ETL-aware Materialized View Selection in Semantic Data Stream Warehouses

Nabila Berkani
Ecole nationale Supérieure d'Informatique,
BP 68M, 16309, Oued-Smar, Alger, Algérie.
Email: n_berkani@esi.dz

Ladje Bellatreche
LIAS/ISAE-ENSMA
Futuroscope, Poitiers, France
Email: bellatreche@ensma.fr

Carlos Ordonez
Department of Computer Science
University of Houston, USA.
Email: ordonez@central.uh.edu

***Abstract*—For 25 years, several companies spent a lot of efforts and money in building warehouse ($\mathcal{DW}$) applications for data analytics purposes. This technology contributes to the success stories of several companies. Nowadays, companies are looking for real-time analytics for data issued from fresh data sources and external resources as knowledge bases and linked open data. The traditional life-cycle of designing $\mathcal{DW}$ applications has to be revisited to meet this requirement. Note that this life-cycle is composed of several well-connected phases. Integrating this requirement will seriously impact all phases in charge of data which are: ETL (Extract, Transform, Load) and the physical design phase, in which physical optimization structures are selected to speed up OLAP queries. In this paper, we propose a Near Real Time Data Warehouse design ($\mathcal{NRTDW}$) dealing with semantic data sources, with a particular focus on ETL and physical design phases. Firstly, we propose a dynamic materialized view selection method based on a workload of Sparql queries. Secondly, optimized algorithms are proposed to orchestrate the ETL flows considering the selected materialized views. Thirdly, an incremental view maintenance strategy re-computing only the graphs that involve the updated data sources is proposed. Finally, our findings are validated through an intensive experimentation using a detailed cost model on a real DBMS.**

I. Introduction

The data warehouse ($\mathcal{DW}$) and On-Line analysis have become a mature technology. This maturity is characterized by the presence of a well-known life cycle for designing business applications and widely adopted by companies. It includes the following phases [18]: (i) user requirement elicitation/validation, conceptual modeling, logical modeling, ETL (Extract, Transform, Load), deployment, physical design and exploitation. Each phase is associated to several features. In software development, a feature is a component added to the core body of software. Typically, features are added incrementally, at various stages in the life-cycle, usually by different developers [9].

It should be noticed that the phases of the $\mathcal{DW}$ life cycle are strongly dependent – the outputs of a given phase are the inputs of another one. To illustrate this point, let us consider two important phases of this life-cycle: ETL and physical design. ETL takes data from variety of sources, cleans, transforms it according the $\mathcal{DW}$ format and finally loads this data into the target warehouse [25]. It plays a crucial role in designing a $\mathcal{DW}$, because its quality strongly depends on ETL (*garbage in garbage out principle*). The ETL process is associated with different features incrementally added to deal with new requirements of data sources, decision makers and $\mathcal{DW}$ designers such as: (i) data quality [20], (ii) data cleaning [2], (iii) polystore deployment [23], etc. Recently, the ETL phase has been revisited in the context of the Connected World (e.g. the case of Internet of Things and Linked Open Data) under the name of *data ingestion* [22].

The physical design is a crucial phase for $\mathcal{DW}$. It is considered as the hinge point of the other phases and is the one which also conveys the image of the operational $\mathcal{DW}$ to its end-users. This is because it is usually associated to the non-functional requirement satisfaction (e.g. query response time, maintenance, energy consumption, etc.). During this phase, several features are considered such as materialized views, advanced indexes, data compression, data partitioning, etc. Each feature corresponds to a complex process to select its optimization structures. For instance, if the feature is materialized views, then algorithms for selecting these views are needed [4]. Usually, this selection is known as a NP-hard [4]. Note that this phase assumes that the $\mathcal{DW}$ is already deployed.

Note that the interaction among features (*intra-phase feature interaction*) of the physical design has been highlighted in [40]. This identification of this interaction contributes in developing intelligent tools to speed up OLAP query processing [40]. To illustrate this point, knowing that indexes and materialized views are both redundant structures (they duplicate data), compete for the same resource representing storage space and maintenance overhead allows replacing a materialized view by an index and vice versa for performance issue and constraint satisfaction (e.g. the storage cost). This interaction makes the problem of feature selection harder. More precisely, let us assume that for a given workload, the number of possible indexes and materialized views in N_I and N_V respectively. The combined search space could be as large as $2^{N_I+N_V}$. This is because different features could potentially interact with one another. We say that a phase P_i "strongly" depends on a phase P_j $(i \neq j)$ if a change in selection of P_j often results in a change in that of P_i. Otherwise, we say P_i "weakly" depends on P_j. DB2 Design Advisor of IBM has been developed by considering this interaction [40].

This complexity obliges researchers to tackle one phase in isolation way and without really considering the other ones, except the ETL phase. The particularity of this phase is that it has a strong interaction with conceptual, logical, deployment and physical phases. The first studies on ETL dealt with sources considering their physical and deployment phases [34], [28], [29], but without taking into account the process of selecting optimization structures of the physical design features. After that, ETL processes were focused on the logical level of data sources. In this perspective, [38] proposed an ETL workflow modeled as a graph, where its nodes represent activities, record-sets, attributes, and its edges describe the relationships between nodes defining ETL transformations. A formal ETL logical model is given using L_{DL} [27] as a formal language for expressing the operational semantics of ETL activities [36]. To make ETL processes more generic, a couple of studies consider it at the conceptual level of sources [37], [32], [16].

The interaction between phases (*inter-phase interaction*) has to be considered for when designing $\mathcal{DW}$ from new types of sources with high velocity. *We believe that this interaction will have the same impact on the quality of the $\mathcal{DW}$ design as the intra-phase feature interaction did.*

To illustrate this interaction, we consider the example of semantic warehouses ($\mathcal{SDW}$) that have emerged as an important demand for Linked Open Data [1], [35]. Note that streams are appearing more and more often on the Web[1]. The integration of new data, known as $\mathcal{DW}$ refreshment, is traditionally performed in off-line way. This means that during the update of data area with executed processes, OLAP users and applications cannot access any data. In fact, the set of ETL activities usually take place in a present loading time window in order to avoid the overloading of operational OLTP (On-line Transactional Processing) source systems with the extra workload caused by these processes. Still, decision makers are pushing for higher levels of freshness, since more and more enterprises operate in a business time schedule of 24x7 [2]. Designing a real time $\mathcal{DW}$ impacts automatically different phases of the life cycle, and more especially the phases that are in charge of data such as ETL and physical design.

In this paper, we *simultaneously* select, maintain materialized views (a feature of the physical design phase) and develop ETL tasks in the context of semantic $\mathcal{DW}$ from stream sources.

The rest of the paper is organized as follows: Section 2 overviews the main important studies related to the selection of materialized views and ETL in semantic warehouses. This section is concluded by a comparison with five relevant criteria of these studies. Section 3 describes our real-time processes for selecting materialized views and performing ETL processes. Mathematical cost models are also given in Section 4 to quantify the benefit our materializing or not a view. The performance evaluation results are presented in Section 5. Finally, Section 6 concludes the paper and discusses future work.

