Incremental Maintenance Of Materialized Ontologies

Raphael Volz[1,2], Steffen Staab[1], and Boris Motik[2]

[1] Institute AIFB, University of Karlsruhe
76128 Karlsruhe, Germany
{volz,staab}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS/
[2] WIM, Forschungszentrum Informatik (FZI)
76131 Karlsruhe, Germany
{volz,motik}@fzi.de
http://www.fzi.de/wim/

This paper discusses the incremental maintenance of materialized ontologies in a rule-enabled Semantic Web. Materialization allows to speed up query processing by explicating the implicit entailments which are sanctioned by the semantics of an ontology. The complexity of reasoning with the ontology is thereby shifted from query time to update time. We assume that materialization techniques will frequently be important to achieve a scalable Semantic Web, since read access to ontologies is predominant. Central to materialization are maintenance techniques that allow to incrementally update a materialization when changes occur.

We present a novel solution that allows to cope with changes in rules and facts. To achieve this we extend a known approach for the incremental maintenance of views in deductive databases. We show how our technique can be employed for a broad range of existing Web ontology languages, such as RDF/S and subsets of OWL and present a first evaluation.

1 Introduction

Germane to the idea of the Semantic Web are the capabilities to assert facts and to derive new *implicit* facts from the asserted facts using the semantics specified by an ontology. The current building blocks of the Semantic Web, Resource Description Framework (RDF) [10] and Web Ontology Language (OWL) [15], define how to assert facts and specify how implicit facts should be derived from stated facts.

The derivation of implicit information is usually achieved at the time clients issue queries to inference engines. Situations where the query rate is high or the procedure to derive implicit information is time consuming and complex lead to slow performance. Materialization can be used to increase the performance at query time and to make implicit information explicit. This avoids to recompute derived information for every query.

Materialization has been applied successfully in many applications where reading access to data is predominant. For example, data warehouses usually apply materialization techniques to make *online* analytical processing possible. In the traditional web, portals maintain cached web pages to offer fast access to dynamically generated web pages.

We assume that reading access to ontologies is predominant in the Semantic Web and other ontology based applications, hence materialization seems to be a promising technique for fast query processing.

Central to materialization approaches is the issue of maintaining a materialization when changes occur. This issue can be handled by simply recomputing the whole materialization, however as the computation is often complex and time consuming, typically more efficient techniques need to be applied.

Contribution We present a technique for the incremental maintenance of materialized ontologies. Our technique can be applied to a wide range of ontology languages, namely those that can be axiomatized by a set of rules[3].

The challenge that has not been tackled before comes from the fact that updates of ontology definitions are equivalent to the update and new definitions of rules, whereas existing maintenance techniques only address the update of ground facts.

To cope with changing rules, our solution extends a declarative algorithm for the incremental maintenance of views [19] that was developed in the deductive database context. We show the feasibility of our solution in a first performance evaluation.

Paper structure The remainder of the paper will consist of a review of how Web ontology languages and rules interplay (Section 2), presentation of the algorithm for maintenance for changing facts (Section 3), extension of the algorithm for maintenance with changing rules (Section 4), a first performance evaluation (Section 5), a review of related work (Section 6) followed by general conclusions (Section 7).

2 Web ontology languages and rules

2.1 Axiomatization of the language

Since the early days of the Semantic Web, many systems have tried to reason with Web ontology languages using rule-based systems, e.g. SilRi [4], CWM[4], Euler [16], JTP[5] or Triple [17]. To do so, a particular Web ontology language is axiomatized via a static set of rules which capture the semantics specified for a particular ontology language.

For example, Figure 1 presents the Datalog axiomatization of RDF/S. This axiomatization implements the semantics of RDF specified by the RDF model theory [10] (without datatype entailments and support for stronger iff semantics of domain and ranges). The ontology and associated data is stored in a single ternary predicate t, i.e. the extension of t stores all triples that constitute a particular RDF graph.

In many applications, e.g. editors, it is necessary to distinguish between asserted information and entailed information [2]. This can be achieved by turning the predicate t into a completely intensional predicate (view) that is derived from the explicitly asserted

[3] The underlying rule language used for our approach is Datalog with stratified negation.
[4] `http://www.w3.org/2000/10/swap/doc/cwm`
[5] `http://ksl.stanford.edu/software/jtp/`

t(P,a,rdf:Property)	:-	t(S,P,O).	*rdf1*
t(S,a,C)	:-	t(P,domain,C), t(S,P,O).	*rdfs2*
t(O,a,C)	:-	t(P,range,C), t(S,P,O).	*rdfs3*
t(S,a,Resource)	:-	t(S,P,O).	*rdfs4a*
t(O,a,Resource)	:-	t(S,P,O).	*rdfs4b*
t(P,subPropertyOf,R)	:-	t(Q,subPropertyOf,R), t(P,subPropertyOf,Q).	*rdfs5a*
t(S,R,0)	:-	t(P,subPropertyOf,R), t(S,P,O).	*rdfs6*
t(C,a,Class)	:-	t(C,subClassOf,Resource).	*rdfs7*
t(A,subClassOf,C)	:-	t(B,subClassOf,C), t(A,subClassOf,B).	*rdfs8*
t(S,a;B)	:-	t(S,a,A), t(A,subClassOf,B).	*rdfs9*
t(X,subPropertyOf,member)	:-	t(X,a,ContainerMembershipProperty).	*rdfs10*
t(X,subClassOf,Literal)	:-	t(X,a,Datatype).	*rdfs11*
t(Resource,subClassOf,Y)	:-	t(X,domain,Y), t(rdf:type,subPropertyOf,X).	*rdfs12*

Fig. 1. Static Datalog rules for implementing RDF(S)

information. Hence, asserted RDF triples constitute a separate extensional predicate, say t_{Ext}, and t is derived from t_{Ext} via a rule:

$$t(X, Y, Z) \text{ :- } t_{Ext}(X, Y, Z).$$

Such view definitions allow not only to distinguish between asserted and entailed information in queries but also to re-use results established for the incremental maintenance of views (intensional predicates) in the deductive database context for the purpose of materialization.

