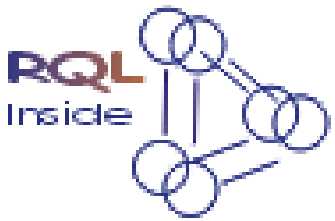
Benchmarking Database Representations of RDF/S Stores

Yannis Theoharis[1], Vassilis Christophides[1],
Grigoris Karvounarakis[2]

[1]Computer Science Department, University of Crete
and Institute of Computer Science – FORTH
Heraklion, Crete, Greece

[2]Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA, USA

RDF/S Repositories



DLDB
(OWL rep.)

PARKA









- But how can we compare RDF/S stores?
- Our approach:

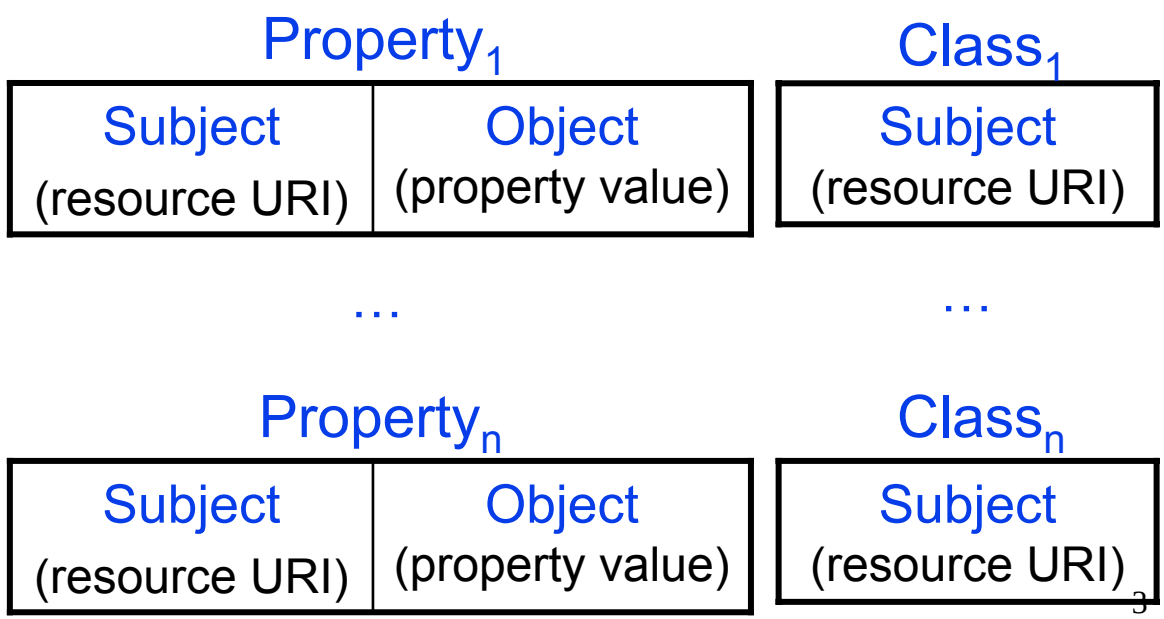
  Benchmark their internal storage schemes

Main Database Representations

- Schema-oblivious: One database schema for all RDFS schemata
  - One ternary relation for any RDF/S schema or resource description graph



- Schema-aware: One database schema for each RDFS schema
  - One binary (unary) relation per RDF/S schema property (class)



Main Database Representations

- Hybrid: one database schema for all RDFS schemata (as schema-oblivious), but distinguishes classes from properties and properties themselves according to their range types (as schema-aware)
  - A ternary relation for every different property range type and a binary relation for all class instances (as in schema-aware)
  - Property (class) instances with range values of the same type are stored in the same relation, distinguished by the property (class) id (as in schema-oblivious)

Properties with range Resource

Subject (resource URI)	Predicate (property name)	Object (property value)

…

Properties with range integer

Subject (resource URI)	Predicate (property name)	Object (property value)

Class Instances

Subject (resource URI)	Object (classid)

Main Database Representations Comparison

- Schema-oblivious:
  - straightforward schema evolution, but
  - disregards type information
- Schema-aware:
  - preserves type information, but
  - difficult schema-evolution and
  - significant overhead for large number of tables
- Hybrid:
  - easy schema evolution, while
  - preserving type information
- But how can we evaluate their query performances???

Outline

- We benchmark various alternatives
  - Schema-oblivious: URI, ID
  - Schema-aware: ISA, NOISA, MatView
  - Hybrid
- We focus on the evaluation of taxonomic queries.
  Inferred triples:
  - On-the-fly Transitive Closure Computation
  - Precomputation of Transitive Closure
- Synthetic RDF/S Data generator
  - schemata of different size
  - various distribution modes for populating classes
  - queries at different levels of a subsumption hierarchy
- Conclusions and future work

Taxonomic Query Evaluation

- The issue: How to compute the Transitive Closure of resource descriptions by taking into account class/property subsumption (RDF/S Semantics)
  - Taxonomic queries: explicit triples + inferred triples

❶ Precomputation and materialization:
- avoids to recompute TCs
- harder update propagation
- significant storage overhead

- Usually employed by stores adopting schema-oblivious representations:
  - URI: stores the URIs in the table holding the triples
  - ID: uses integer identifiers to represent resources and properties
- Also employed by a store following a schema-aware representation:
  - MatView: for each class (or property) a materialized view holds both its proper and transitive instances

Taxonomic Query Evaluation

❷ On the fly computation of inferred triples:

- less storage requirements
- need for a database representation of subsumption relationships