[1]http://activitystrea.ms/

II. Related Work

This section overviews the main important studies tackling the ETL and the selection of materialized views in an isolated way in the context of Semantic $\mathcal{DW}$.

Contrary to traditional $\mathcal{DW}$ [15], little attention has been paid to the problem of materialized view selection for semantic databases and $\mathcal{DW}$. [24] propose an RDF-3X engine system which implements a solution that stores and builds indexes for RDF triples. [11], [10] propose a system called RDFMatView, inspired from traditional databases [12]. It allows RDF view selection from triple table for a given SPARQL workload and creates indexes using patterns of shared triples to speed up queries. The authors of [13], [14] propose a cost-based approach to select materialized views from a well-known workload. They propose an algorithm to identify a set of candidate views for materialization. They take in account implicit triples entailment and supports query rewriting. In [21], authors propose to select materialized views at the ontological level using a rule-based approach hiding the implementation aspects. In a second step, they use Directed Acyclic Graph (DAG) to select views at the logical level based on a cost model which considers the diverse storage layouts.

All these approaches have in common that they focus on facilitating efficient processing of RDF queries and updates. However, they consider a static workload which contradicts the dynamic nature of the web. In fact, any change to the workload should be reflected to the view selection as well.

The studies dealing with real time ETL considering optimization structures concern mainly the traditional $\mathcal{DW}$. The work of [26] dealt with the problem of elimination of duplicate data by the means of equality and similarity features. The authors of [17] used a queue network, where an Active Data Staging Area (ADSA) is built between the sources and the $\mathcal{DW}$. The ETL transformations are done by choosing the right topology and communication methods. In [33], the authors dealt with the ETL workflow optimization that includes a module that automates the job allocation which allows content enrichment using indexes defined on the table. [6] focused on the speed arrival of stream data and the Inputs/Outputs. They proposed some algorithms that manages disk and memory access and uses MESH join operations in transformation phase that is faster compared to other joins. In the same perspective, [29] have proposed an algorithm that uses Semi-Streaming Index Join (SSIJ) during join operations in dynamic nature, it manages the memory and the incoming stream. To the best of our knowledge, there are no works that have dealt with Dynamic materialized view selection during Semantic ETL process.

A comparison including existing works and our proposal is given in Table I based on five criteria:

1) the nature of the workload queries (NQW),
2) the data structure used to manipulate input queries (DS).
3) the reordering of queries depending on requirement selection (RQ)
4) Type of Optimization structure used (OS)

5) The use of Materialized view results (UMVR).

III. Dynamic Views Selection during Semantic $\mathcal{ETL}$ process

This section presents our approach that selects dynamically RDF views during the ETL process. Before detailing it, we present a motivating example that illustrates our proposal.

Given a set of RDF data sources constructed based on the schema of Lehigh University Benchmark[2](LUBM) and integrated on a semantic $\mathcal{DW}$ deployed on an RDF Quad table (Fig. 1-a), i.e. triple plus graph per row. The columns are G for graph, P for predicate, S for subject and O for object. Let us assume that a materialized view V is selected for this $\mathcal{DW}$ (Fig. 1-d). The real-time execution of ETL transformations [30] such as extract, merge, join, aggregate, etc. uses significantly this view. More concretely, let us consider the SPARQL query (Fig. 1-b) generated by the join transformation of ETL represented by an RDF graph (Fig. 1-c). Two scenarios are possible to execute this query: (i) directly from the $\mathcal{DW}$ this scenario is very costly in terms of disk accesses and the join execution time and (ii) directly from the view V, which savings these above costs, especially the selection of this view is performed dynamically to reflect the real situation of the $\mathcal{DW}$.

A. Description of our Methodology

Our methodology, illustrated on Fig. 2, consists of three main components: (1) the main memory and cache management; (2) a dynamic materialized view selection and (3) a stream $\mathcal{ETL}$ process.

1) *The main memory and cache management:* ensure the management of the main memory and the cache which are necessary to the execution of the ETL process. This component is responsible of the memory usage during the $\mathcal{ETL}$ process. The main idea is to use main memory as primary storage for the ETL transformations and materialized views, removing disk access as main performance bottleneck. Let M be the quota of the memory. It is then divided into four partitions, with a variable size: (1) the buffer of inputs issued from sources (BI), (2) the buffer of streams (BS), (3) the view cache (VC) and the cache memory (CM). Briefly, the scenario is as follows: The BI receives extracted instances from sources. The BS is similar to the traditional buffer of query optimizers. It stores data from the warehouse to speed up some ETL transformations to avoid unnecessary accesses to the $\mathcal{DW}$. The VC represents the pool of selected views that are managed dynamically in terms of allocation and eviction of views. MC Memory cache is the area where the transformations are performed.
2) *The dynamic materialized view selection component:* identifies the most relevant materialized views that matches with requirement and houses them in VC. Note that VC contains a set of views dynamically selected during the ETL process. Views which are no longer used are evicted from memory.
3) *Stream $\mathcal{ETL}$ process component:* starts by extracting instances from RDF data sources and temporarily backs up in the BI. The BI merges the instances that do not requires data from $\mathcal{DW}$ to the execution of the ETL process and load them in the $\mathcal{DW}$. The BI houses in the BS the remaining instances (instances requiring data from the $\mathcal{DW}$ to the execution of the ETL process). BS deals with retrieved instances (Triple/quad blocks) from sources and data recovered from $\mathcal{DW}$ disk or VC, needed for the execution of ETL process. Finally, ETL operations are executed in CM.

B. Materialized view selection process

In this section, we propose a solution that evaluates SPARQL queries in order to dynamically selects materialized views. We formally introduce all necessary concepts for this idea and describes the different steps, which are: (1) identifying mappings between SPARQL queries, (2) capturing of interaction among queries, (3) generation of views candidate and (4) query scheduling.

a) Identifying mappings between SPARQL queries: During this step, the SPARQL queries are placed in a queue. To reduce the search space of the problem of selecting semantic materialized views, we identify similarities and equivalences that may exist between queries. Firstly, we start by considering variables from the SPARQL queries in order to identify query containment. A such strategy requires finding mappings between elements. More concretely, it is done by the identification of a pattern contained in another query pattern. To do so, we develop an algorithm that finds all mappings between each query and all patterns by enumerating all of them. Afterward, identify the most frequently used mappings and we select the relevant ones.

b) Capturing of interaction among queries: The process of materializing views in traditional $\mathcal{DW}$is usually guided by the interaction between queries. To capture this interaction, several data structures were proposed such as Multiple View Processing Plan (MVPP) [39]. The MVPP is a directed acyclic graph. The roots of this graph are the queries and the leaves are the base relations, and the intermediate nodes are the operations of selection, projection, join, or aggregation of data. The structure of MVPP is constructed by unifying single query processing plans of queries. Fig. 3 shows an example of a MVPP constructed from the Star Schema Benchmark (SSB) queries[3]. The nodes Lo, Pa, Sp and Da represent respectively the table of SSB: *Lineorder, Part, Supplier*, and *Date*. Several types of Intermediate nodes are distinguished: join (J_i), aggregate (A_j), selection (S_k) and projection (P_l). This structure has been used in our system SLEMAS [7] that captures the interaction among very large number of queries in the *context of relational data warehouses*. SLEMAS is available at the forge of our laboratory[4].