2.2 Dynamic rule sets

The set of rules is typically not immutable. With the advent of higher layers of the Semantic Web stack, i.e. the rule layer, users can create their own rules. Hence, we are facing a scenario where not only base facts can change but also the set of rules. This requires the ability to maintain a materialization in this situation.

Besides support for a rule layer, the ability to maintain a materialization under changing rule sets is also required for approaches where the semantics of the ontology language is not captured via a static set of rules but instead compiled into a set of rules. Such an approach is for example required by Description Logic Programs (DLP) [6], where OWL ontologies are translated to logic programs. Other implementations of knowledge representation languages, e.g. O-Telos [11] and F-Logic [21], have also been achieved via such a compilation.

Semantic Web Rule layer We now briefly present some languages for the specification of Semantic Web rules that may be compiled into the paradigm we use. The Rule Markup Initiative aims to develop a canonical Web language for rules called RuleML. RuleML covers the entire rule spectrum and spans from derivation rules to transformation rules to reaction rules. It has a well-defined Datalog subset, which can be enforced using XML schemas, and for which we can employ the materialization techniques developed within this paper. The reader may note, that materialization is not an issue for many other aspects found in RuleML, e.g. transformation rules or reaction rules.

In parallel to the RuleML iniative, Notation3 (N3) has emerged as a human-readable language for RDF/XML. Its aim is to optimize expression of data and logic in the same language and has become a serious alternative since many systems that support inference on RDF data (e.g. cwm, Euler, Jena2) support it. The rule language supported by N3 is an extension of Datalog with existential quantifiers in rule heads Hence, the materialization techniques developed within this paper can be applied to the subset of all N3 programs which do not make use of existential quantification in the head.

Description Logic Programs Both of the above mentioned approaches allow the definition of rules but are not integrated with the ontology layer in the Semantic Web architecture. Description Logic Programs aim to integrate rules with the ontology layer by compiling ontology definitions into a logic program which can later be extended with additional rules. This approach can deal with a very expressive subset of the standardized Web ontology language OWL (i.e. OWL without existential quantification, negation and disjunction in rule heads).

The following example OWL fragment declares Wine to be potable liquids who are made by Wineries:

$$\text{Wine} \sqsubseteq \text{PotableLiquid} \sqcap \forall \text{hasMaker}.\text{Winery}$$

This will be translated in DLP to the following set of rules:

$$\begin{array}{lll}
\text{Wine}(X) & \text{:-} & \text{PotableLiquid}(X), \text{hasMaker}(X, Y), \text{Winery}(Y). \\
\text{PotableLiquid}(X) & \text{:-} & \text{Wine}(X). \\
\text{Winery}(Y) & \text{:-} & \text{Wine}(X), \text{hasMaker}(X, Y).
\end{array}$$

Hence, a change to the ontologies class and property structure will result in a change of the compiled rules. Again, it is necessary to be able to maintain a materialization in case of such a change.

3 Changing facts

This section presents the maintenance of a materialization when facts change, viz. new tuples are added or removed from the extension of a predicate. We first demonstrate how changes can effect a materialization in an example and then recapitulate the incremental maintenance approach [19] upon which we build our approach.

3.1 An example

Let's consider the effects of adding and removing a subClassOf relationship in RDF/S with respect to some of the relevant rules that axiomatize RDF Schema. The first rule

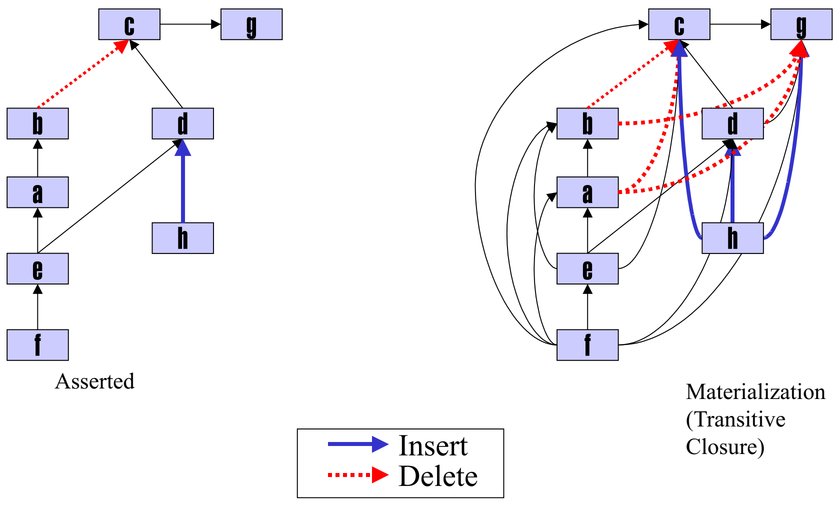

Fig. 2. Changes to a RDF/S taxonomy and its materialization

links the intensional predicate t with ground facts. The second rule implements the transitive closure of the subClassOf relationship:

R_1 : $t(X, Y, Z)$:- $t_{ext}(X, Y, Z)$.
R_2 : t(A,subClassOf,C) :- t(B,subClassOf,C), t(A,subClassOf,B).

Let us consider the effects of updates on the taxonomy depicted in Figure 2:

– b is no longer a subclass of c, viz. the $t_{ext}(b, \text{subClassOf}, c)$ fact is deleted.
– h is asserted to be a subclass of d, hence a $t_{ext}(h, \text{subClassOf}, d)$ fact is inserted.

The deletion has the following consequences to t: It eliminates the links between (a,c), (a,g), (b,c) and (b,g). Since the facts $t_{ext}(d, \text{subClassOf}, c)$ and $t_{ext}(e, \text{subClassOf}, d)$ exist, alternative derivations also exist for the links (e,c), (e,g), (f,c) and (f,g). These alternative derivations have to be taken into account in our approach. The insertion yields three new derivations namely links between (h,c), (h,d) and (h,g).

3.2 Generating Maintenance Rules

Several algorithms (e.g. [9, 12, 7]) have been presented for the incremental maintenance of views (or intensional predicates) in the deductive database context. The most common procedure is to compute the changes (differentials) to views in three steps:

– Firstly, to overestimate the consequences of deletions, so a super set of the facts that are eventually deleted is computed for deletion.
– Secondly, a rederivation step prunes those computed deletions from the set of deletions for which alternative derivations (via some other rules defining the view) exist.