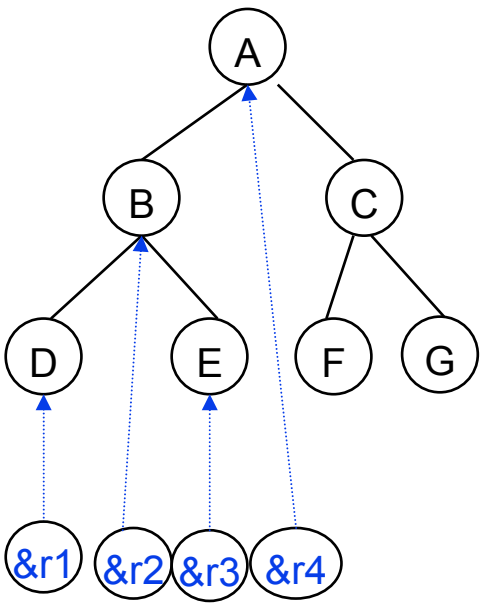
- Employed by stores adopting:
  - Hybrid: employs an interval-based encoding of subsumption relationships for storing together the data with the schema information
  - Schema-aware:
    - NOISA: relies on the same encoding
    - ISA: exploits the object-relational features of SQL99 for representing subsumption relationships using subtable definitions

Existing Semantic Web Stores

RDF/S Stores	Schema-aware			Hybrid	Schema-oblivious	
	ISA	NOISA	MatView (materialized)		URI (materialized)	ID (materialized)
RDFSuite	✓	✓		✓		
Jena					✓	✓
Sesame	✓					✓
DLDB			✓			
RStar						✓
KAON						✓
PARKA		✓				
3Store						✓

SQL Translation of Taxonomic Queries (URI)

- Straightforward taxonomic query evaluation for the stores adopting TC materialization
- URI



Triples

&r4	typeof	A
&r2	typeof	B
&r2	typeof	A
&r1	typeof	D
&r1	typeof	B
&r1	typeof	A
&r3	typeof	E
&r3	typeof	B
&r3	typeof	A
…	…	…

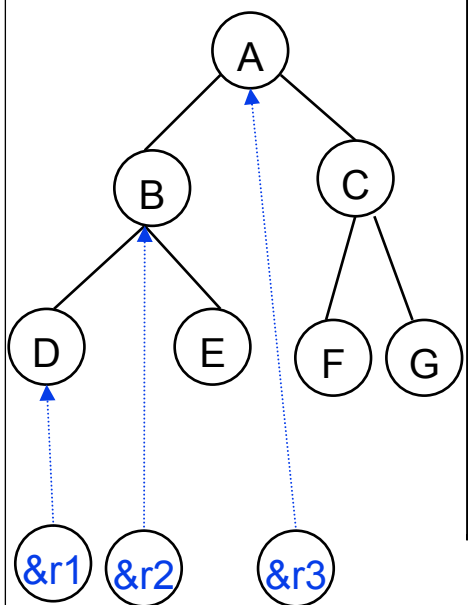
Find transitive instances of ‘’A”:

```
SELECT T.SubjectURI
FROM   Triples T
WHERE  T.predicate = ‘typeof’
       and T.object = ‘A’
```

Selection

SQL Translation of Taxonomic Queries (ID)

- ID



Triples

9	8	1
10	8	2
10	8	1
11	8	4
11	8	2
11	8	1
…	…	…

Instances

A	1
B	2
C	3
D	4
E	5
F	6
G	7
typeof	8
&r3	9
&r2	10
&r1	11

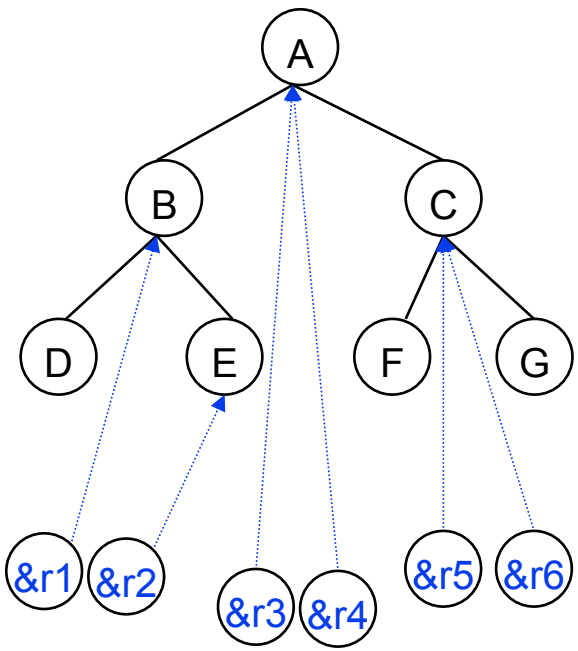
Find transitive instances of ‘’A”:

```
SELECT I.URI
FROM   Triples T, Instance I
WHERE  T.predicate = 8 and
       T.ObjectID = 1 and
       I.ID = T.SubjectID
```

Join

SQL Translation of Taxonomic Queries (MatView)

- MatView



Class A

&r3
&r4

Class B

&r1

Mat_View_A

&r3
&r4
&r1
&r5
&r6
&r2

Mat_View_B

&r1
&r2

Class C

&r5
&r6

Class E

&r2

Find transitive instances of ‘’A”:

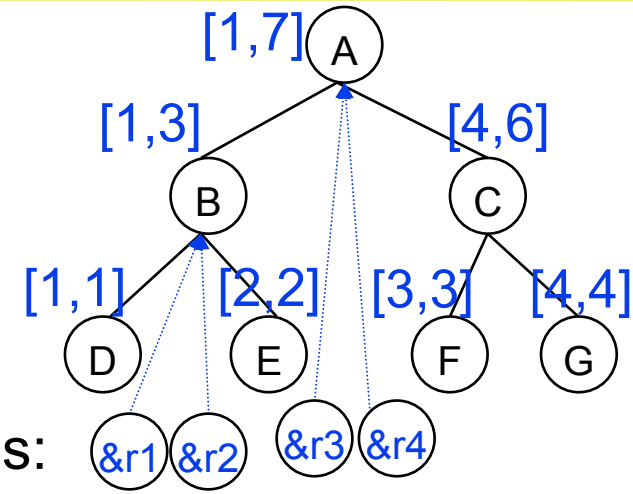
```
SELECT   MV.URI
FROM     Mat_View_A MV
```