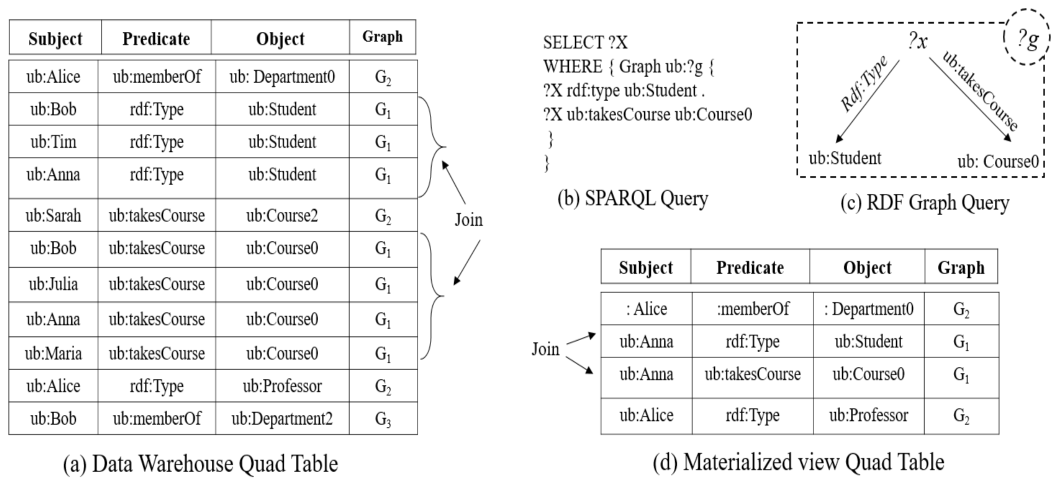
[2]http://swat.cse.lehigh.edu/projects/lubm/

[3]https://www.cs.umb.edu/~poneil/StarSchemaB.PDF

[4]https://forge.lias-lab.fr/projects/bigqueries

Criteria \ Work	Neumann and al[24]	Castillo and al[10]	Goasdoue and al[14]	Bery and al[21]	Our Proposal
NQW	Static	Static	Static	Static	Dynamic
DS	Triples	Triples	multigraph	DAG	Hyper-graph
RQ	No	No	No	No	Yes
OS	MV on Indexes	Indexes	Static views	Static MV	Dynamic views
UMVR	No	No	No	No	ETL Process

Table I. *Comparison of RDF Materialized View Selection approaches*



Subject	Predicate	Object	Graph
ub:Alice	ub:memberOf	ub: Department0	G_2
ub:Bob	rdf:Type	ub:Student	G_1
ub:Tim	rdf:Type	ub:Student	G_1
ub:Anna	rdf:Type	ub:Student	G_1
ub:Sarah	ub:takesCourse	ub:Course2	G_2
ub:Bob	ub:takesCourse	ub:Course0	G_1
ub:Julia	ub:takesCourse	ub:Course0	G_1
ub:Anna	ub:takesCourse	ub:Course0	G_1
ub:Maria	ub:takesCourse	ub:Course0	G_1
ub:Alice	rdf:Type	ub:Professor	G_2
ub:Bob	ub:memberOf	ub:Department2	G_3

(a) Data Warehouse Quad Table

```
SELECT ?X
WHERE { Graph ub:?g {
?X rdf:type ub:Student .
?X ub:takesCourse ub:Course0
 }
}
```

(b) SPARQL Query

(c) RDF Graph Query

Subject	Predicate	Object	Graph
: Alice	:memberOf	: Department0	G_2
ub:Anna	rdf:Type	ub:Student	G_1
ub:Anna	ub:takesCourse	ub:Course0	G_1
ub:Alice	rdf:Type	ub:Professor	G_2

(d) Materialized view Quad Table

Figure 1. *A Motivating Example*

This MVPP has been revisited in the context of SPARQL queries [19] to generate a graph structure called, unified query plan (UQP). This motivates us to adapt our SLEMAS system to SPARQL queries.

This adaptation is performed as follows: (i) transforming the UQP to a hypergraph; (ii) partitions this hypergraph into several components. This partitioning is performed using HMETIS tool[5]. Each component contains a query plan that shared at least one join node. The first join node, called pivot node is shared by all queries of its component. The pivot has a direct impact on the other nodes of its component. (iii) After the partitioning process of the hypergraph into several small sub-hypergraps, the generation of UQP becomes a simple transformation of each sub-hypergraph into an oriented graph and (iv) finally, we merge the resulting graphs. Fig. 4 describes an example of query interactions applied to the 14 LUBM queries. The hyper-graph is created and partitioned according to the existing interactions among queries. Each partition is transformed to an acyclic directed graph. The result obtained is a unified query plan having as leaf nodes the form of quads $< g, s, o, p >$. The root node represents the final query result and the intermediate nodes are the SPARQL algebra operators (such as join, filter, etc.) and solution modifiers (such as projection, distinct, limit, or order by).

c) Materialized Views candidates: Materializing all nodes of the global plan is not feasible [8]. To decide which nodes have to be materialized, we need to evaluate their impact on overall query processing. This decision is performed using a cost model that estimates the query processing. It defines a benefit function returning a value between 0 and 1. The cost function is given as follows:

$benefit(V) = cost_{WO}(q, V) - cost_{WV}(q, V) - cost_{Mat}(V)$, where:

- $Cost_{WO}(q)$: the processing cost of the query q without view(s).
- $Cost_{WV}(q, V)$: the query processing cost of query q using the materialized view V.
- $Cost_{Mat}(V)$: the maintenance cost of the view V.
- $Size(V)$: the cost needed to store the view V.

Instead of treating the whole search space including all candidates as in the usual approaches for materializing views, we adopt the divide-conquer approach [7], where the search space is divided into several sub search spaces, each one corresponds to a connected component of the global graph.

d) Query re-ordering and materialization: The ordering of queries is done in order to avoid view dropping. All materialized views should optimize the maximum of queries before theirs dropping, following these steps: (1) Identify all nodes having maximal benefit; (2) An ordering is applied based on their benefit; (3) The benefit identified is propagated to the queries of those nodes. At this stage, each query will have a weight representing the sum of the benefit of its nodes. (4) the ordering of queries is done based on these weights and a decision is taken on materializing or de-materializing views.