– Thirdly, the consequences of insertions to extensional predicates are added to the view, if applicable.

Our approach is based on the results established in [19], which realize the DRed (delete and rederive) algorithm presented in [7] in a purely declarative way. The approach is based on rewriting the original program into a maintenance program, which is evaluated instead of the old program.

Hence, an original Datalog rule:

$$p \text{ :- } r_1, \ldots, r_n.$$

is rewritten into a set of new predicates and several maintenance rules that calculate the differentials required to maintain a materialization. The maintenance of an intensional predicate p is achieved via six additional predicates:

1. p^{Del} computes an overestimation of facts that ought to be deleted from the materialization so-called deletion candidates. For extensional predicates p^{Del} contains explicitly what should be removed from the materialization.
2. p^{Ins} contains the facts that ought to be inserted into the materialization. For extensional predicates p^{Ins} contains explicitly what should be inserted into the materialization.
3. p^{Red} stores those facts that are marked for deletion but have alternative derivations.
4. p^{New} describes the new state of the materialization after updates.
5. p^{Plus} computes the net insertions required to maintain the materialization.
6. p^{Minus} computes the net deletions required to maintain the materializaion.

The extension of p itself contains the materialization. The reader may note that the evaluation of the set of maintenance rules computes the set of implicit insertions and deletions that have to be propagated to the materialization of the predicate and to other predicates, which depend on the predicate through some rules.

Deletion Candidates The first subset of maintenance rules derive all possible deletion candidates for a predicate p. Deletion candidates are constituted by deleted facts in the body predicates. For all rules and all body predicates r_i in these rules, we define a rule

$$D_i: \quad p^{Del} \text{ :- } r_1, \ldots, r_{i-1}, r_i^{Del}, r_{i+1}, \ldots, r_n.$$

If r_i is a extensional predicate, r_i^{Del} contains either the explicitly deleted facts in r_i or it contains a superset of the derived facts to be deleted in r_i due to deletions caused by other rules.

With respect to our example we would have to generate three deletion rules. The transformation of $R1$ yields one deletion rule, while the transformation of $R2$ results in two deletion rules.

$$\begin{array}{lll}
R_1D_1: & t^{Del}(X,Y,Z) & \text{:- } t^{Del}_{ext}(X,Y,Z). \\
R_2D_1: & t^{Del}(A, \text{subClassOf}, C) & \text{:- } t^{Del}(B, \text{subClassOf}, C), t(A, \text{subClassOf}, B). \\
R_2D_2: & t^{Del}(A, \text{subClassOf}, C) & \text{:- } t(B, \text{subClassOf}, C), t^{Del}(A, \text{subClassOf}, B).
\end{array}$$

If b is no longer a subclass of c (viz. $t^{Del}_{ext}(b, \text{subClassOf}, c)$ exists), the extension of the t^{Del} predicate would be constituted by the following derived subclass relationships:

$$(b, c), (b, g), (a, c), (a, g), (e, c), (e, g), (f, c), (f, g)$$

Rederivation We now have to check which tuples in p^{Del} have alternative derivations in the new database state. This is achieved using the following rule:

$$R: \quad p^{Red} \text{ :- } p^{Del}, r_1^{New}, \ldots, r_n^{New}.$$

With respect to our example, t^{Red} is axiomatized by two rules:

$$\begin{array}{lll}
R_1R: & t^{Red}(X, Y, Z) & \text{:- } t^{Del}(X, Y, Z), t^{New}_{ext}(X, Y, Z). \\
R_2R: & t^{Red}(A, \text{subClassOf}, C) & \text{:- } t^{Del}(A, \text{subClassOf}, C), \\
 & & \quad t^{New}(B, \text{subClassOf}, C), t^{New}(A, \text{subClassOf}, B).
\end{array}$$

t^{Red} has the following subClassOf relationships as its extension:

$$(e, c), (e, g), (f, c), (f, g)$$

This is due to the facts $t(d, \text{subClassOf}, c)$ and $t(e, \text{subClassOf}, d)$, which generate these alternative derivations.

Insertion The next rules propagate insertions of extensional facts to the intensional predicates. This is done by ordinary semi-naive rewriting, i.e. by constructing rules (I_i) that join new facts inserted into one body relation with full extensions of all others:

$$I_i: \quad p^{Ins} \text{ :- } r_1^{New}, \ldots, r_{i-1}^{New}, r_i^{Ins}, r_{i+1}^{New}, \ldots, r_n^{New}.$$

With respect to our example we have to generate three insertion rules (one for R_1 and two for R_2):

$$\begin{array}{lll}
R_1I_1: & t^{Ins}(X, Y, Z) & \text{:- } t^{Ins}_{ext}(X, Y, Z). \\
R_2I_1: & t^{Ins}(A, \text{subClassOf}, C) & \text{:- } t^{Ins}(B, \text{subClassOf}, C), t^{New}(A, \text{subClassOf}, B). \\
R_2I_2: & t^{Ins}(A, \text{subClassOf}, C) & \text{:- } t^{New}(B, \text{subClassOf}, C), t^{Ins}(A, \text{subClassOf}, B).
\end{array}$$

If we assert h to be a subclass of d (viz. $t^{Ins}_{ext}(h, \text{subClassOf}, d)$ exists), we can derive the following new subclass relationships in the extension of t^{Ins}:

$$(h, d), (h, c), (h, g)$$

Description of the new state The new state of a predicate p after updates is captured in a new predicate p^{New}. A set of rules captures the changes:

N_1: p^{New} :- $p, \neg p^{Del}$.
N_2: p^{New} :- p^{Red}.
N_3: p^{New} :- p^{Ins}.

Rule N_1 ensures that the new state of a predicate p does not contain the deleted information. Rule N_2 ensures that rederived facts are part of the new state of the predicate. Finally, rule N_3 pushes insertions into the new state of the predicate.

Computation of differentials We can compute the deletions and insertions that have to be performed to maintain the materialization of a predicate p via two predicates p^{Plus} and p^{Minus}, which compute the net insertions and deletions to a predicate p.

p^{Plus} contains those facts, which were not present before and are derived for insertion. p^{Minus} contains those facts, which are deletion candidates and are neither inserted nor re-derived.

