$\mathcal{P}$at$\mathcal{M}$anQL: A language to manipulate patterns and data in hierarchical catalogs

Panagiotis Bouros, Theodore Dalamagas, Timos Sellis, Manolis Terrovitis

Knowledge and Database Systems Lab
School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 157 73, Athens, Hellas
{pbour,dalamag,timos,mter}@dblab.ece.ntua.gr

Abstract. Hierarchical structures and catalogs is a way to organize and enrich semantically the available information in the Web. From simple tree-like structures with syntactic constraints and type information, like DTDs and XML schemas, to hierarchies on a category/subcategory basis, like thematic hierarchies and RDF(s) models, such structures group data under certain properties. Paths in these structures are the knowledge artifacts to represent such groups. Considering paths in hierarchies as patterns which provide a conceptual clustering of data in groups sharing common properties, we present $\mathcal{P}$at$\mathcal{M}$anQL, a language to manipulate patterns and data in hierarchical catalogs.

1 Introduction

The Internet is today's greatest source of information. Huge volumes of data are posted and retrieved through the Web. Despite this vast exchange of information on the web, there is no consistent and strict organization of data [1]. Hierarchical structures and catalogs is a way to provide such kind of organization and enrich semantically the available information. From simple tree-like structures with syntactic constraints and type information (e.g. DTDs, XML schemas [6]) to hierarchies on a category/subcategory basis (e.g. thematic hierarchies of portal catalogs, RDF(s) models [6]), such structures group data under certain properties. *Paths* in these structures are the *knowledge artifacts* to represent such groups, and act as semantic guides to reach the data of each group either through a browsing task or through path expression query languages. For example, the path /motorcycles/bmw/cruizers/ represents several BMW models designed for long-time travelling and can be used in a path expression to search for motorcycles with price less than 16000: /motorcycles/bmw/cruizers[price<16000].

We consider paths in hierarchies as *patterns* which provide a conceptual clustering of raw data in groups sharing common properties. We do not focus on the extraction and creation of these patterns, but rather at their manipulation, combined with the manipulation of data. See for example Figure 1, where

two portal catalogs, Adorama and B&H[1], are presented. These catalogs provide photo equipment organized in a hierarchy on a category/subcategory basis. Searching for lenses in the first catalog needs the path /cameras & lenses/lenses, while in the second catalog needs the path /photo/35mm systems/lenses. Such paths can be seen as alternative pattern versions for the same group of data. On the other hand, there are cases where the owner of a catalog may need to provide integrated photo systems, with camera bodies from Adorama and the appropriate lenses from B&H. In such case, the two paths /cameras & lenses/35mm SLR and /photo/35mm systems/lenses form a kind of complex pattern representing different raw data, i.e. bodies and lenses, that should be grouped together.