[5]http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

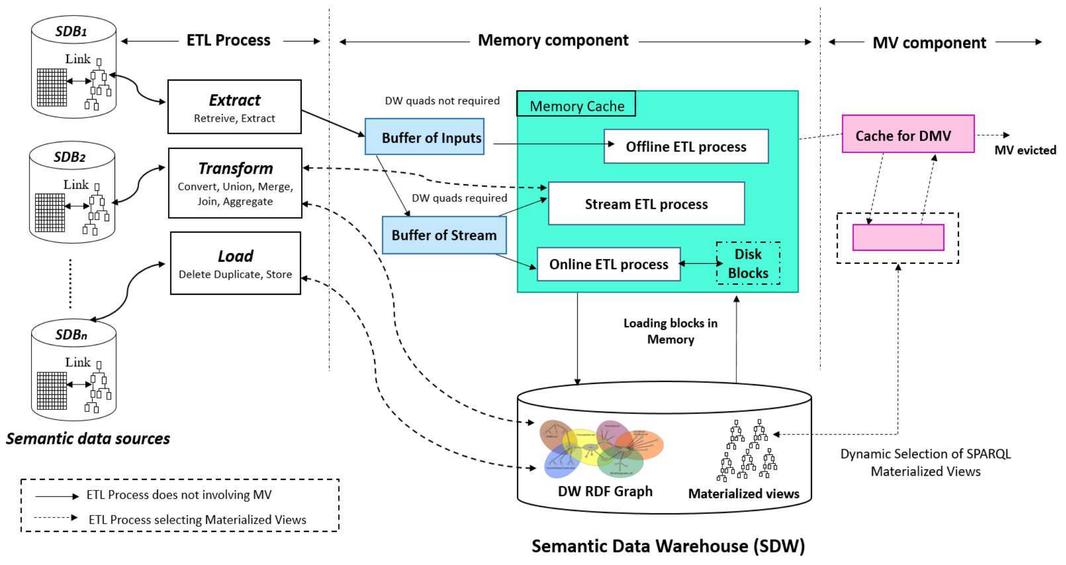


Figure 2. *Our general solution : $\mathcal{NRTDW}$ architecture.*

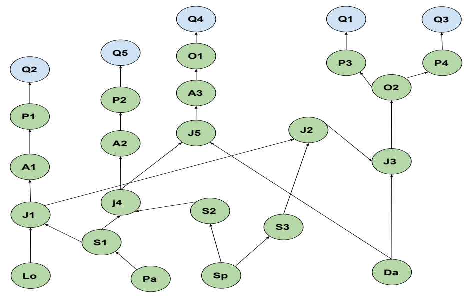


Figure 3. *Example of a MVPP.*

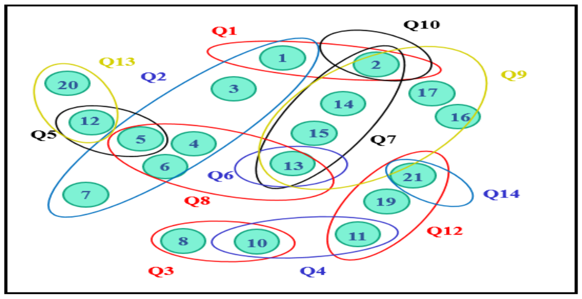


Figure 4. *Example of query interactions using the 14 SPARQL Queries of LUBM.*

C. Near Real Time Semantic Data Warehouse

In this section, we describe the different phases of the proposed real time ETL process for Semantic $\mathcal{DW}$. We consider as starting point some RDF graphs defining data sources and views selected dynamically. The main goal is then to define an appropriate set of rules, determining how the flow of ETL operations from sources to the target $\mathcal{DW}$ nodes can be executed (Fig. 5). Essentially, each rule is responsible for executing an operator in the ETL process. The obtained RDF graph represents the $\mathcal{DW}$ resulting from the integration of data sources.

Our ETL process uses 10 generic conceptual ETL operators defined in [30]. We have overloaded them to consider the characteristics of RDF graphs. We redefined their signatures in order to satisfy our requirement, which is manipulating the RDF data structure. The data sources and target $\mathcal{DW}$ are represented by the RDF graph (G), where the nodes (N), edges (E) and labels (L) represents respectively classes, instances and data properties, object properties and **DL** constructors.

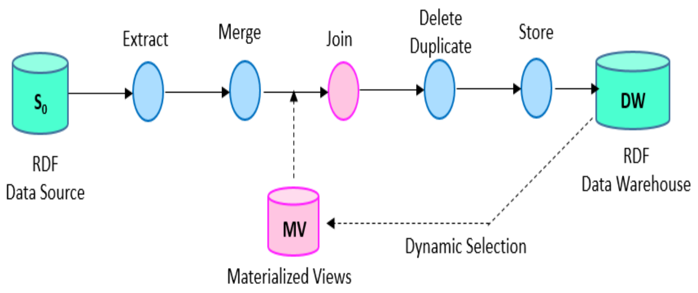


Figure 5. *An example of ETL flow generated using the stream ETL algorithm.*

We consider three phases, namely: (1) *Pre-processing phase* which deals with transformations that do not require $\mathcal{DW}$ instances; (2) *Stream phase* that applies transformations requiring instances stored in materialized views; and (3) *Off-line phase* which manages transformations requiring $\mathcal{DW}$ instances, available only in the disk. The maintenance of the selected views is also performed in this phase.

Pre-processing phase. During this phase, our algorithm starts by extracting instances in the format of RDF graphs (quad-sets) from the data sources participating in the integration process. These data are moved to the *BI* for a pre-processing. Extracted instances that do not require $\mathcal{DW}$ instances for ETL transformations are moved to *memory cache zone* for transformations and loading in the $\mathcal{DW}$. The remaining instances are moved to *BS* for an on-line processing. The ETL operators concerned by this processing are: retrieve, extract, convert, merge, filter, Delete Duplicate (DD) and Store.

Algorithm 1 Pre-processing ETL Algorithm

Input: S_i: Data sources (Semantic and Graph structure), $\mathcal{DW}$: schema + instances (RDF Graph structure).
Output: $\mathcal{DW}$ as RDF graph.

```
1: c := 0;
2: for Each $S_i$ do
3:     read stream graphs;
4:     $g_{S_i}$:=ExtractGraph($S_i$);
5:     Put $g_{S_i}$ in InputBuffer;
6:     for Each graph $g \in$ InputBuffer do
7:         if data does not need $\mathcal{DW}$ instances then
8:             Merge operations and output in $G_{ETL}$;
9:             Clean and Store instances in DW;
10:        else
11:            if (END_OF_STREAM is true) then
12:                Move subgraph to StreamBuffer zone;
13:                Increment counter $c$ with number of stream sub-graph;
14:            else
15:                Go To Stream ETL Algorithm ;
16:            end if
17:        end if
18:    end for
19:    Remove graph g from cache InputBuffer;
20: end for
```

Streaming phase (Online phase). In this phase, the algorithm processes the existing RDF graphs (quad-sets) in the *BS*. It waits for a minimum of graphs threshold to accumulate (until to receives the message END_OF_STREAM). Then, it applies the transformation operations that requires instances from the $\mathcal{DW}$. Indeed, the algorithm checks the availability of the instances precomputed in views already materialized; ie join and aggregation operations precomputed through the materialized view dynamically selected. Otherwise, it moves to offline phase where required instances will be extracted from the disk and then computed. The ETL operators concerned by this processing are: Union, Join, Aggregate, Store.