P: p^{Plus} :- $p^{Ins}, \neg p$.
M: p^{Minus} :- $p^{Del}, \neg p^{Ins}, \neg p^{Red}$.

In our example the extension of t^{Plus} is made up by the following subclassOf assertions:

$$(h, d), (h, c), (h, g)$$

The extension of t^{Minus} is:

$$(a, c), (a, g), (b, c), (b, g)$$

3.3 Static RDF/S rules revisited

The 15 static Datalog rules for the axiomatization of RDF/S (cf. Figure 1) contain 21 body predicates. This leads to the generation of 21 insertion rules and 21 deletion rules. Additionally 15 rederivation rules are generated. 5 predicates are generated to capture the new state of the predicate t and the differentials. The total number of 62 generated rules can be found online[6].

3.4 Evaluation of maintenance rules

[18] show that the evaluation of the maintenance rules is a sound and complete procedure for computing the differentials between two database states when extensional update operations occur.

During the evaluation it is necessary to access the old state of a predicate. Bottom-up evaluation approaches therefore require that all intensional relations involved in the computation are completely materialized, viz. the initial rules defining the predicates are not considered during the evaluation of the maintenance rules.

[6] http://kaon.semanticweb.org/research/materialization

The rewriting contain negated predicates to express the algebraic set difference operation. Hence, even though the original rules may be pure Datalog (without negation), a program with negation is generated. The rewriting transformation keeps the property of stratifiability, since newly introduced predicates do not occur in cycles with other negations. Hence, the evaluation can partition predicates into strata such that no two predicates in one stratum depend negatively on each other. Predicates may only occur negatively in rules that define predicates of a higher stratum. The evaluation can then proceed as usual stratum-by-stratum starting with the extensional predicates themselves.

Not only changed predicates have to be maintained but also all predicates that depend on predicates whose extension changes. An axiomatization of RDF/S based on a single ternary predicate (cf. Figure 1) therefore leads to complete re-materialization in case of updates. We present an optimization for this case in Section 4.3 which results in more efficient results.

4 Changing rules

This section presents the maintenance of a materialization if the definition of views (intensional predicates) changes, viz. rules that define the predicate are added or removed. Our solution has two main components. Firstly, the materialization itself has to be maintained. Secondly, the materialization rules for a predicate p themselves have to be maintained.

	DL-Style	DLP Translation	explanation
1.	hasChild $\sqsubseteq$ inDynasty	inDynasty(X,Y) :- hasChild(X,Y).	*hasChild constitutes inDynasty*
2.	inDynasty$^+$ $\sqsubseteq$ inDynasty	inDynasty(X,Y) :- inDynasty(X,Z), inDynasty(Z,Y).	*the inDynasty relation is transitive*
3.	inDynasty$^-$ $\equiv$ inDynasty	inDynasty(X,Y) :- inDynasty(Y,X).	*the inDynasty relation is symmetric*

Table 1. Versions of the example ontology: (a) DL-based (b) DLP translation

4.1 An example

Table 1 depicts a small sample ontology that describes the relationships in a family dynasty. The OWL rules stated in DL-style syntax are compiled into appropriate Datalog rules using the DLP approach. Let's assume that DLP has been used to implement the semantics. If the ontology only contains the first and second axiom then the extension of `hasChild` is constituted by the tuples $\{(1,2),(2,3),(4,3)\}$. Hence, the extension of `inDynasty` corresponds to hasChild $\cup$ $\{(1,3)\}$

If the first axiom is deleted, `inDynasty` has an empty extension. Similarly the extension of `inDynasty` would be equivalent to `hasChild`, if the second axiom were deleted.

Now assume that the third rule is inserted. Apparently the new extension will contain the tuple $(1, 4)$ (among others), which is derived by one of the existing rules, i.e. rule 2 which operated on tuples derived by rule 3.

4.2 Maintaining the materialization

Every change in the rule set causes changes in the extension of a predicate p. Hence, the materialization has to be updated as well. However, unlike in the case of changing extensions, the existing maintenance rules cannot capture this situation.

As we can see in the above examples, adding and removing rules requires the reevaluation of all other rules defining a predicate. The reader may note, that it does not suffice to simply change the maintenance rules. Since there is no change to `hasChild`, both predicates which capture the difference hasChild^{Plus} and hasChild^{Minus} are empty. In consequence inDynasty^{Ins}, inDynasty^{Del}, inDynasty^{Red} and thereby inDynasty^{Plus} and inDynasty^{Minus} are empty.

Our solution is based on the creation of a temporary predicate p^{Temp}, which is used to calculate the extension of p using the changed set of rules. Hence, p^{Temp} is axiomatized using the updated rule set for a predicate p. Self-references of the predicate have to be substituted by the temporary predicate:

$$\begin{array}{lll}
\text{inDynasty}^{Temp}(x,y) & \text{:-} & \text{hasChild}(x,y). \\
\text{inDynasty}^{Temp}(x,y) & \text{:-} & \text{inDynasty}^{Temp}(x,z), \text{inDynasty}^{Temp}(z,y). \\
\text{inDynasty}^{Temp}(x,y) & \text{:-} & \text{inDynasty}^{Temp}(y,x).
\end{array}$$

Then, p^{Temp} isused for the calculation of p^{Plus} and p^{Minus} by augmenting the definition of p^{Ins} and p^{Del} with the following rules:

$$\begin{array}{lll}
p^{Ins} & \text{:-} & p^{Temp}, \neg p. \\
p^{Del} & \text{:-} & p, \neg p^{Temp}.
\end{array}$$

The view maintenance process is carried out by evaluating the maintenance rules without the initial rules that define p. All predicates, which depend on p can be updated using p^{Ins} and p^{Del}. The reader may note, that our solution allows the simultaneous modification of both rules and facts. However, the new facts can already be taken into account when p^{Temp} is computed.

4.3 Selection-based Optimization

Alternatively to DLP, the semantics of the ontology stated in Table 1 could have been given by specifying several rules that axiomatize a single triple predicate. For example the symmetry of the dynastic relationship could be axiomatized as follows:

$$t(X, \text{inDynasty}, Y) \text{ :- } t(Y, \text{inDynasty}, X).$$

In this case our approach requires to access the whole database, since only one predicate is materialized and all rules defining this predicate have to be reevaluated. Naturally, this situation corresponds to the simple strategy of recomputing a materialization from scratch.