Sequential scan

SQL Translation of Taxonomic Queries (NOISA ISA)

- Schema-aware NOISA:
  - (intentional) Find the subclasses of class A:

```
SELECT S.end    FROM    Subclass S
WHERE   S.start ≥ 1 and S.end ≤ 7
```



  - (extensional) **Union** all the corresponding tables:

```
(SELECT URI FROM D) UNION ALL
(SELECT URI FROM E) UNION ALL … UNION ALL
(SELECT URI from A)
```

Union

Subclass

1	7
1	3
1	1
2	2
4	6
4	4
5	5

A

&r3
&r4

B

&r1
&r2

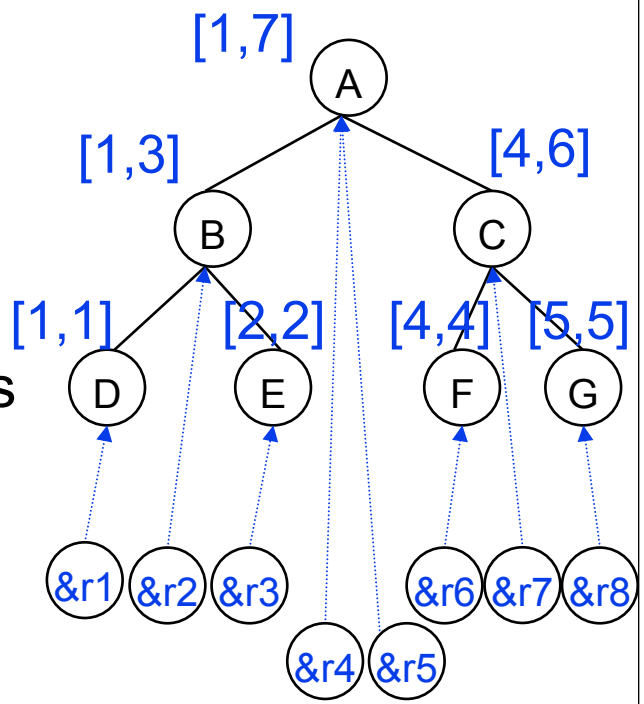
- Schema-aware ISA: Use PostgreSQL inheritance feature

```
select  URI from A
```

Union

SQL Translation of Taxonomic Queries (Hybrid)

- Hybrid: both schema filtering and instance scanning are performed in a single phase



Find transitive instances of ''A":

```
SELECT I.URI
FROM   ClassInstances I
WHERE  I.classid ≥ 1 and
       I.classid ≤ 7
```

Range Query

ClassInstances

&r1	1
&r3	2
&r2	3
&r6	4
&r8	5
&r7	6
&r4	7
&r5	7

Synthetic RDF Data Generation

- (Binary) tree-shaped subsumption hierarchies
- The three critical parameters:
  - The size of the schema (determined by depth $2^{depth+1}-1$)
  - The total number of classified resources
  - Their distribution mode under nodes at various hierarchy levels
- Three categories of schemata:
  - small (up to 4 levels, i.e. 31 nodes)
  - medium (up to 6 levels, i.e. 127 nodes)
  - large (more than 7 levels)



- Three scales of resource bases: 10,000 – 100,000 – 1,000,000

Zipfian Distribution

- Favouring leaves
  - Lower rank values to leaf classes while the Root class has the highest value