Offline phase. In this phase, the algorithm deal with existing RDF sub-graphs (quad-sets) located in *Memory Cache*. The concerned transformations involve expensive joins between the newly arrived instances and some warehouse data (not available in views). So, it selects instances required from disk. Then, it applies the transformation operations. Finally, the materialized view component is refreshed in order to select and/or evict new materialized views.

Algorithm 2 Stream ETL Algorithm

Input: Stream Buffer zone, $\mathcal{DW}$ (schema and instances), number of stream sub-graph, value_limit (max memory size of stream buffer).
Output: $\mathcal{DW}$as RDF graph.

```
1: $c$:=number of stream sub-graph;
2: if $c$ reach value_limit then
3:     Send End_of_stream =false;
4: else
5:     for Each graph $g_i$ in StreamBuffer do
6:         check the availability of instances required in MV;
7:         if Materialized view available then
8:             Apply ETL transformations (join, aggregate, Union);
9:             Output the $G_{ETL}$ after apply ETL operator;
10:            Clean and Store instances in DW;
11:            Remove graph g from cache StreamBuffer;
12:        else
13:            Move subgraphs to in-Memory zone;
14:            Go to Offline Phase Algorithm;
15:        end if
16:    end for
17: end if
```

Algorithm 3 Offline ETL Algorithm

Input: in-Memory zone, $\mathcal{DW}$ (schema and instances).
Output: $\mathcal{DW}$as RDF graph.

```
1: for Each graph $g_i$ in in-MemoryCache do
2:     Read required blocks from disk
3:     Apply ETL transformations (join, aggregate, Union)
4:     Output the $G_{ETL}$ after apply ETL operator;
5:     Clean and Store instances in DW;
6:     Remove graph g from cache in-MemoryCache;
7: end for
8: Refresh Materialized view selection (selects/evicts);
```

D. Incremental Maintenance of RDF Materialized views

We distinguish two types of views updates, depending on whether the initial $\mathcal{DW}$ is empty. Either perform a full materialization of RDF graphs by running the ETL process during the creation of $\mathcal{DW}$ or carry out an incremental materialization gradually as the data sources evolution.

Our system performs the maintenance of materialization incrementally. The process can be divided into four main steps: (i) Identify the updated RDF graphs (quad-sets) from sources using change data capture techniques; (ii) Load updated quad-sets into the staging area which is stored in the main memory component, specifically in the buffer of inputs (BI); (iii) apply the transformations needed in order to derive new quad-sets depending on $\mathcal{DW}$ schema (the execution of the previously described ETL process); and finally (iv) add all the new derived quad-sets into the materialized views, making them available for querying.

The identification of updated quads using Change data capture, involves the generation of a set of rules required for the process of maintenance views. It depends on the identification of the set of all relations R in sources S that are relevant to materialized views. A relation R is relevant to a materialized view (MV), if any of these three rules are generated: (i) R is triggered by deletions of quads on S; (ii) R is triggered by insertions of quads on S (iii) R is triggered by updates of quads on S. An update is treated as a deletion followed by an insertion. The following is an example of Oracle trigger implemented in each data source participating in the integration process.

```
Create or replace
TRIGGER Updates_On_Sources(S(i))
BEFORE INSERT OR
UPDATE ON university_rdf_data
BEGIN
Identify and extract the set of quads
from sources
U:= INSERT_ON_Source(S(i), new_quads);
Updates the API to be executed on MV
Apply_Updates_On_Views(MV, U);
END;
```

IV. Cost Model

In this section, we derive the general formula for calculating the cost of our ETL solution. The main objective is to evaluate the memory usage and processing costs. This is possible by interrelate the key parameters of the ETL Algorithm: input size *n*, ETL cost to process *n* quads and available memory *M*. Table II describes the notation used.

Generally, the streaming ETL process requires that the effect of a stream graphs is visible in the $\mathcal{DW}$ store (output) before a given time limit. This time limit is associated with each stream graph in the form of a deadline. In addition, the whole process has to meet the deadline of its time window for completing the integration of data sources with minimum cost process and memory. We assume that the deadline and time window are given as part of the technical specifications for the ETL design. It includes also the expectations about the size of the input RDF quads.

Let d and w be respectively the deadline for each stream RDF quad and the time window for the execution of the whole ETL process. The time to run the ETL process for n quads must be less than the time window w, knowing that each quad must meet the deadline d to be visible in the $\mathcal{DW}$. Consequently, the objective function for ETL optimization is to satisfy the following constraints:

$$Time(ETL(n,t)) \leq w / \forall t \in DW : time(ETL(t)) \leq d \quad ....(1)$$

Memory cost. As we said before the memory is divided into four components. We assume that the total memory allocated (M) is used for the execution of the ETL Algorithm. The memory for each component of the proposed solution can be calculated as follows:

- Memory reserved for the waiting queue (Buffers of Inputs and Streams) = B_{input} + B_{stream}.
- Memory for the basic ETL Transformations (e.g., merge, union, convert)= $\alpha \times C_{mem}$.
- Memory for the join and aggregation ETL transformations = (1-α) (C_{mem}).
- Memory for the materialized views selected = C_{mv}.

The total memory used by the proposed solution is computed as follows:

$$M= B_{input} + B_{stream} + \alpha\ C_{mem} + (1\text{-}\alpha)\ C_{mem} + C_{mv} \quad .... (2)$$

Processing cost. Here, we first calculate the processing time cost for each RDF triple/quad separately, then we calculate the processing cost of the proposed ETL Algorithm for *n* quad-sets. The total processing time cost of the proposed algorithm:

$$Cost_{ETL} = (n_{disk} \times C_{I/O}) + C_{MV} + n(C_{I/O} + C_{Del} + C_S).$$

The algorithm processes *n* stream quads. On this basis, the service rate μ can be calculated by dividing n by the cost of ETL Algorithm. With the high arrival stream quad-sets n and limited memory the algorithm tries to achieve a high service rate. The service rate calculation formula is given as follows: $mu = \frac{n}{Cost_{ETL}}$.