We therefore introduce an optimization, which improves the recomputation by limiting the part of the database which takes part in the evaluation. Selection-based optimization assumes that the extension of the database is split depending on split points. Split points are given by constants that occur at a certain argument position of a predicate.

The optimization transforms a Datalog program into an equivalent program, such that all references to p where a split point occurs are replaced by split predicates. This is the case, if a constant c was used as the i-th argument in the predicate p.

To generate split predicates, we split the extension of a predicate p_{Ext} into several edb predicates of the form $p^{c_i}_{Ext}(Var_1, Var_2, \ldots, Var_{i-1}, c, Var_{i+1}, Var_n)$ to store tuples based on equal constant values c in their i-th component.

Hence, instead of using a single predicate p_{Ext} for representing the direct RDF assertion, the database is split into several $p^{c_i}_{Ext}$. Again, we want to distinguish between asserted and derived information and consequently introduce intensional predicates (views) for each component of the extension (i.e. rules of the form p^{c_i} :- $p^{c_i}_{Ext}$).The complete predicate p is represented by n rules that unify the used split predicates: p :- p^{c_i}

Returning to the triple based axiomatization of the example, we can transform the program by introducing a split point $t^{inDynasty_2}$ for the `inDynasty` constant (when used as second argument in the ternary predicate t):

– We use two extensional predicates: $t^{Rest}_{Ext}, t^{inDynasty_2}_{Ext}$ to store the extension in two disjoint sets.
– We capture the intensional predicates and integrate the splits into a complete extension of t and rewrite the example rule 2 such that split predicates are used instead of the full predicate:

$$\begin{array}{lll}
t^{Rest}(X,Y,Z) & \text{:-} & t^{Rest}_{Ext}(X,Y,Z). \\
t^{inDynasty_2} & \text{:-} & t^{inDynasty_2}_{Ext}(X,Y,Z). \\
t(X,Y,Z) & \text{:-} & t^{Rest}(X,Y,Z). \\
t(X,Y,Z) & \text{:-} & t^{inDynasty_2}(X,Y,Z). \\
t^{inDynasty_2}(X, \text{inDynasty}, Y) & \text{:-} & t^{inDynasty_2}(Z, \text{inDynasty}, Y), \\
 & & t^{inDynasty_2}(X, \text{inDynasty}, Y).
\end{array}$$

Assume now that the third rule of Table 1 is inserted. We can again transform this rule into a rule, which uses the split predicate $t^{inDynasty_2}$. However, the maintenance of $t^{inDynasty_2}$ can now be carried out by ignoring all non-relevant rules for t. Hence, the whole extension of t can be updated via the insert and delete maintenance rules that are created for $t^{inDynasty_2}$ only, viz. without using the complete database.

4.4 Maintaining maintenance rules

Additionally to the maintenance of the materialization itself, the maintenance rules have to be altered when changes occur. In case of insertion of a rule r new maintenance rules are generated. In case of deletion of a rule r, all maintenance rules generated from the rule are deleted as well. If the predicate p itself is deleted, i.e. the last rule r defining p is deleted, all maintenance predicates, e.g. p^{New}, p^{Red} etc. are removed from the database by removing the rules that define those predicates.

The maintenance algorithms operate on the following data structures:

– R set of rules
– MR set of maintenance rules,
– P set of predicates,
– MP set of maintenance predicates,
– $ruleMaintenance : R \rightarrow MR$ function that maps rule to its maintenance rules
– $head : R \rightarrow P$ function maps a rule to the rule head
– $rules : P \rightarrow R$ function that maps a predicate to defining rules

Two procedures generate the maintenance rules for a given rule r and its head p:

– $staticMaintenanceRules$ generates the rules defining p^{New}, p^{Plus} and p^{Minus},
– $dynamicMaintenanceRules$ generates the rules defining p^{Del}, p^{Red} and p^{Ins}

Maintaining maintenance rules is achieved by algorithm 1. The if branch checks whether a rule is the first (respectively last) rule defining a predicate p and generate (respectively delete) the static maintenance predicates and rules.

Algorithm 1 Changing Rules

```
Require: New rules Changes that are changed, operator $\circ = \{\cup, \backslash\}$ specifying insert or delete
  $R = R\circ$ Changes
  for all $r \in Changes$ do
    maintenanceRules = $\emptyset$
    p = head( r )
    if $((\{p\} \cap P = \emptyset) \wedge (\circ = \cup)) \vee ((\text{rules}(p) \backslash \{r\} = \emptyset) \wedge (\circ = \backslash))$ then
      $P = P \circ \{p\}$
      $MP = MP \circ \{p^{New}, p^{Temp}, p^{Red}, p^{Minus}, p^{Plus}, p^{Del}, p^{Red}, p^{Ins}\}$
      maintenanceRules = staticMaintenanceRules( $p$ )
    end if
    rules( $p$ ) = rules( $p$ ) $\circ$ r
    maintenanceRules = maintenanceRules $\cup$ dynamicMaintenanceRules( $r$ )
    ruleMaintenance( $r$ ) = ruleMaintenance( $r$) $\circ$ maintenanceRules
    $MR = MR\circ$ maintenanceRules
  end for
Ensure: Updated sets $MR, MP, R, P$ and maps $ruleMaintenance \ldots$
```

5 Evaluation

This section reports on our prototypical implementation and a first evaluation of our approach.

5.1 Implementation

We implemented our approach to materialization as an extension to the KAON Datalog engine. The implementation is freely available via the KAON web site[7].

In case of the materialization of a predicate all changes to facts relevant for the predicate and the rule set defining a predicate are monitored. The materialization and corresponding maintenance rules are updated as described in the previous sections.

The maintenance process is carried out as follows. When a program is designated for materialization, all maintenance rules are generated, the program itself is evaluated and the extension of all predicates designated for materialization is stored explicitly. The maintenance program is used for future evaluation instead of the original program. Therefore, all rules defining non-materialized predicates are added to the maintenance program.