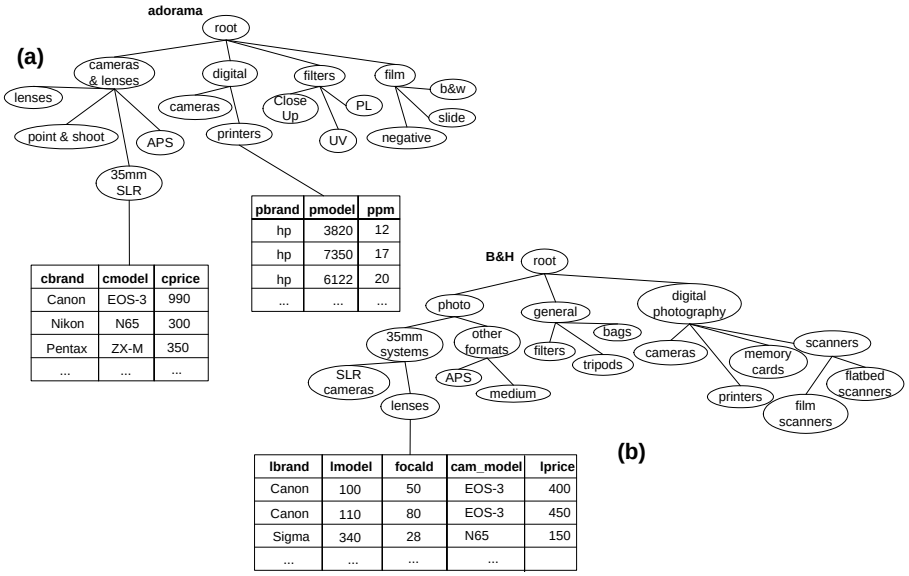


Fig. 1. Parts of Adorama and B&H catalogs

Our work addresses part of the requirements defined for pattern management in [11], emphasizing on the management of path-like patterns and data in hierarchical catalogs. Similar requirements have been also introduced in the inductive database framework [7], where patterns are integrated within the database environment as full-fledged objects. A number of specialized inductive query languages have been proposed [12, 5], but most of these concern descriptive rules.

Also, related to our work are algebras to manipulate tree-structured data, especially in XML documents. In [4] the authors present an algebra for XML data, but it is tuple-based and not tree-based. A navigational algebra is presented

[1] www.adorama.com, www.bhphotovideo.com

in [9], but again it treats individual nodes as manipulation units. In [8], TAX algebra is defined, but as a means for selecting and reconstructing bulk XML data.

In our work, we manipulate the descriptions of data as path-like patterns. We capture the notion of alternative path-like patterns and complex patterns to provide a framework to query raw data from many catalogs together with their patterns.

In this sense, the main contributions of our work are:

1. Models to represent hierarchical catalogs, emphasizing on the role of paths as knowledge artifacts in such structures.
2. The $\mathcal{P}$at$\mathcal{M}$anQL language with operators to manipulate path-like patterns (*select*, *project*, *cartesian product*, *union*, *intersection* and *difference*) together with raw data.
3. The development of a system implementing these operators.

The rest of the paper is organized as follows. Section 2 discusses representation issues for hierarchical catalogs. Section 3 defines a set of operators for $\mathcal{P}$at$\mathcal{M}$anQL and Section 4 presents several query examples based on these operators. Finally, Section 5 discusses some implementation details of our system and concludes this paper.

2 Representing hierarchical catalogs

We consider data in hierarchical catalogs to be organized in relations called *resource items*. Every resource item is characterized by a number of *attributes*. Resource items are mapped to the leaves of the hierarchy. Data present in hierarchies are instances (records) of resource items. For example, SLR cameras is a resource item with attributes brand, model and price, reached through the path /cameras & lenses/35mm SLR in Figure 2. Three products (cameras) are instances of this resource item. The hierarchy together with the resource items form the *catalog schema*, as the following definition states.

Definition 1. *A* catalog schema (CS) *is a tree* $CS = \{r, \mathcal{N}, \mathcal{R}\}$ *having a root* r*, a set* $\mathcal{N}$ *of nodes as non-leaf nodes and a set* $\mathcal{R}$ *of resource items as leaf-nodes.*

Figure 2 presents the catalog schema of Adorama's catalog for photo equipment. We denote root by $\bigotimes$, nodes in $\mathcal{N}$ by $\bigcirc$ and resource items in $\mathcal{R}$ by $\square$.

We can combine several catalog schemas with common resource items, creating *tree-structured relations (TSRs)*. Common resource items are items with a similar semantic interpretation[2] in the knowledge domain that catalogs refer to. Intuitively, a TSR represents the different ways of accessing a resource item in a set of catalog schemas and can be modeled using an AND/OR-like graph. This graph can be seen as a set of patterns, where each pattern is one of the different ways to access a resource item.

[2] In this work we consider schema matching and semantic mismatch issues to have been resolved using techniques suggested in the literature [10].

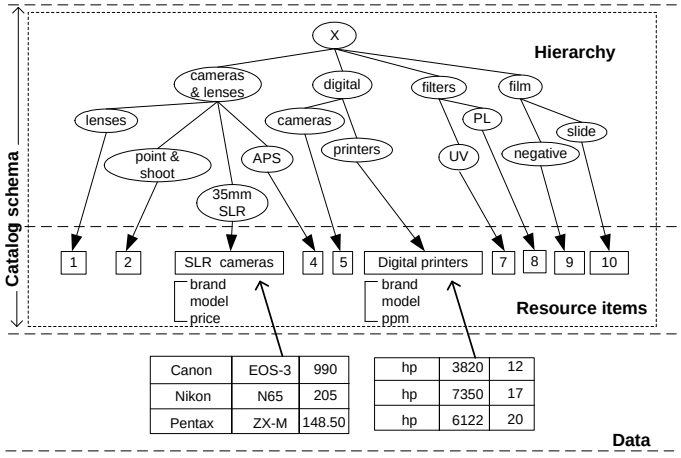


Fig. 2. A catalog schema for Adorama's catalog.