Parameter name	Symbol
Service rate	μ
Total allocated memory (bytes)	M
Buffer of Inputs (bytes)	B_{input}
Buffer of Streams (bytes)	B_{stream}
Memory Cache (bytes)	C_{mem}
Materialized View Cache (bytes)	C_{mv}
Size of disk instances (triple/quad) (bytes)	$S_{instance}$
Memory weight for Basic ETL operations	α
Memory weight for Join&Aggregation operations	$1-\alpha$
Number of Input from disk	n_{disk}
Cost to read one bloc from disk into the Memory	$C_{I/O}$
Cost to remove one triple/quad from the Memory	C_{Del}
Cost to read one stream quad from source	C_S
Cost to handle one ETL transformation	C_T

Table II. *Notations used in cost estimation*

V. Experimental Study

In this section, we show the results of our experiments. We first conduct an experimental study to analyze the effectiveness of the optimization approach proposed. Then, we evaluate the loading and query performance in comparison with our previous work [5].

A. Platform and data layout.

We use Oracle Semantic Database 12c release 2 as the database back-end. For efficiency, we implemented a B-tree indexes on quad Table and SPARQL query hints. The storage of data ix done on quad table, using a distinct integer for each distinct URI or literal value. In order to enhance the efficiency of frequently queries, we have indexed the quad table on *s*, *p*, *o*, *g*, and all two and three column combinations.

Moreover, some PL/SQL APIs were invoked after the integration of each data source (load of instances). The API SEM_PERF.GATHER_STATS allow collecting statistics for data sources and the API SEM_APIS.ANALYZE_MODEL for $\mathcal{DW}$ models in the semantic network graph. The memory SGA and PGA are also increased to 2GB.

In addition, Oracle has incorporated a reasoner engine defined based on *TrOWL* and *Pellet* reasoners. Oracle provides full support for native inference in the database for RDFS, RDFS++, OWLPRIME, OWL2RL, etc. It uses forward chaining to do the inference. It compiles entailment rules directly to SQL and uses Oracle's native cost-based SQL optimizer to choose an efficient execution plan for each rule. The following is an example of user defined rules applied, they are saved as records in tables. (a) Co-author rule: PublicationAuthorOf(?A1, ?P) $\wedge$ PublicationAuthorOf(?A2, ?P) $\rightarrow$ CoAuthor(?A1, ?A2) and (b) the validation of Co-author : publicationAuthorOf(?A1, ?P) $\wedge$ publicationAuthorOf(?A2, ?P) $\wedge$ notEqual(?A1, ?A2) $\rightarrow$ CoAuthor(?A1, ?A2)

Data and workload. Our experiments are based on YAGO KB, version 3.0.2, having an architecture classified on themes. Each theme is a set of facts. A fact is an RDF graph. YAGO has defined the context relation between individuals [31] which we used to extract the set of themes related to our context study (university domain). The resulting contextual YAGO KB contains around 5,9 x 10^6 quads. From this set, we have generated five data-sets, representing data sources (SDB), equivalent to our previous work [5] in order to make a comparison. The five SDBs and the $\mathcal{DW}$ schema have been deployed using Oracle DBMS. Oracle offers different format for data loading such as: RDF/XML, N-TRIPLES, N-QUADS, TriG and Turtle. We choose N-QUADS format to load instances using Oracle SQL*Loader.

The LUBM benchmark is an ontology for the university domain. It offers fourteen extensional queries representing a variety of properties. We have added 6 more queries inspired from [3] defining analytical SPARQL queries (using count, avg, sum, Group By, etc.) and computing number of courses offered by departments, number of courses taught by professors in each department, number of graduate courses in each department, etc.

Hardware. Our evaluations were performed on a laptop computer (HP Elite-Book 840 G2) with an Intel(R) CoreTM i5-5200U CPU 2.20 GHZ and 8 GB of RAM and a 500 GB hard disk. We use Windows10 64 bits. We use Oracle Database 12c release 2 that offers RDF Semantic Graph features of Oracle Spatial and Graph. Cytoscape[6] is used for visualization.

Now, we report the results of our experiments: (i) a validation of the dynamic materialized view selection method; (ii) an analysis of the performance of the ETL process; and (iii) a comparison between our proposal and the work of [5].

B. *Validation of the dynamic $\mathcal{MV}$ method.*

In order to evaluate the efficiency of our algorithm for selecting materialized views, we first check the scalability of

[6]http://www.cytoscape.org/

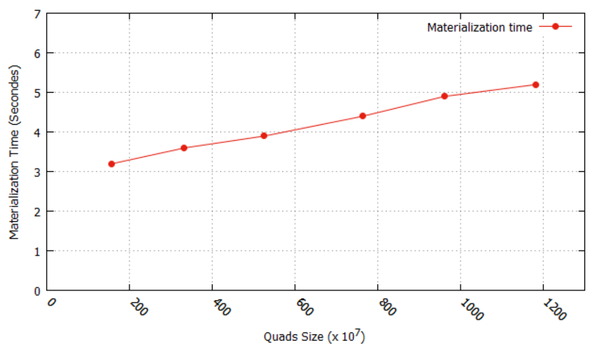


Figure 6. *Execution time(s) of some LUBM queries over raw data and views.*

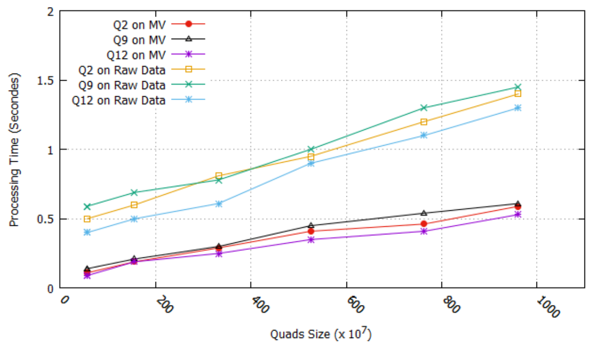


Figure 7. *Materialization time vs. Quad size.*

the approach. We create larger RDF graphs such that the size of materialized views would be multiplied by a factor of 1 to 5, with respect to the different steps explained above. The corresponding materialization time is shown in Fig. 6 which demonstrates a linear scale-up w.r.t. the data size.

Moreover, we have run the fourteen queries of the LUBM bench on the $\mathcal{DW}$ before and after the execution of the materialized view selection algorithm. Fig. 7 describes the runtime of some queries over raw data and views, for different sizes of the $\mathcal{DW}$. It shows the performance of queries over materialized views that becomes more evident with the growth in the volume of data, due to the dynamic selection of the appropriate views. Moreover, evaluating queries using materialized views is on average 4 times faster than raw data. This finding can be explained by the availability of partial results from these materialized views.