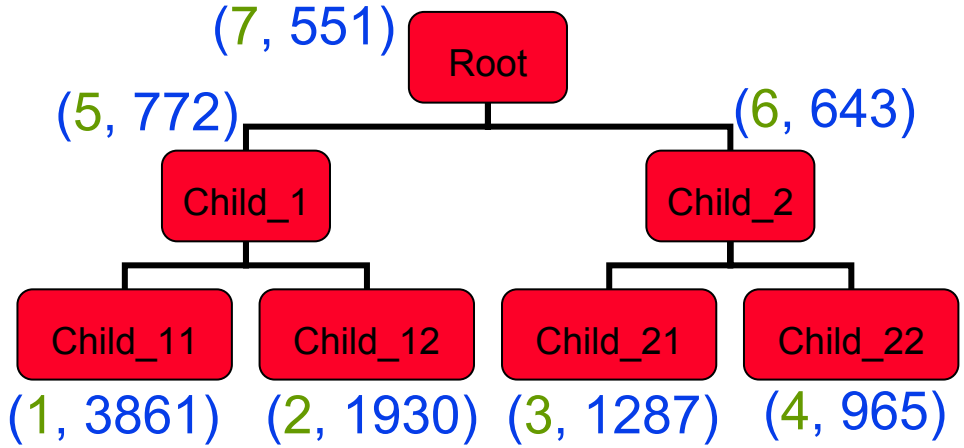
$$Zipf\ (A, i) = \frac{A}{i^z * h}$$

A: # of resources, i: rank value,

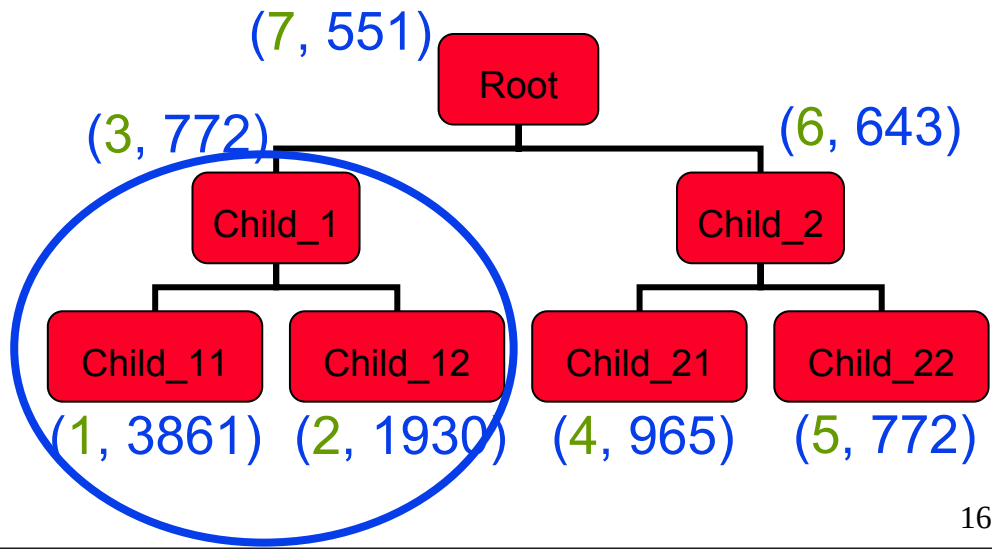
z: skew parameter,

h: normalization factor

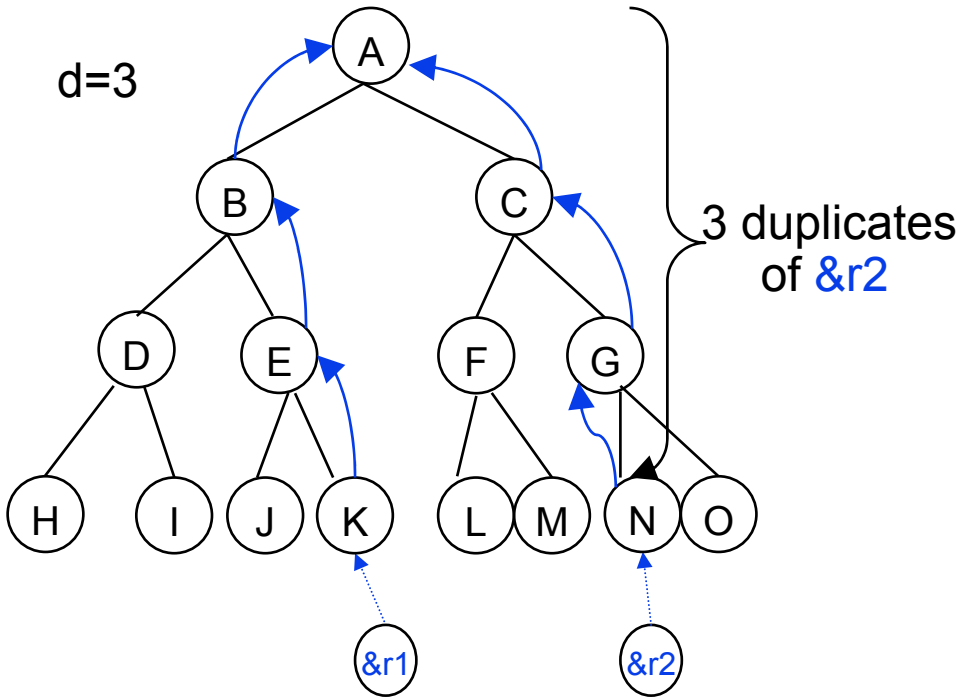
10,000 resources



- Favouring subtrees
  - Lower rank values to the classes of a given subtree beginning from the leaf classes of the selected subtree



Storage Overhead for Materialization



- Complete-binary tree shaped RDFS schema and uniform distribution
  - The total number of triples is:

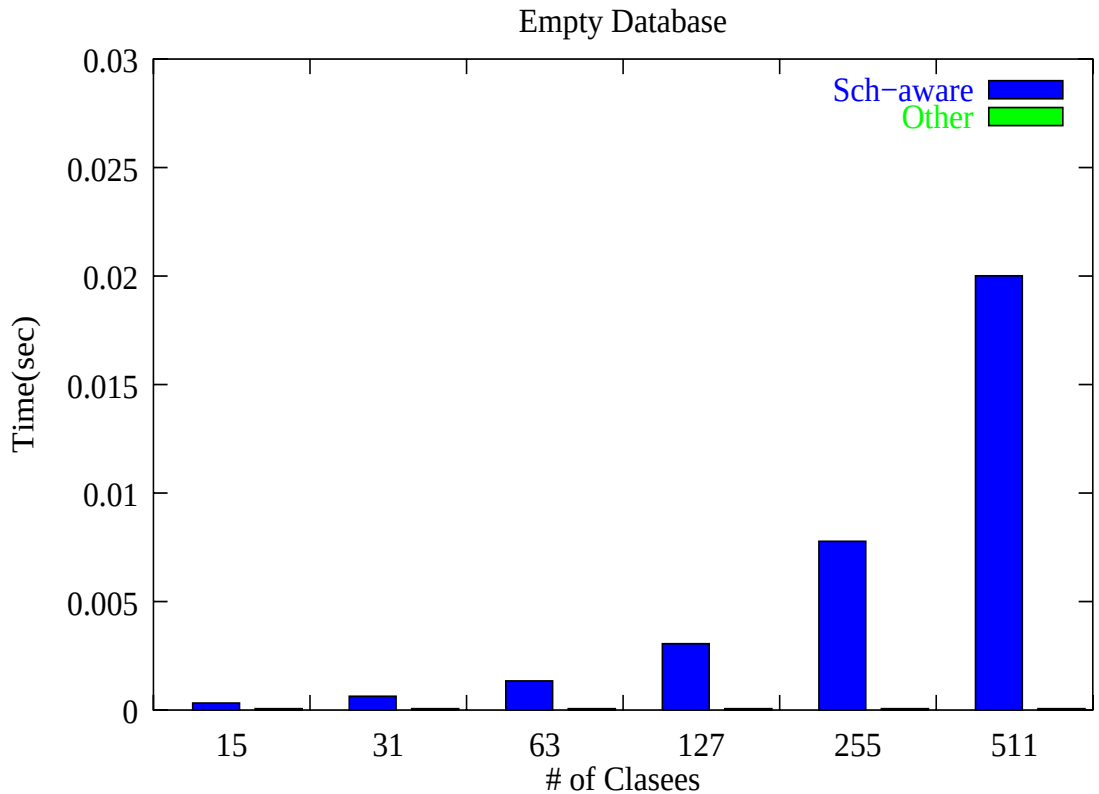
$$totalTriples(A,d) \approx d * A$$

- zipfian: almost all resources on leaf classes
  - total triples would converge with (d+1) * A

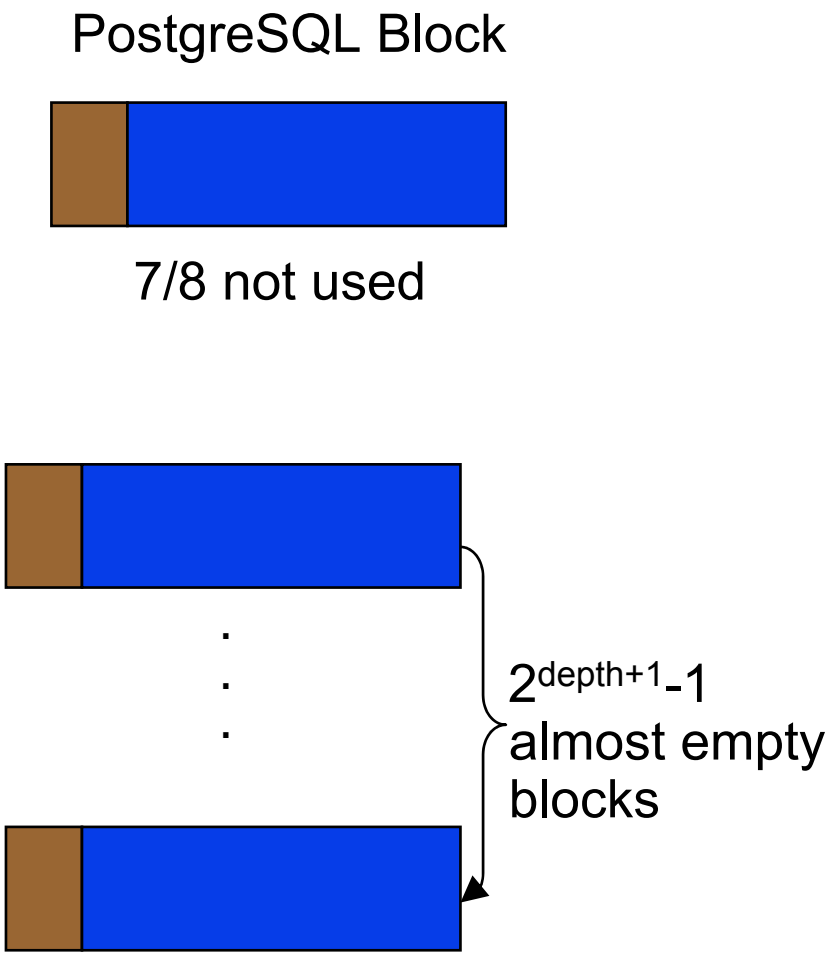
# of Resources	**ISA, NOISA Hybrid**	**URI**	**ID**	**MatView**
10,000	10 MB	depth * 20 MB	10 + 0.52*depth MB	depth * 10 MB
100,000	100 MB	depth * 200 MB	100 + 5.2*depth MB	depth * 100 MB
1,000,000	1 GB	depth * 2 GB	1 + 0.052*depth GB	depth * 1 GB

The Effect of Schema Size (1/2)

- Extensional filtering phase of taxonomic queries:
  - schema-aware (both ISA and NOISA) need to scan a number of (possibly empty) instance tables
  - all the other representations need to scan only one (possible empty) table regardless of the number of schema classes