Definition 2. *Let* $\mathcal{S} = \{\{r, \mathcal{N}_1, \mathcal{R}_1\}, \{r, \mathcal{N}_2, \mathcal{R}_2\}, \ldots, \{r, \mathcal{N}_k, \mathcal{R}_k\}\}$ *be a set of* k *catalog schemas in a knowledge domain and* ri *a resource item in one or more* $\mathcal{R}_i$, $1 \leq i \leq k$. *A* TSR *is a graph* $\mathcal{G}$ *having a root* r, *a node representing the resource item* ri *and all paths from* r *to* ri. *Paths are grouped in* OR components. *Every* OR *component can be either an individual path or an* AND group *of paths.*

Figure 3(a) presents a simple TSR example with individual paths as OR components. Intuitively, an OR component captures a way to access a resource item. For example, /cameras & lenses/cameras/35mm SLR is one of the two alternative patterns for the resource item SLR cameras. Figure 3(b) shows a TSR with two OR components, one of which is an AND group denoted by the curved line crossing the involved paths /photo/35mm SLR/bodies and /photo/lenses. Intuitively, an AND group corresponds to complex resource items that can be constructed using items from one or many catalog schemas. To access such items, one should exploit all paths of the group. For example, the AND group in Figure 3(b) corresponds to SLR systems built from camera bodies and lenses, two resource items that are in different catalog schemas. The construction of AND groups is closely related to the *cartesian product* operator that we present in the next section.

We use *path index variables* $\$i, i > 0$, to identify paths in a TSR leading to the related resource item and OR *index variables* $\#j, j > 0$, to identify OR components in a TSR:

1. A path index variable $\$i$ refers to the ith path of every OR component of a TSR, starting from the left side. In Figure 3(b), for example, \$1 refers to /photo/35mm SLR/bodies and /photo/35mm systems, while \$2 refers only to /photo/lenses, since the right OR component has one individual path. With $\$\_$, we refer to any of the paths in every OR component in a TSR. Path

index variables are used in the *select* and *project* operator that we present in the next section.

2. An OR index variable $\#j$ refers to the jth OR component of a TSR, starting from the left side. In Figure 3(b), for example, $\#2$ refers to the right OR component of the TSR, which is the path /photo/35mm systems. OR index variables are used in the *project* operator.

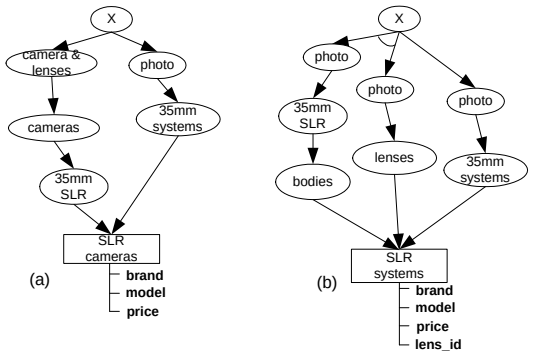


Fig. 3. TSR examples

We next introduce comparison operators for two given paths:

1. *Equality* (=). Two paths P_1, P_2 are *equal*, denoted by $P_1 = P_2$ if they contain exactly the same sequence of nodes.
2. *Strict subset* ($\subset$). A path P_1 is a *strict subset* of P_2, denoted by $P_1 \subset P_2$, if the sequence of nodes in P_1 appears identically in P_2. For example, /photo/35mm SLR $\subset$ /photo/35mm SLR/bodies.
3. *Loose subset* ($\sqsubset$). A path P_1 is a *loose subset* of P_2, denoted by $P_1 \sqsubset P_2$, if all nodes of P_1 exist in P_2 in the same order, no matter if there are other nodes between them. For example, /photo/bodies $\sqsubset$ /photo/35mm SLR/bodies.