C. *Performance of the ETL process.*

In this experiment, we evaluate the ETL algorithm which considers the different possible cases namely: preprocessing, streaming and off-line. For each phase, we evaluate the response time of the ETL process with and without memory cache. Note that a memory cache is allocated for the preprocessing phase (buffer cache) and for the stream phase (Stream cache and materialized view cache). Fig. 8 depicts the results obtained. The dynamic selection of materialized views greatly

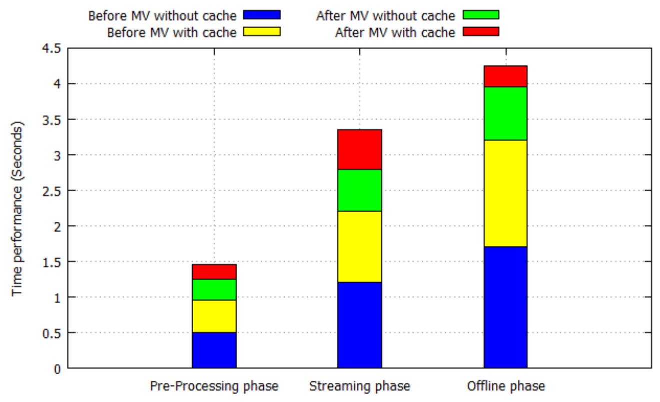


Figure 8. *Query response time before and after MV selection (with cache and without cache).*

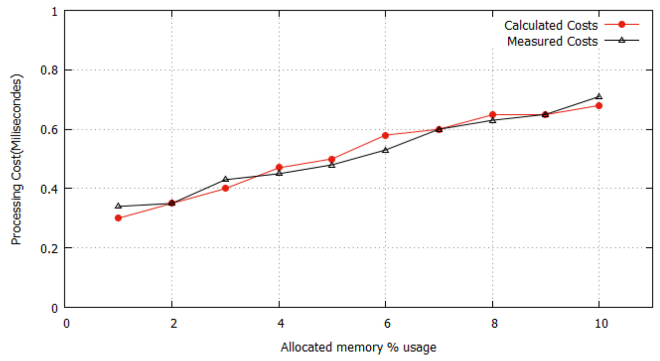


Figure 9. *Cost validation.*

improves the response time of ETL process. The use of the cache as well. Indeed, the recently accessed streaming data and materialized views selected are cached in the memory respectively in Stream buffer and materialized view cache. The use of off-line phase shows less performance caused by an extra inputs-outputs (I/O) operations from disk-based access.

a) Cost validation: In order to validate our results, we have compared between our mathematical cost models and the real measures. Fig. 9 presents the comparisons of both costs for the proposed ETL algorithm. This Figure shows that the predicted cost closely matches the measured one. This finding demonstrates the quality of our proposed cost models.

b) Inference performance: We evaluate inference performance using OWLPrime fragment and user defined rules. We used reasoner mechanism to infer instances from integrated data. Table III shows results obtained. It clearly demonstrates that number of quad inferred is important when it comes to use contextual $\mathcal{KB}$ instances. It includes a new dimension and thus allows more graph analysis.

Criteria	Inference Results
Integrated instances	5,4 x 10^6
Inferred instances	34K
Time inference(minutes)	5,4

Table III. *Inference performance : Time and number of Quads.*

D. Comparison between our proposal and the previous work [5].

Our experiments demonstrate the feasibility of our approach, based on the standard RDF. We showed a scalable performance during materialized view creation and integration of data sources. Fig. 10 demonstrates a comparison between the two approaches on the basis of time(s) performance of instances loaded in DW per concept. It clearly indicates that our proposal based on dynamic materialized view selection outperforms the [5] approach.

On the other hand, we have studied the ETL Algorithm and we were interested on the time complexity. The algorithm is implemented based on the graph theory, where nodes represent concepts and edges roles definitions. We have examined the number of iterations of our algorithm to generate the ETL graph (semantic DW populated) and we have compared it with the state of the art [5]. The algorithms are based on concepts searches (Tbox for intentional mappings) and not instances (Abox for extensional mappings). The time complexity is $O(n)$ for both algorithms, where n represents the number of involved nodes (concepts). This depends on the resolving of constraints defined on data sources, which are at least $O(m)$, where m is the number of involved schemes. Fig. 11 describes the number of iterations per concepts involved in the $\mathcal{DW}$ schema. It indicates a polynomial time. This finding proves the feasibility and efficiency of our approach and shows better results in terms of reduced number of iterations thanks to the dynamic selection of materialized views. This is explained by the decrease in the number of joins and aggregations.

Table IV demonstrates a comparison between the fast approach for selecting materialized views for integration and offline integration done from scratch. On the basis of some criteria identified during the experiment, the results clearly indicate the interest to move towards the selection of materialized views during the ETL process to enhance the integration time. Note that time is given in minutes.

Criteria	Offline [5]	Our Proposal
Optimized Structure	No	Dynamic views
Memory/Disk Usage	Disk	Memory
Data type	RDF N-Triples	RDF N-Quads
Deadline	≥ 5	≤ 1
Data Loading	from scratch	Incremental
Query execution time	$4 \times n$	n
Overall Integration Time	3,2	1,1

Table IV. *Comparison between Our proposal and previous work.*

Our experiments demonstrate the feasibility of the dynamic selection of materialized views during the ETL process, which exploits standard RDF functionalities offered by Oracle such as: quad storage, graph definition and reasoning. We showed scalable performance when loading and integrating data sources. We proved that semantic $\mathcal{DW}$ is enriched with concepts and instances deducted from $\mathcal{KB}$ using inference mechanism. This will give more possibility for query answering and data analysis.