Changes in facts are stored in the appropriate p^{Ins} and p^{Del} predicates. The change triggers the evaluation of the maintenance rules and consequently updates the extensions of all materialized predicates is updated by adding p^{Plus} and removing p^{Minus}. Afterwards the extension of extensional predicates is updated by adding p^{Ins} and removing p^{Del}. As a last step, the extension of p^{Ins} and all other auxiliary predicates is cleared.

Changes in rules are carried out by retrieving all rules defining the same predicates as the changed rules from the original program. These rules are then rewritten to refer to p^{Temp} only. Then the maintenance rules are evaluated and extensions are updated as described for facts. As a last step the auxiliary rules defining p^{Temp} are deleted.

5.2 Evaluation Setting

Test Assumptions The evaluation has been carried out with changing OWL ontologies that are compiled into logic programs via the DLP translation [6]. It is assumed that all predicates are materialized. We assume that an inference engine builds its knowledge base by aggregating data from several web sources. Therefore bulk updates will be predominant.

Test Procedure. Each test is characterized by a certain ontology structure and a class whose extension is to be read. The ontology structure has been generated for different input parameters, resulting in ontologies of different sizes. The average of five such invocations has been taken as the performance measure for each test. If the engine ran out of Memory results are denoted with OoM in the result table (cf. Appendix 1).

We obtain six measures: *(a)* the time of query processing without materialization, *(b)* the time required to set up the materialization and the maintenance program, *(c)* the time required to perform maintenance when rules are added, *(d)* rules are removed, *(e)* facts are added, and *(f)* facts are removed. Finally, *(g)* assesses the time of query processing with materialization.

Test Platform. We performed the tests on a laptop with Pentium IV Mobile processor running at 2 GHz, 512 MB of RAM using the Windows XP operating system. The implementation itself is written in Java and executed using Sun's JDK version 1.4.1_01.

[7] http://kaon.semanticweb.org/

5.3 Evaluation Scenarios

First we give an overview of the types of tests we conducted. In describing the results (cf. Appendix 1) we use *D* to denote the depth of the class hierarchy, *NS* to denote the number of sub classes at each level in the hierarchy, *NI* to denote the number of instances per class and *P* to denote the number of properties.

To test changes in facts, we add and remove a random percentage *Change* of the facts. For rules, we add and remove a random rule. This is due to the limitation of the underlying engine, which currently does not allow to alter rules in a bulk manner. The test was performed for different depths of the taxonomy *D* = 3, 4, 5 while the number of sub classes and the number of instances was not altered (*NS* = 5; *NI* = 5). Test 2 and 3 made use of properties. Here, every class had five properties, which are instantiated for every third instance of the class (*NI* = 5). We carried out each test using varying *Change* ratios of 10% and 15% of the facts.

Test 1: Taxonomy Extended taxonomies, e.g. WordNet, currently constitute a large portion of the ontologies that are in use. Our goal with this test is to see how the very basic task of taking the taxonomy into account when retrieving the extension of a class is improved. The taxonomy is constituted by a symmetric tree of classes. We did not make use of properties, hence *P* = 0. The test query involved computing the extension of one of the concepts on the first level of the class hierarchy. This is a realistic query in systems where taxonomies are used for navigation in document collections. Here, navigation typically starts with top-level classes and the set of documents is displayed as the class extension.

Test 2: Database-like The goal of this test was to see how ontologies with larger number of properties are handled. Our goal was to answer a simple conjunctive query on top of this ontology. The DL-like query is $\mathrm{c1} \sqcap \exists \mathrm{p0.c12}$.

Test 3: DL-like This test shows how materialization performs in DL-like ontologies, which contain simple class definitions. Each class in the class tree is defined using the following axiom: $\mathrm{c}_i \sqcup \exists \mathrm{p}_k.\mathrm{c}_{i-1} \sqsubseteq \mathrm{c}$ (where c_i denotes i-th child of concept c). The query retrieves the extension of some random class in the first-level of the taxonomy.

5.4 Results

Figure 3 depicts the average time[8] for querying an ontology without using materialization, setting up the materialization and cost of maintenance for different types of changes (adding and removing rules and facts). Finally, the time for answering the same query using the materialization is depicted. The exact results of the evaluation can be found in Appendix 1.

As we can see in the figure, maintenance costs do not vary significantly with the quantity of updates. All costs are directly related to the size of the ontologies. The performance behavior between the taxonomy and DB-like ontologies do also not alter significantly. However, more complex rules as they are constituted by DL-like ontologies

[8] in milliseconds on a logarithmic scale

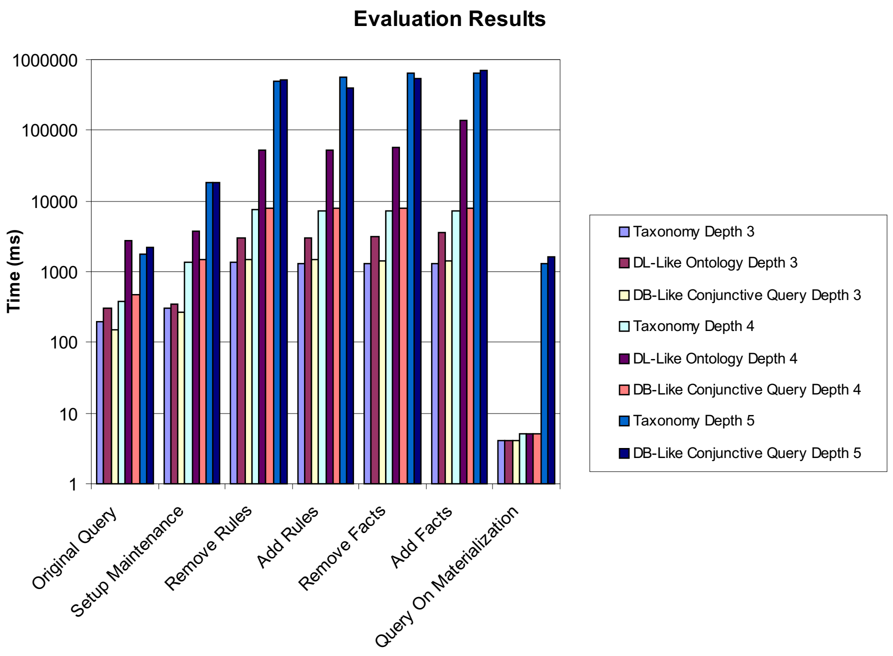

Fig. 3. Evaluation Results (Average Values)

are always more expensive to evaluate, therefore setup costs and the cost of evaluating the maintenance rules is also higher.