3 The $\mathcal{P}$at$\mathcal{M}$anQL language

In this section, we present the $\mathcal{P}$at$\mathcal{M}$anQL language, defining the operators *select*, *project*, *cartesian product*, *union*, *intersection*, and *difference* to manipulate TSRs from catalog schemas. All operators are applied on TSRs and result in new TSRs.

1. **Select (σ)**. *Select* operates on a TSR to produce a new one, selecting instances of resource items and OR components whose paths satisfy the defined predicates. Its syntax follows:

$$\sigma_{<attribute\ condition><path\ condition>}(TSR)$$

Specifically, attribute condition is a list of predicates in the form of (attr *op* val) or (attr *op* attr), where attr is an attribute of the resource item related to TSR, val is a value from attr's domain and op $\in \{=, \neq, >\}$. Also, path condition is a list of predicates in the form of (path_var op path) or (path_var op path_var), where path_var is a path index variable in TSR, path is a path in TSR and op $\in \{=, \neq, \subset, \sqsubset\}$. Predicates can be used with AND,OR operators, denoted by ',' and '|', respectively. Figure 4(a) shows an example involving a select operator in the query 'construct a TSR to include all cameras from TSR SLR systems, other than Pentax, with price greater than 200, having "/photo/35mm systems" in their paths':

$$\sigma_{<brand\neq\text{"Pentax"},\ price>200><\text{"/photo/35mm systems"}\subset\$\_>}(SLR\ systems)$$

Results are presented in Figure 4(b). Notice that "/photo/35mm systems"$\subset$ $\$\_$ holds if there is a path p in an OR component, such that /photo/35mm systems $\subset p$. In this case, the whole OR component is retrieved.

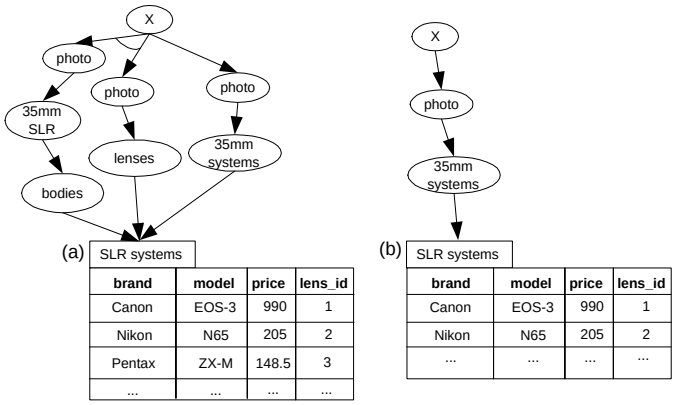


Fig. 4. Select operator

2. **Project (π).** *Project* operates on a TSR to produce a new one, keeping only some of the paths of each OR component or OR components on the whole, and some of the attributes of the related resource items. Its syntax follows:

$$\pi_{<attribute\ list><path\ or\ OR\ variable\ list>}(TSR)$$

Specifically, attribute list is a list of attributes of the resource item related to TSR, path variable list is a list of path index variables and OR variable list is a list of OR index variables. Figure 5(a) presents an example of a project operator in the query 'construct a new TSR from the TSR SLR systems

keeping only the rightmost component and the attributes model and lens_id of the resource item':

$$\pi_{<model,\ lens\_id><\#2>}(SLR\ systems)$$

Results are presented in Figure 5(b).