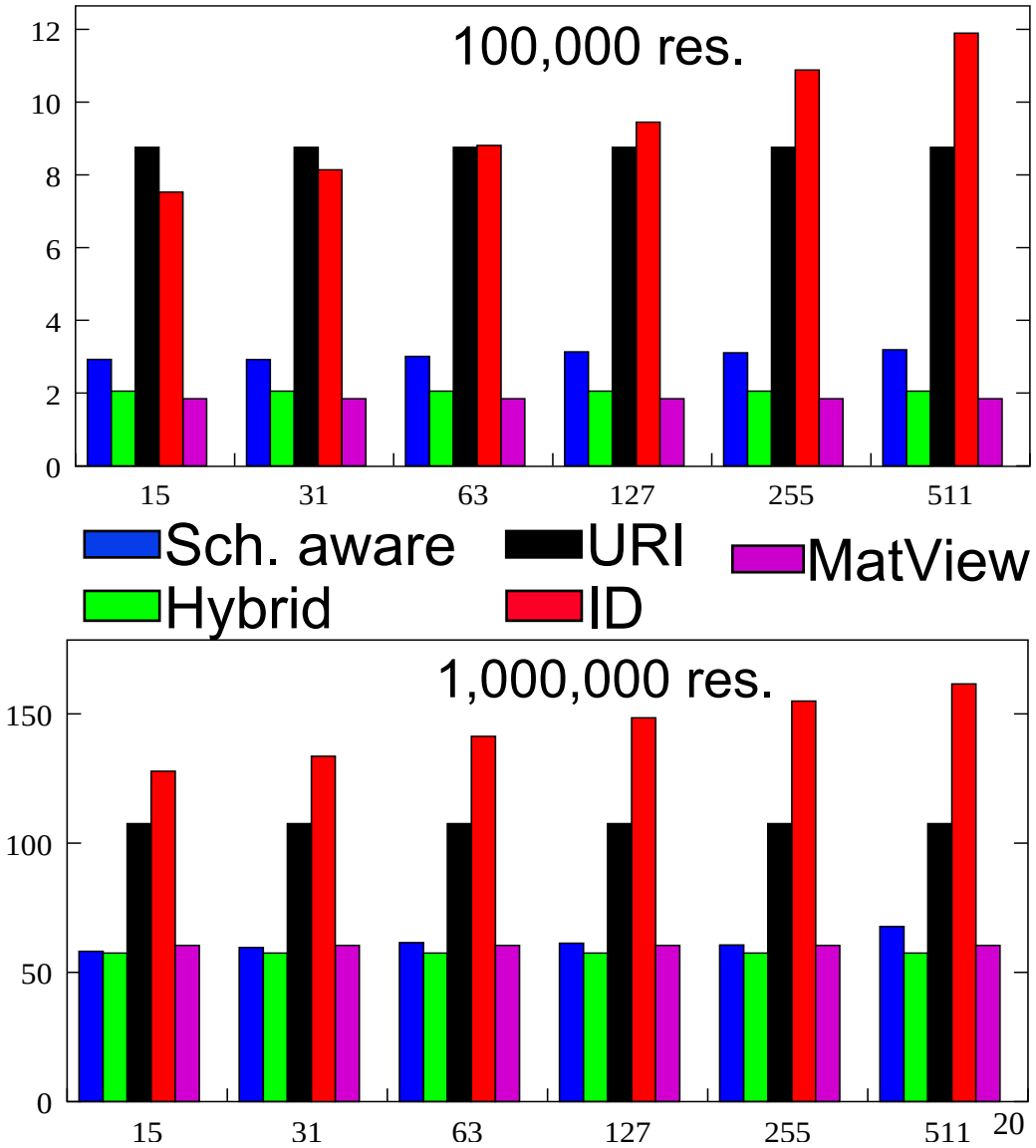
The Effect of Schema Size (2/2)



- The last block of every table in an RDBMS is not completely full
- Schema-aware uses one table per class
  - $2^{depth+1}-1$ blocks can be almost empty
- For a given URI size of 1KB this storage overhead varies between $0 - (2^{depth+1}-1) * 8KB$
  - More I/O activity incurs in the two schema-aware representations