We want to stress that we measured the performance of concrete tools. Although algorithms implemented by a system are certainly important, the overall performance of a system is influenced by many other factors as well, such the quality of the implementation or the language. It is virtually impossible to exclude these factors from the performance measurement. For example, our Datalog engine ran out of memory with the DL-like ontology where the taxonomic depth was five, viz. the set of rules was generated from 3950 class and 19750 property definitions, while the underlying knowledge base contained 19750 class instantiations and 32915 property instantiations.

5.5 Discussion

The different costs of each step in the maintenance procedure are always higher than the costs of evaluating a single query. The question whether or not to materialize is therefore determined by the application and the issue whether the system can handle its typical workload, e.g. can it handle the intended number of users if answering a single query takes almost 3 seconds ?

With materialization the cost of accessing the materialized predicates is can be neglected. However, the time for the evaluation of the maintenance rules can be a significant bottleneck for a system especially for large knowledge-bases. For example, in one of our test runs it took almost 16 minutes to recompute the materialization after fact changes for the DB-like test with taxonomic depth 5. Fortunately, materialization can be carried out in parallel to answering queries on top of the existing materialization.

In consequence, users will have to operate on stale copies of data. Staleness of data cannot be avoided in distributed scenarios like the Web in the first place, and existing experiences, e.g. with outdated page ranks of a web pages in Google, show that the quality of query answering is still good enough, if data is updated occasionally.

6 Related Work

We can find related work in two areas: Firstly, incremental maintenance of materialized views in deductive databases. Secondly, truth maintenance systems in the Artificial Intelligence context.

Incremental Maintenance of materialized views Several algorithms have been devised for the incremental maintenance of views. All of these approaches do not consider changes in the set of rules and differ in the techniques used to cope with changes in facts. In order to cope with changing facts [1, 12] efficiently compute the standard model of a stratified database after a database is updated. The proposed solution of [1] uses sets of positive and negative dependencies that are maintained for all derived facts. This leads to low space efficiency and high cost for maintaining the dependencies. Another drawback of the approach is the granularity of the materialization which is the whole database.

[12] also derives rules (so-called meta-programs) to compute the difference between consecutive database states for a stratified Datalog program. The rules do not follow our three step principle and some of the generated rules are not safe making it impossible to implement the rules in typical Datalog engines. Additionally duplicate derivations are not discarded in the algorithm.

[7] present the DRed algorithm, which is a procedural approach to view maintenance in Datalog with stratified negation. We follow their principal approach in the for the computation of changes, in fact their procedural algorithm has been rewritten into the declarative version we use by [19].

The Progragation Filtration algorithm of [9] is similar to the DRed algorithm, except that changes are propagated on a predicate by predicate basis. Hence, it computes changes in one intensional predicate due to changes in one extensional predicate, looping over all derived and extensional predicate to complete the maintenance procedure. In each step of the loop the delete, re-derive and insert steps are executed. The algorithm ends up fragmenting computation and rederiving changed and deleted facts over and over again.

Truth Maintenance Systems Truth maintenance (also called belief revision or reason maintenance) is an area of AI concerned with revising sets of beliefs and maintaining the truth in the system when new information alters existing information. To this extent a representation of beliefs and their dependencies is necessary to achieve the retraction of believes and to identify contradictions. For example, justification-based TMS [5] uses a graph data structure where nodes are augmented with two fields indicating their belief status and supporting justification. When the belief status is changed, dependencies are propagated through the graph. Making TMSs more efficient was a cottage industry in the late 1980s, with most of the attention focused on the Assumption-based TMS [3]. The primary advantage of the ATMS is its ability to rapidly switch among many different contexts, e.g. it is simpler to propagate the withdrawal of facts, but this comes at the cost of an exponential node-label updating process. Disadvantages of TMS is that the set of justifications (and nodes) grows monotonically as it is not allowed to retract a justification, but only disable information. The fact that the set of assumption is always in flux introduces most of the complexity in the TMS algorithms. More recent work (e.g. [14]) primarily tried to reduce the cost for incremental updates. However, the underlying principle of labelling does not change. To the best of our knowledge, there is no TMS, where the aggregation of all historic information is avoided, viz. facts are permanently removed from the system. Additionally the primary technique deployed in TMS (backtracking) does not fit well with the bottom-up computation that is usually applied in deductive databases.

7 Conclusion

We have presented an incremental maintenance technique for the materialization of intentional predicates (views). Unlike previous approaches, our approach allows to change the set of rules in a stratified Datalog program. We have presented a preliminary performance evaluation which underlines the feasibility of our solution. We regard our results to be central to achieve scalability in large-scale Semantic systems such as presented by the Semantic Web. We have shown how our approach can be used with current means for specifying semantics in the Semantic Web. As we present a generic solution, future developments, e.g. for the rule layer of the Semantic Web, are likely to benefit from our technique as well.

Materialization is certainly not a panacea to all tractability problems. One drawback is that it trades off required inferencing time against storage space and access time.

In spite of such restrictions, we conjecture that materialization as explained in this paper will help to progress the Semantic Web and to build the large Semantic Web engines of tomorrow — the Semantic Web analogon to a syntactic Google.

Future work will address the maintenance when existential quantification is available in the rule language, such as in N3. This will involve maintaining skolem constants, which are used in the implementation of existential quantification. Additionally, we will investigate the obvious space-time trade-off between the solutions presented in sections 3 and 4 and non-materialized evaluation. This needs to be investigated quantitatively, including different options how to materialize: fully, by view indexes, with or without intermediate results, etc. Further, one might integrate algorithms that determine when modifications leave a predicate unchanged. This could be done for facts in style of [13] and for rules in style of [8].