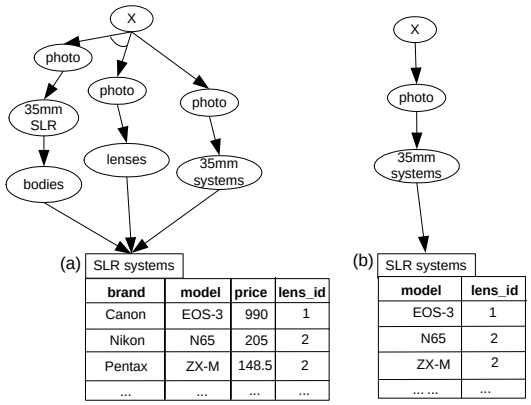


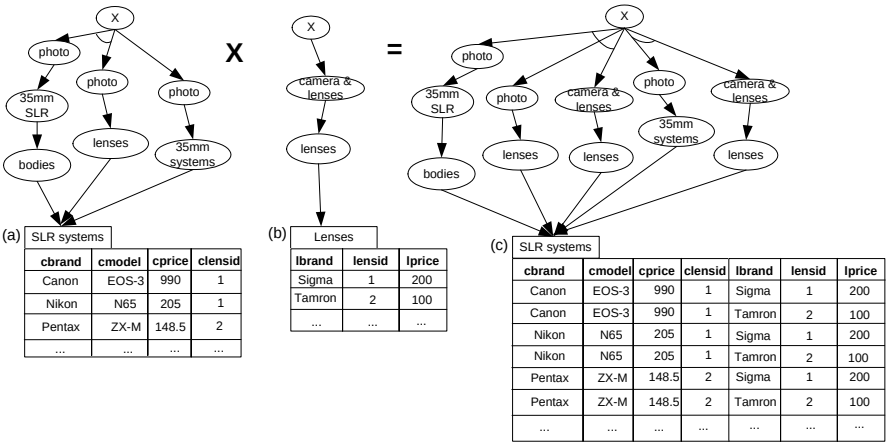
Fig. 5. Project operator

3. **Cartesian product** ($\times$). *Cartesian product* operates on two TSRs to produce a new TSR, combining (a) every OR component of the first TSR with every OR component of the second TSR, constructing an AND group of paths, and (b) every instance of the resource item in the first TSR with every instance of the resource item in the second TSR. Its syntax follows:

$$(TSR) \times (TSR)$$

Figure 6(c) shows the cartesian product of the two TSRs of Figure 6(a) and (b). The AND group of paths /photo/35mm SLR/bodies and /photo/lenses (which forms the left OR component of TSR SLR systems) is combined with the one and only OR component /camera & lenses/lenses of TSR Lenses, to construct a new AND group of three paths. Similarly, the right OR component of TSR SLR systems is combined with /camera & lenses/lenses of TSR Lenses, to construct a new AND group of two paths. The new TSR has a resource item with attributes from both TSRs and combinations of instances.

4. **Union** ($\bigcup$). *Union* operates on two TSRs with the same set of resource item attributes. It produces a new TSR with all the OR components of the two TSRs and the union of their instances. Its syntax follows:



(a) SLR systems

cbrand	cmodel	cprice	clensid
Canon	EOS-3	990	1
Nikon	N65	205	1
Pentax	ZX-M	148.5	2
...	...	...	...

(b) Lenses

lbrand	lensid	lprice
Sigma	1	200
Tamron	2	100
...	...	...

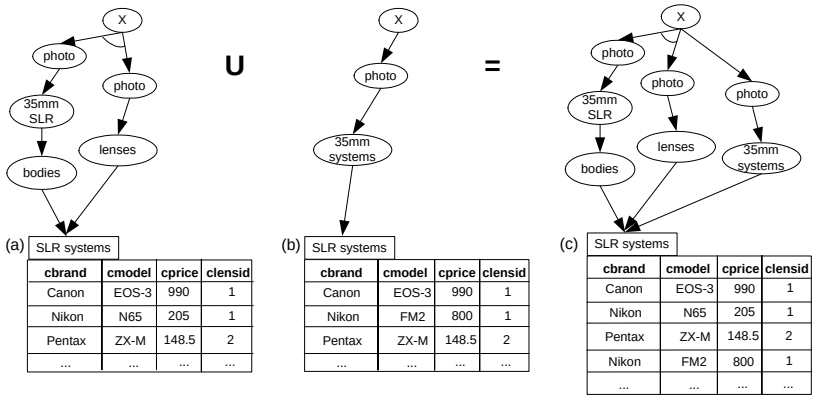
(c) SLR systems

cbrand	cmodel	cprice	clensid	lbrand	lensid	lprice
Canon	EOS-3	990	1	Sigma	1	200
Canon	EOS-3	990	1	Tamron	2	100
Nikon	N65	205	1	Sigma	1	200
Nikon	N65	205	1	Tamron	2	100
Pentax	ZX-M	148.5	2	Sigma	1	200
Pentax	ZX-M	148.5	2	Tamron	2	100
...	...	...	...	...	...	...

Fig. 6. Cartesian product operator

$$(TSR) \bigcup (TSR)$$

Figure 7(c) shows the union of the two TSRs of Figure 7(a) and (b). The TSR constructed has all OR components of both TSRs. Also, the result contains instances either from the first or the second TSR.