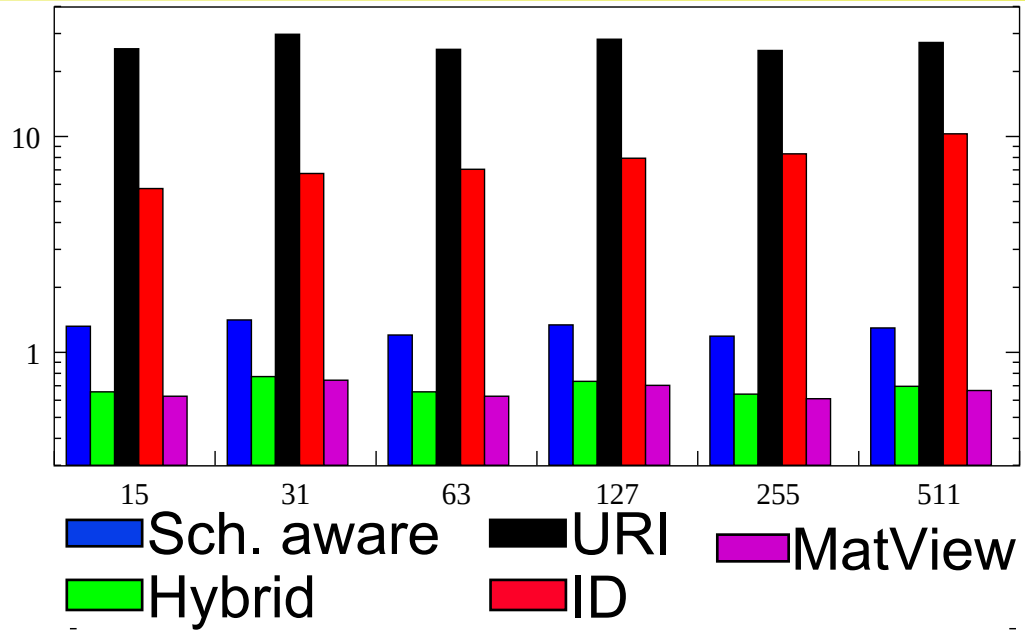
Querying the Root Class

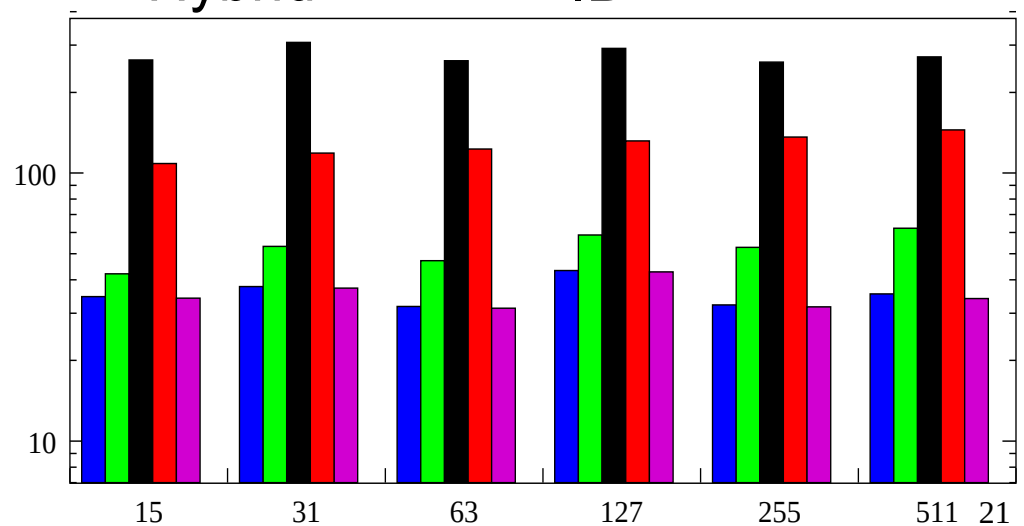
- All need only a scan
- **Hybrid**, **MatView** overall best
- **Schema-aware** is penalized by the storage overhead
- **Hybrid**, **URI**:
  - the size of the record is important
    - **URI** triple size is 2 times bigger than the tuple size of **Hybrid**
- **ID** is the worst:
  - requires an extra join
  - depth times larger number of triples than those explicit given



Querying a Middle Level Class

- Small and medium # of resources: **Hybrid**, **MatView** overall best
- Large # of resources: **schema-aware**, **MatView** overall best
- **ID** performs better than **URI**
  - Far worse than the others

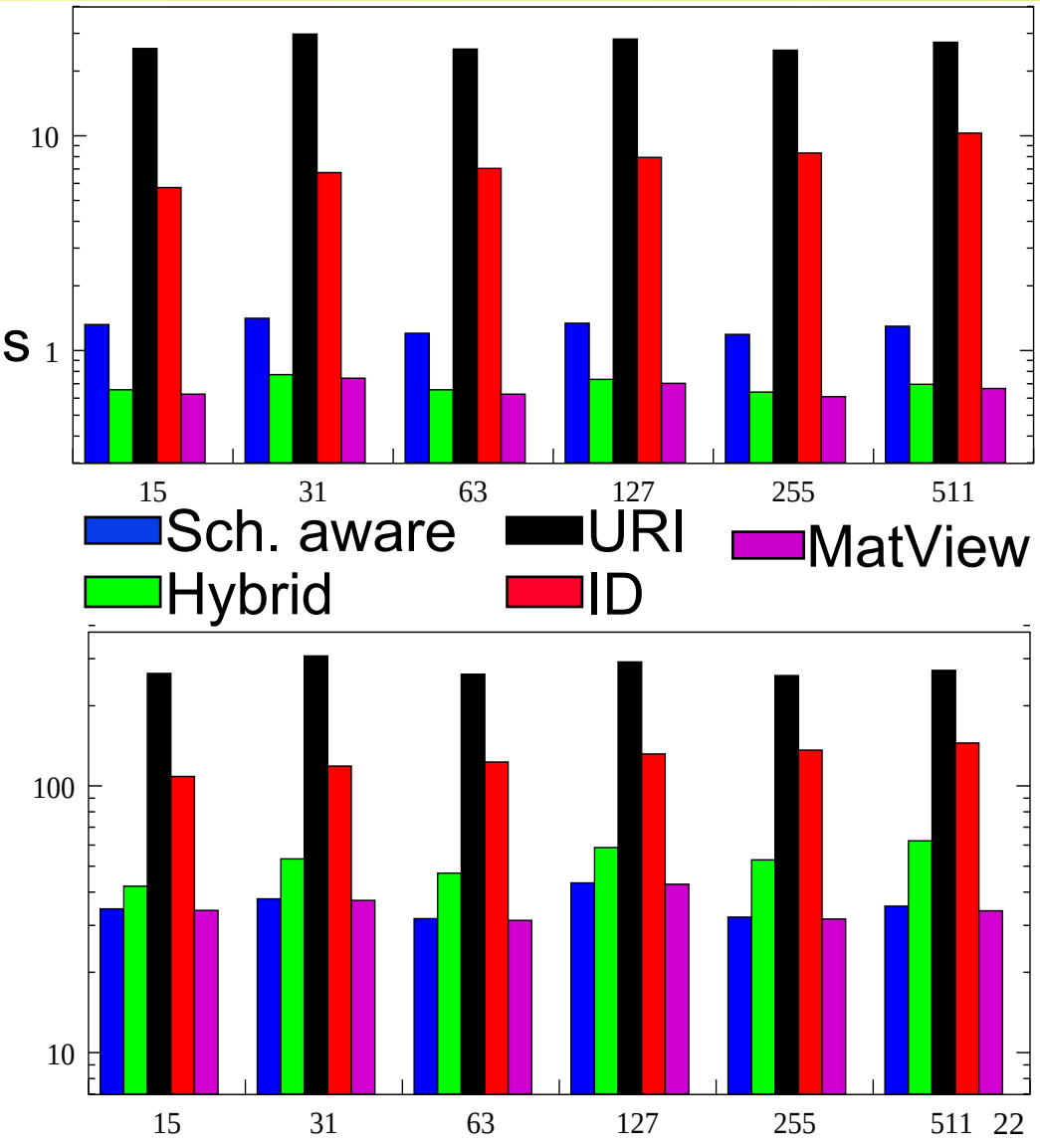




	Selectivity
Zipfian favouring subtrees	35% - 45%
Zipfian favouring leaves	45% - 55%
Uniform	80% - 94%

Querying a Middle Level Class

- **Hybrid** and **URI** need a selection 'INDEX SCAN':
  - is efficient for high selectivities (uniform)
  - less efficient for low selectivities (zipfian)
- The overhead of accessing the index in **URI** is even bigger than in **Hybrid**