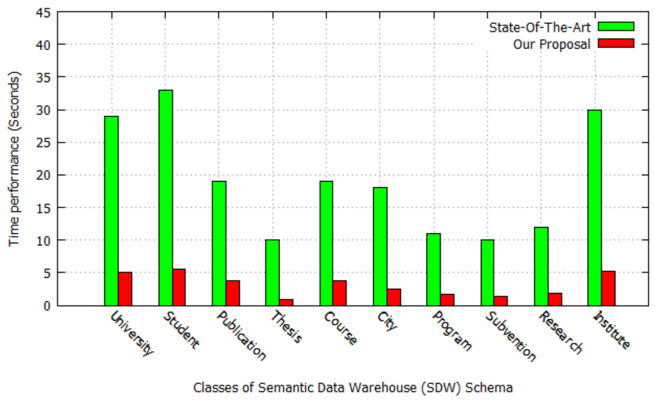


Figure 10. *Evaluation time(s) of Quads loaded in SDW per concept.*

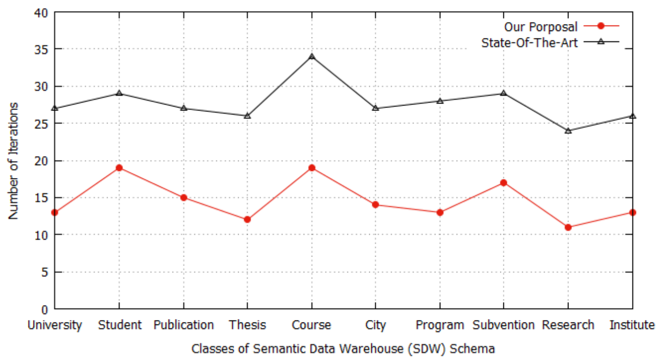


Figure 11. *Complexity of the ETL Algorithm.*

VI. Conclusion

In this paper, we highlight the importance of studying the interaction among phases of the life-cycle for designing advanced data warehouse applications. Serious research studies accompanied by database tools have shown the role of capturing interaction between features of some phases (we can cite the example of physical design in optimizing OLAP queries). We generalize this interaction between phases; where we considered two of them: ETL and physical design in the context of near time semantic data warehouse design. We have proposed a new approach for selecting materialized views that dynamically identifies relevant SPARQL queries to speed-up the execution of the $\mathcal{ETL}$ process that supports very fast streams and exploits the available memory. The use of views minimizes the graph comparison against the RDF quads set. At runtime, $\mathcal{ETL}$ transformations are analyzed to see whether they can be executed using the allocated memory and the selected materialized views. Experiments have shown the benefit of coupling materialized view selection and ETL processes.

Currently, we are working on the scalability of our approach by considering very large datasets and considering other non-functional requirements.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. B. Llavori, V. Nebot, M. J. A. Cabo, and A. Simitsis. Using semantic web technologies for exploratory OLAP: A survey. *IEEE Trans. Knowl. Data Eng.*, 27(2):571–588, 2015.

[2] S. M. F. Ali and R. Wrembel. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *VLDB J.*, 26(6):777–801, 2017.

[3] E. A. Azirani, F. Goasdoué, I. Manolescu, and A. Roatis. Efficient OLAP operations for RDF analytics. In *ICDE, 2015*, pages 71–76, 2015.

[4] L. Bellatreche. Optimization and tuning in data warehouses. In *Encyclopedia of Database Systems*, pages 1995–2003. 2009.

[5] N. Berkani, L. Bellatreche, and B. Benatallah. A value-added approach to design BI applications. In *DaWaK 2016*, pages 361–375, 2016.

[6] M. A. Bornea, A. Deligiannakis, Y. Kotidis, and V. Vassalos. Semi-streamed index join for near-real time execution of ETL transformations. In *ICDE 2011*, pages 159–170, 2011.

[7] A. Boukorca, L. Bellatreche, and A. Cuzzocrea. SLEMAS: an approach for selecting materialized views under query scheduling constraints. In *COMAD 2014.*, pages 66–73, 2014.

[8] A. Boukorca, L. Bellatreche, S. B. Senouci, and Z. Faget. Coupling materialized view selection to multi query optimization: Hyper graph approach. *IJDWM*, 11(2):62–84, 2015.

[9] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, 2003.

[10] R. Castillo and U. Leser. Selecting materialized views for RDF data. In *ICWE 2010*, pages 126–137, 2010.

[11] R. Castillo, C. Rothe, and U. Leser. *RDFMatView: Idexing RDF Data for SPARQL Queries*. Professoren des Inst. für Informatik, 2010.

[12] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *VLDB*, pages 146–155, 1997.

[13] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. Rdfviews: a storage tuning wizard for RDF applications. In *CIKM*, pages 1947–1948, 2010.

[14] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View selection in semantic web databases. *PVLDB*, 5(2):97–108, 2011.

[15] H. Gupta. *Selection and maintenance of views in a data warehouse.* PhD thesis, Stanford University USA., 1999.

[16] M. J-N. and J. Trujillo. An mda approach for the development of data warehouses. In *JISBD*, pages 208–208, 2009.

[17] A. Karakasidis, P. Vassiliadis, and E. Pitoura. Etl queues for active data warehousing. In L. Berti-Equille, C. Batini, and D. Srivastava, editors, *IQIS*, pages 28–39. ACM, 2005.

[18] S. Khouri, K. Semassel, and L. Bellatreche. Managing data warehouse traceability: A life-cycle driven approach. In *CAiSE*, pages 199–213, 2015.

[19] W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for sparql. In *(ICDE)*, pages 666–677. IEEE, 2012.

[20] B. Leopoldo and M. Mostafa. Ontological multidimensional data models and contextual data quality. *J. Data and Information Quality*, 9(3), 2018.

[21] B. L. Mbaiossoum, L. Bellatreche, and S. Jean. Materialized view selection considering the diversity of semantic web databases. In *ADBIS*, pages 163–176, 2014.

[22] J. Meehan, C. Aslantas, S. Zdonik, N. Tatbul, and J. Du. Data ingestion for the connected world. In *8th Biennial Conference on Innovative Data Systems Research, Chaminade*, 2017.

[23] J. Meehan, S. Zdonik, S. Tian, Y. Tian, N. Tatbul, A. Dziedzic, and A. J. Elmore. Integrating real-time and batch processing in a polystore. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2016.

[24] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.

[25] C. Ordonez, S. Maabout, D. S. Matusevich, and W. Cabrera. Extending ER models to capture database transformations to build data sets for data mining. *Data Knowl. Eng.*, 89:38–54, 2014.

[26] E. Schallehn, K. Sattler, and G. Saake. Advanced grouping and aggregation for data integration. In *CIKM*, pages 547–549, 2001.

[27] O. Shmueli and S. Tsur. Logical diagnosis of ldl programs. *New Generation Computing*, 9(3/4):277–304, 1991.

[28] A. Simitsis, P. Vassiliadis, and T.-K. Sellis. Optimizing etl processes in data warehouses. In *ICDE*, pages 564–575, 2005.

[29] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing etl workflows for fault-tolerance. In *ICDE*, pages 385–396, 2010.

[30] D. Skoutas and A. Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *Int. J. Semantic Web Inf. Syst.*, 3(4):1–24, 2007.

[31] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[32] J. Trujillo and S. Luján-Mora. A uml based approach for modeling etl processes in data warehouses. In *ER*, pages 307–320, 2003.

[33] Y. Tu and C. Guo. An intelligent etl workflow framework based on data partition. In *(ICIS)*, volume 3, pages 358–363. IEEE, 2010.

[34] P. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of etl workflows. In *DOLAP*, pages 49–56, 2007.

[35] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen. QB2OLAP: enabling OLAP on statistical linked open data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 1346–1349, 2016.

[36] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of etl scenarios. *Inf. Syst.*, 30(7):492–525, 2005.

[37] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for etl processes. In *DOLAP*, pages 14–21, 2002.

[38] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Modeling etl activities as graphs. In *DMDW*, pages 52–61, 2002.

[39] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the International Conference on Very Large Databases*, pages 136–145, 1997.

[40] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. J. Storm, C. Garcia-Arellano, and S. Fadden. DB2 design advisor: Integrated automatic physical database design. In *Proceedings of the International Conference on Very Large Databases*, pages 1087–1097, 2004.