(a) SLR systems

cbrand	cmodel	cprice	clensid
Canon	EOS-3	990	1
Nikon	N65	205	1
Pentax	ZX-M	148.5	2
...	...	...	...

(b) SLR systems

cbrand	cmodel	cprice	clensid
Canon	EOS-3	990	1
Nikon	FM2	800	1
Pentax	ZX-M	148.5	2
...	...	...	...

(c) SLR systems

cbrand	cmodel	cprice	clensid
Canon	EOS-3	990	1
Nikon	N65	205	1
Pentax	ZX-M	148.5	2
Nikon	FM2	800	1
...	...	...	...

Fig. 7. Union operator

5. **Intersection** ($\bigcap$). Similarly to the union operator, *intersection* operates on two TSRs with the same set of resource item attributes. It produces a

new TSR with all the OR components of the two TSRs and their common instances. Its syntax follows:

$$(TSR) \bigcap (TSR)$$

6. **Difference (-)**. *Difference* operates on two TSRs with the same set of resource item attributes. It produces a new TSR with the OR components of the first TSR and the instances of the first which do not exist in the second. Its syntax follow:

$$(TSR) \text{ - } (TSR)$$

In the following section we present an example of manipulating TSRs from catalog schemas.

4 Using $\mathcal{P}$at$\mathcal{M}$anQL to manipulate catalog schemas

We next present a scenario to show how the operators presented in this paper can help the manipulation of paths as patterns, together with traditional data processing, in an environment of many catalogs related to photo equipment.

Figure 8 presents TSRs from catalog schemas related to photo equipment. Specifically, (b) and (c) are TSRs from the catalog schemas of Adorama and B&H catalogs presented in Figure 1, while (a) is a TSR of a new, imaginary catalog owned by X to serve the needs of our example. X sells integrated photo equipment, that is camera bodies and lenses as one package. Since new lenses are out in the market, X needs to find among the lenses provided by B&H, those that fit in Canon bodies provided by Adorama, and are not in her stock as integrated systems. The steps to construct such a query follow:

1. $q_1 = \pi_{<cbrand,cmodel,lmodel><>}(\sigma_{<cmodel=cam\_model,cbrand=\text{“Canon”}><>}((SLR\ cameras)\times(lenses)))$ results in a TSR presented in Figure 9(a), based on the cartesian product on TSRs SLR cameras and Lenses and keeping only the attributes cbrand, cmodel and lmodel of the resource items for Canon camera bodies and fitted lenses. Paths from both TSRs are included in an AND group. Query q_1 constructs a TSR for systems with Canon bodies from Adorama and lenses from B&H.
2. $q_2 = (q_1) - (\pi_{<cbrand,cmodel,lmodel><>}(SLR\ systems))$ results in a TSR presented in Figure 9(b), with one OR component, showing that there is an integrated photo system, including a body EOS-3 and lens 110 not offered by X. Query q_2 constructs a TSR for systems with Canon bodies from Adorama and lenses from B&H which are not in X's catalog.

Having as a result only the lenses without the appropriate camera bodies, requires a projection operation: $\pi_{<lmodel><\$2>}(q_2)$, which keeps only lmodel attribute and the path /photo/35mm systems/lenses (see Figure 9(c)).

X now needs to build a catalog with her own catalog plus all integrated photo systems having Canon camera bodies from Adorama's catalog and lenses from B&H catalog, to compare their prices. The steps to construct such a query follow:

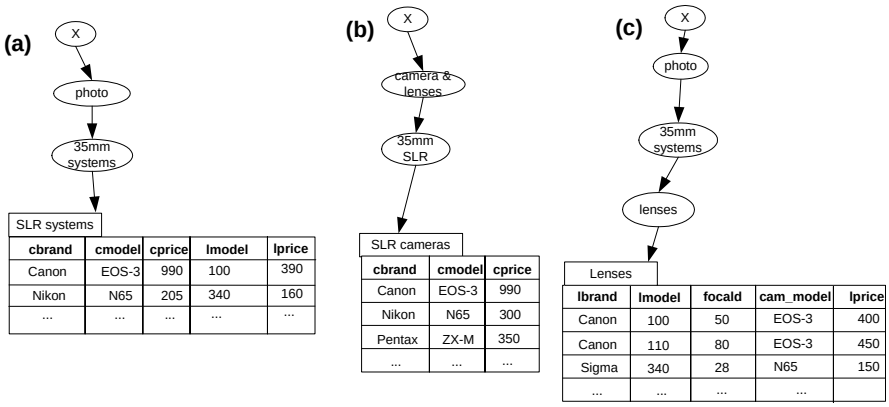


Fig. 8. Example TSRs

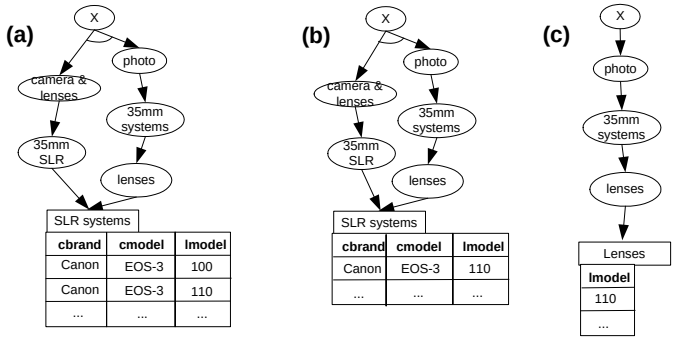


Fig. 9. Manipulating TSRs (I)

1. $q_3 = \pi_{<cbrand,cmodel,cprice,lmodel,lprice><>}(\sigma_{<cmodel=cam\_model,cbrand=\text{“Canon”}><>}(q))$, where $q = (SLR\ cameras) \times (Lenses)$, results in a TSR presented in Figure 10(a), based on the cartesian product on TSRs SLR cameras and Lenses and keeping only the attributes cbrand, cmodel, cprice, lmodel and lprice of the resource items for Canon camera bodies and fitted lenses. Paths from both TSRs are included in an AND group. Query q_3 constructs a TSR for systems with Canon bodies from Adorama and lenses from B&H.
2. $q_4 = (\pi_{<cbrand,cmodel,cprice,lmodel,lprice><>}(SLR\ systems)) \bigcup (q_3)$ results in a TSR presented in Figure 10(b), with two OR components, showing all integrated photo systems from all catalogs. Query q_4 constructs a TSR for systems with Canon bodies from Adorama and lenses from B&H, as well as systems from X.

Searching for paths that lead to lenses including nodes photo and lenses requires a selection operation: $q_5 = \sigma_{<><\text{“/photo/lenses”}\sqsubset\$\_>}(q_4)$). A final projection will produce a TSR only for lenses (see Figure 10(c)): $\pi_{<lmodel,lprice><\$2>}(q_5)$).

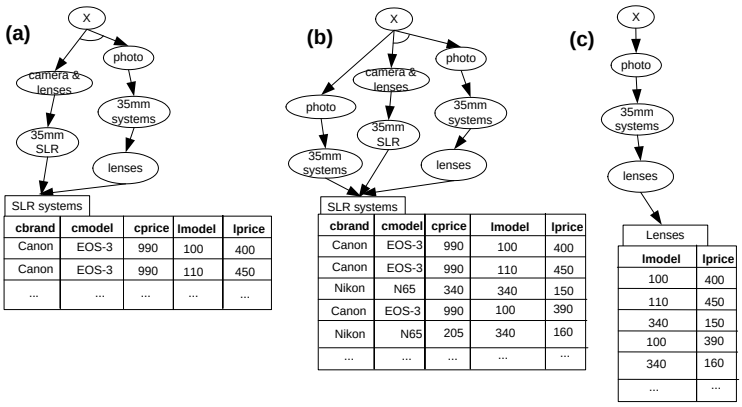


Fig. 10. Manipulating TSRs (II)

5 Discussion and Conclusions

In this work we presented a framework to manipulate path-like patterns and data in hierarchical catalogs. We modeled hierarchical catalogs using catalog schemas and we introduced tree-structured relations (TSRs) to represent the different ways of accessing a resource item in a set of catalog schemas. TSRs maintain alternative path-like pattern versions and complex patterns. Considering paths in these structures as knowledge artifacts to group raw data sharing common properties, we suggested $\mathcal{P}$at$\mathcal{M}$anQL, a set of operators for TSRs to manipulate paths together with the raw data: select, project, cartesian product, union, intersection and difference. Also, we showed examples of using such operators to manipulate hierarchical catalogs.