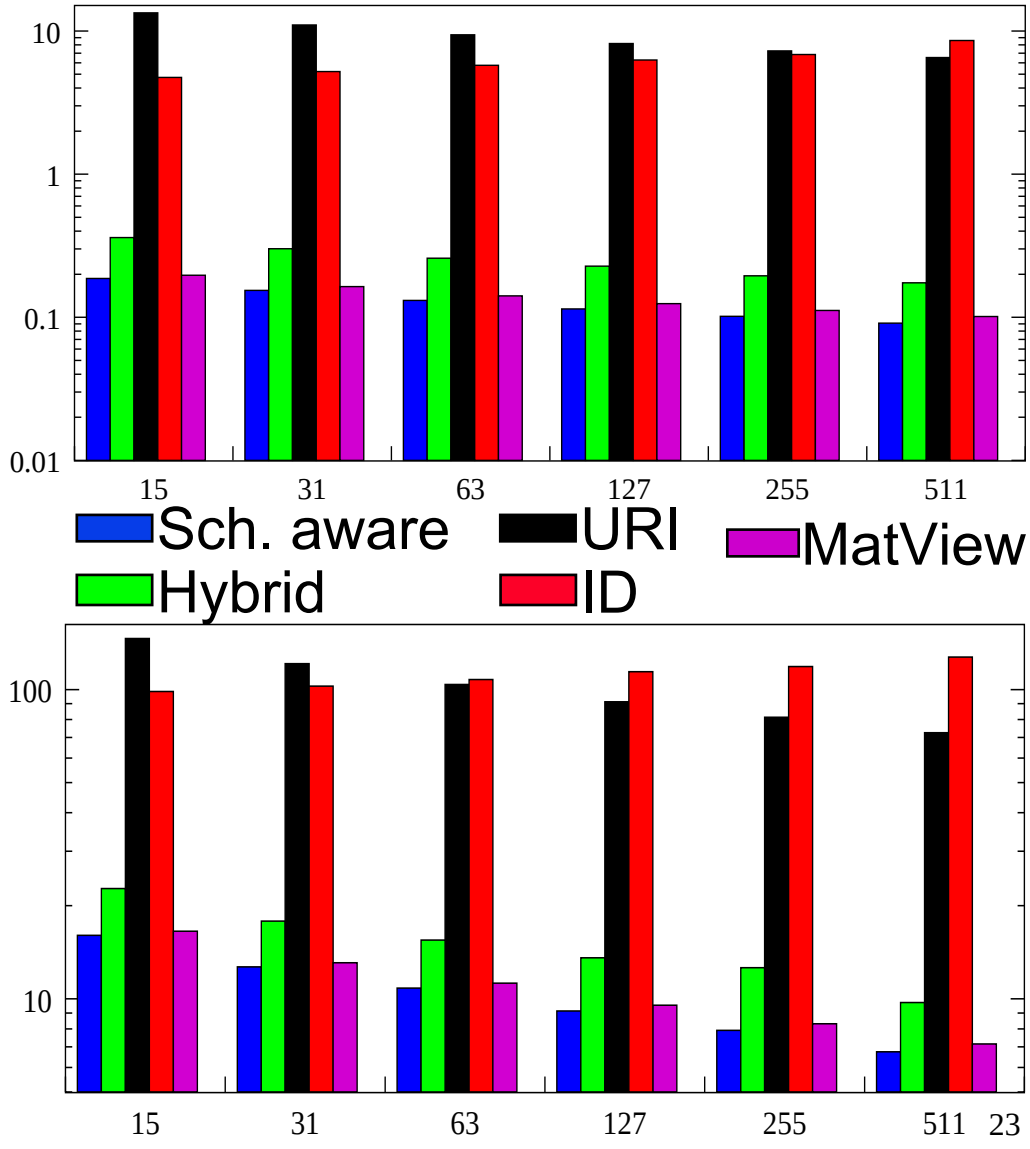


Index size

# of res.	Hybrid	URI
10,000	15.6 KB	13 MB
100,000	156 KB	130 MB
1,000,000	1.56 MB	1.3 GB

Querying Leaves

- **Schema-aware** have to scan only a single table
  - no significant I/Os due to space left at the end of the blocks
  - exhibit the same (overall best) behavior as **MatView**
- Higher selectivity than in case of queries at middle level
  - **Hybrid** performances converge with those of **schema-aware** and **MatView**
- **URI** and **ID** follow by far



Conclusions

- Hybrid and MatView *outperform* the other storage schemes for taxonomic queries
  - Compared to MatView, Hybrid is also space optimal
- Schema-aware representations achieve similar performances to Hybrid and MatView for
  - ***medium*** or ***large*** number of resources and queries on ***root*** class
  - they exhibit best performance for queries at **leaf** level classes
- Schema-oblivious representations like URI and ID *exhibit the worst performance*
  - for **zipfian** distribution and queries at ***middle*** or ***leaf*** level classes ID outperforms URI
  - otherwise, URI outperforms ID

Conclusions

		Querying Root			Querying Middle Level Classes				Querying Leaves		
		Small	Med	Large	Small	Med	Large		Small	Med - Large	
							U	Z		U	Z
Sch. aware		-	o	+	o	o	+	+	+	+	+
MatView		+	+	+	+	+	+	+	+	+	+
Hybrid		+	+	+	+	+	+	o	+	+	o
Schema Oblivious	URI	o	-	-	-	-	-	-	-	-	-
	ID	-	-	-	-	-	-	-	-	-	-

+ Best
o Good
- Bad

Future Work

- Taxonomic queries is the first step for benchmarking database representations of RDF/S Stores
- Further benchmark path queries involving:
  - data,
  - schema, and
  - mixed
- We plan to extend our generator with appropriate distribution modes of properties over (domain or range) classes

Questions ?



?