLUTION: IN-MEMORY CACHING

EXAMPLE

Heart ⊑ MuscularOrgan   Heart ⊑ ∃isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids
- Resolve uniqueness of ids using SQL quieries when the tables are large enough

On-Disk

class	id
Heart	1
MuscularOrgan	2

object property	id

$A \sqsubseteq B$	
1	2

$A \sqsubseteq \exists r.B$		

In-Memory

class	id
Heart	3
CirculatorySystem	5

object property	id
isPartOf	4

$A \sqsubseteq B$

$A \sqsubseteq \exists r.B$		
3	4	5

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3

ax_t1

$A \sqsubseteq B$	
1	2
1	4
2	3

- Repeated application of joins are necessary to compute the closure

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3
1	2
1	4
2	3

ax_t1

$A \sqsubseteq B$	
1	2
1	4
2	3

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure

s_t1

A	$\sqsubseteq B$
1	1
2	2
3	3
1	2
1	4
2	3
1	3

ax_t1

A	$\sqsubseteq B$
1	2
1	4
2	3

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3

ax_t1

$A \sqsubseteq B$	
1	2
1	4
2	3

tmp

$A \sqsubseteq B$	
1	1

PROBLEM 2: SLOW JOINS

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3

ax_t1

$A \sqsubseteq B$	
1	2
1	4
2	3

tmp

$A \sqsubseteq B$	
1	1
1	2
1	4

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory

s_t1

A	$\sqsubseteq B$
1	1
2	2
3	3

ax_t1

A	$\sqsubseteq B$
1	2
1	4
2	3

tmp

A	$\sqsubseteq B$
1	1
1	2
1	4
1	3

Problem 2: Slow Joins

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory
- And output the result into the main table

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3
1	2
1	4
1	3

ax_t1

$A \sqsubseteq B$	
1	2
1	4
2	3

PROBLEM 2: SLOW JOINS

CR1 $\dfrac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

s_t1

A	B
1	1
2	2
3	3
1	2
1	4
1	3

ax_t1

A	B
1	2
1	4
2	3

tmp

A	B
2	2

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory
- And output the result into the main table
- Repeat similarly for the other parts

Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitively reduced

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3
1	2
2	3
1	3

Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitively reduced
- Can be done using one self join and marking the result as non-direct subsumptions.

s_t1

$A \sqsubseteq B$		
1	1	×
2	2	×
3	3	×
1	2	
2	3	
1	3	×

Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitively reduced
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the the number of non-direct subsumptions is large

s_t1

A	$\sqsubseteq B$	
1	1	×
2	2	×
3	3	×
1	2	
2	3	
1	3	×

PROBLEM 3: TRANSITIVE REDUCTION

- To produce the taxonomy, the output table needs to be transitively reduced
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the number of non-direct subsumptions is large
- Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk

s_t1

A	$\sqsubseteq B$
1	1
2	2
3	3
1	2
2	3
1	3

tmp

A	$\sqsubseteq B$
1	1
1	2
1	3

Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitively reduced
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the number of non-direct subsumptions is large
- Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk

s_t1

$A \sqsubseteq B$	
1	1
2	2
3	3
1	2
2	3
1	3

tmp

$A \sqsubseteq B$		
1	1	×
1	2	
1	3	×

PROBLEM 3: TRANSITIVE REDUCTION

- To produce the taxonomy, the output table needs to be transitively reduced
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the number of non-direct subsumptions is large
- Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk

s_t1

A	$\sqsubseteq$ B	
1	1	
2	2	
3	3	
1	2	○
2	3	
1	3	

tmp

A	$\sqsubseteq$ B	
1	1	×
1	2	○
1	3	×

Outline

1 Introduction

2 Procedure Outline

3 Problems and Solutions

4 Results

Timings for Different Stages (time in seconds)

Action	NCI	GO	Galen$^-$	Snomed
Loading/Preprocessing	17.85	5.99	23.41	127.51
Completion	5.78	7.29	53.13	783.30
Transitive reduction	10.32	6.10	21.44	249.23
Formating output	1.56	0.98	2.88	23.76
Total	35.51	20.36	100.86	1183.80

- NCI (`www.cancer.gov`) contains 27,652 classes
- GO (`www.geneontology.org`) contains 20,465 classes
- Galen$^-$ (`www.co-ode.org/galen`) contains 23,136 classes (functionality, inverses, and transitivity removed)
- Snomed (`www.ihtsdo.org`) contains 315,489 classes

available at: `db-reasoner.googlecode.com`

Comparison with In-Memory Reasoners (time in seconds)

Reasoner	NCI	GO	Galen^{-}	Snomed
CB	7.64	1.23	3.36	45.17
CEL v.1.0	3.60	1.02	169.23	1302.18
FaCT++ v.1.3.0	4.60	10.50	—	965.84
HermiT v.0.9.3	70.23	92.76	—	—
DB	35.51	20.36	100.86	1183.80

- CB (cb-reasoner.googlecode.com)
- CEL (lat.inf.tu-dresden.de/systems/cel/)
- FaCT++ (owl.man.ac.uk/factplusplus)
- HermiT (hermit-reasoner.com)

Conclusions

- $\mathcal{ELH}$ classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners

Conclusions

- $\mathcal{ELH}$ classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
    - Snomed x 1 = 20min
    - Snomed x 5 = 4h.30min
    - Snomed x 10 = did not finish overnight

Conclusions

- $\mathcal{ELH}$ classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
  - Snomed x 1 = 20min
  - Snomed x 5 = 4h.30min
  - Snomed x 10 = did not finish overnight

Future work

- Extension to $\mathcal{OWL}$ **2 EL**
- Tuning the DB engine / testing on a real DB server

Conclusions

- $\mathcal{ELH}$ classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
  - Snomed x 1 = 20min
  - Snomed x 5 = 4h.30min
  - Snomed x 10 = did not finish overnight

Future work

- Extension to $\mathcal{OWL}$ **2 EL**
- Tuning the DB engine / testing on a real DB server

Please be kind and not ask too difficult questions!

or send them to: `yevgeny.kazakov@comlab.ox.ac.uk`