We have developed the prototype system $\mathcal{P}$at$\mathcal{M}$an [2] with a query execution engine that implements the suggested operators and a storage mechanism for TSRs (see Figure 11). TSRs can be stored (a) using plain XML files or (b) using the MySQL RDBMS system that $\mathcal{P}$at$\mathcal{M}$an exploits, with a relational schema that follows the all-edges-in-one-table example [3]. The systems retrieves TSRs using either the XML file manager (XFM) or the Database Manager (DM) and evaluates the query expression in main memory using the Query Execution Engine (QE).

We plan to extend the work presented in this paper along several directions. First, we will further explore the role of paths as knowledge artifacts in hierarchical catalogs, searching for useful comparison operators for paths. Also, we

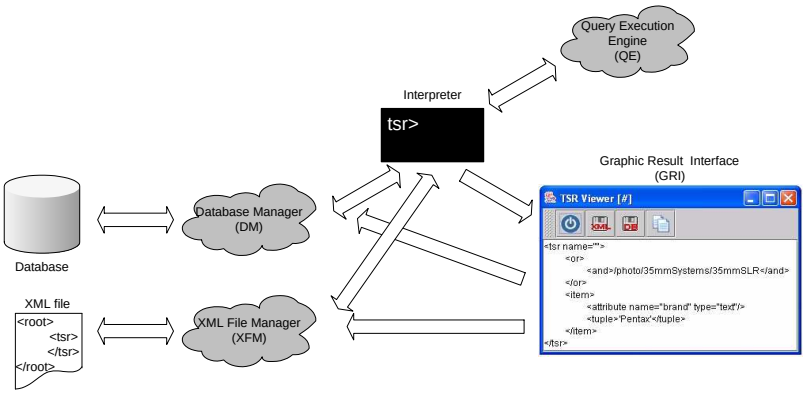


Fig. 11. System architecture

will introduce additional operators for path management. For example, in the previous section one can identify the need for a join operator based on the select and cartesian product operators.

References

1. S. Abiteboul, P. Buneman, and D. Suciu, *Data on the web. From relations to semistructured data and xml*, Morgan Kaufmann Publishers, 2000.
2. P. Bouros, *A query language for hierarchical structures*, KDBS Lab, Dept of Electrical and Computer Eng, NTU Athens, Oct 2003, Diploma Thesis (in Greek).
3. A. Chaudhri, A. Rashid, and R. Zicari, *XML data management: Native XML and XML-enabled database systems*, Addison Wesley, 2003.
4. V. Christophides, S. Cluet, and J. Simeon, *On wrapping query languages and efficient XML integration*, Proc. of the ACM SIGMOD Conf., 2000, pp. 141–152.
5. Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski, and Osmar Zaiane, *DMQL: A data mining query language for relational databases*, Proc. of the SIGMOD'96 DKMD Workshop, Mondreal, Canada, 1996.
6. http://www.w3c.org.
7. Tomasz Imielinski and Heikki Mannila, *A database perspective on knowledge discovery*, Commun. ACM **39** (1996), no. 11, 58–64.
8. H. V. Jagadish, Laks V. S. Lakshmanan, D. Srivastava, and K. Thompson, *TAX: A tree algebra for XML*, DBPL Conference, 2001, pp. 149–164.
9. B. Ludascher, Y. Papakonstantinou, and P. Velikhov, *Navigation-driven evaluation of virtual mediated views*, Lecture Notes in Computer Science **1777** (2000).
10. E. Rahm and P. A. Bernstein, *A survey of approaches to automatic schema matching*, VLDB Journal **10** (2001), no. 4, 334–350.
11. S. Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vassiliadis, M. Vazirgiannis, and E. Vrachnos, *Towards a logical model for patterns*, Proc. of ER'03 Conference, 2003, pp. 77–90.
12. R.Meo, G.Psaila, and S.Ceri, *An extension to SQL for mining association rules in SQL*, Data Mining and Knowledge Discovery **2** (1998), no. 2, 195–224.