References

1. K. Apt and J.-M. Pugin. Maintenance of stratified databases viewed as belief revision system. In *Proc. of the 6th Symposium on Principles of Database Systems (PODS)*, pages 136–145, San Diego, CA, USA, March 1987.
2. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not enough. In *SWWS-1, Semantic Web working symposium*, Jul/Aug 2001.
3. J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28(1986), 127-162.
4. S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL98 - Query Languages Workshop*, December 1998.
5. J. Doyle. A truth maintenance system. In B. Webber and N. J. Nilsson, editors, *Readings in Artfcial Intelligence*, pages 496–516. SMorgan Kaufmann, Los Altos, California, 1981.
6. B. Grossof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.
7. A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD Conference on Management of Data*, 1993.
8. Ashish Gupta, Inderpal Singh Mumick, and Kenneth A. Ross. Adapting materialized views after redefinitions. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 211–222. ACM Press, 1995.
9. John Harrison and Suzanne Dietrich. Maintenance of materialized views in a deductive database: An update propagation approach. In *Workshop on Deductive Databases, JICSLP*, 1992.
10. Patrick Hayes. RDF Semantics. W3C Working Draft, World-Wide Web Consortium (W3C), http://www.w3.org/TR/rdf-mt/, January 2003.
11. Matthias Jarke, Rainer Gallersdoerfer, Manfred A. Jeusfeld, and Martin Staudt. ConceptBase - A Deductive Object Base for Meta Data Management. *JIIS*, 4(2):167–192, 1995.
12. V. Kuchenhoff. On the efficient computation of the difference betwen consecutive database states. In Claude Delobel, Michael Kifer, and Yoshifumi Masunaga, editors, *Proc. of 2nd Int. Conf. on Deductive and Object-Oriented Databases*, volume 566 of *Lecture Notes in Computer Science (LNCS)*, pages 478–502, Munich, Germany, December 1991. Springer.
13. A. Y. Levy and Y. Sagiv. Queries independent of updates. In *Proc. of 19th VLDB*, pages 171–181, 1993.
14. P. Pandurang Nayak and Brian C. Williams. Fast Context Switching in Real-time Propositional Reasoning. In *Proceedings of AAAI-97*, 1997.
15. P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. Web Ontology Language (OWL) Abstract Syntax and Semantics. http://www.w3.org/TR/owl-semantics/, 2002.
16. Jos De Roo. Euler proof mechanism. Internet: http://www.agfa.com/w3c/euler/, 2002.
17. Michael Sintek and Stefan Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, June 2002.
18. M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. Technical Report AIB-95-13, RWTH Aachen, 1995.
19. Martin Staudt and Matthias Jarke. Incremental maintenance of externally materialized views. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 75–86. Morgan Kaufmann, 1996.
20. Guizhen Yang and Michael Kifer. FLORA: Implementing an Efficient DOOD System Using a Tabling Logic Engine. In *Computational Logic 2000*, pages 1078–1093, 2000.

A Evaluation Results

Test	D	NS	NI	P	Change	Orig Query			Setup Maintenance			Removing Rules			Adding Rules			Removing Facts			Adding Facts			Query Materialization
						Orig Average	Minimum	Maximum	Average	Minimum	Maximum	Average	Minimum	Maximum	Average	Minimum	Maximum	Average	Minimum	Maximum	Average	Minimum	Maximum	
Taxonomy	3	5	5	0	10	197	80	491	305	200	441	1386	1292	1592	1317	1252	1463	1302	1282	1332	1284	1262	1312	0
Taxonomy	4	5	5	0	10	373	290	571	1347	1212	1622	7581	7291	8352	7265	7240	7290	7328	7281	7361	7310	7270	7380	0
Taxonomy	5	5	5	0	10	1767	1482	2463	18391	16694	19318	494726	227747	717452	559407	393666	706696	631956	487551	759261	649123	522761	781173	1331
Taxonomy	3	5	5	0	15	147	60	311	245	141	251	1452	1292	1772	1286	1251	1332	1301	1291	1312	1367	1282	1612	0
Taxonomy	4	5	5	0	15	378	280	581	1382	1232	1683	7615	7330	8372	7308	7291	7331	7350	7340	7371	7310	7271	7350	0
Taxonomy	5	5	5	0	15	1765	1373	2464	18293	16714	19017	273874	189933	386005	464588	247826	611980	542294	381628	650265	648495	576319	756978	1252
DL-Like Ontology	3	5	5	5	10	310	170	640	355	230	531	2979	2864	3195	3009	2834	3345	3071	2974	3184	3620	3565	3685	10
DL-Like Ontology	4	5	5	5	10	2764	2523	3475	3715	2894	4747	52613	47128	65214	51864	47047	65444	56754	56371	57002	136128	134463	137928	0
DL-Like Ontology	5	5	5	5	10	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM
DL-Like Ontology	3	5	5	5	15	263	150	511	368	241	571	33128	3055	3555	3307	2884	3565	3242	3125	3355	3790	3725	3895	0
DL-Like Ontology	4	5	5	5	15	2774	2523	3515	3720	2894	4757	61979	50944	66395	61283	47528	67037	58339	58104	58655	90277	89940	90910	0
DL-Like Ontology	5	5	5	5	15	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM	OoM
DB-Like Conjunctive Query	3	5	5	5	10	152	70	341	265	151	431	1492	1382	1722	1469	1392	1662	1402	1392	1412	1399	1392	1402	0
DB-Like Conjunctive Query	4	5	5	5	10	482	310	701	1464	1322	1663	8011	7281	8732	8051	7801	8523	7991	7952	8022	7931	7761	8012	0
DB-Like Conjunctive Query	5	5	5	5	10	2165	19963	2403	18536	16935	19999	517994	284389	723009	400638	150226	541619	537931	299260	787843	714216	460882	878814	1633
DB-Like Conjunctive Query	3	5	5	5	15	172	70	430	272	160	440	1557	1422	1783	1462	1422	1552	1409	1382	1422	1434	1412	1452	0
DB-Like Conjunctive Query	4	5	5	5	15	425	301	701	1467	1352	1652	8112	7822	8723	7936	7902	7981	7876	7841	7901	8011	7891	8262	0
DB-Like Conjunctive Query	5	5	5	5	15	2078	1722	2374	18536	16905	20019	507760	132901	709009	484394	141163	691164	424565	292671	482925	763873	482964	955724	1282