Sihem Amer-Yahia
Zohra Bellahsène
Ela Hunt
Rainer Unland
Jeffrey Xu Yu (Eds.)

LNCS 4156

Database and XML Technologies

4th International XML Database Symposium, XSym 2006
Seoul, Korea, September 2006
Proceedings

Springer

Sihem Amer-Yahia Zohra Bellahsène
Ela Hunt Rainer Unland
Jeffrey Xu Yu (Eds.)

Database and XML Technologies

4th International XML Database Symposium, XSym 2006
Seoul, Korea, September 10-11, 2006
Proceedings

Springer

Volume Editors

Sihem Amer-Yahia
Yahoo! Research
New York, NY, 10011, USA
E-mail: sihem@yahoo-inc.com

Zohra Bellahsène
LIRMM UMR 5506 CNRS/Université Montpellier II
34392 Montpellier, France
E-mail: bella@lirmm.fr

Ela Hunt
Global Information Systems Group,
Institut für Informationssysteme, ETH-Zentrum
8092 Züich, Switzerland
E-mail: elahunt@inf.ethz.ch

Rainer Unland
University of Duisburg-Essen
Institute for Computer Science and Business Information Systems (ICB)
45117 Essen, Germany
E-mail: UnlandR@informatik.uni-essen.de

Jeffrey Xu Yu
The Chinese University of Hong Kong
Department of Systems Engineering and Engineering Management
Shatin, New Territories, Hong Kong, China
E-mail: yu@se.cuhk.edu.hk

Library of Congress Control Number: 2006931570

CR Subject Classification (1998): H.2, H.3, H.4, D.2, C.2.4

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN	0302-9743
ISBN-10	3-540-38877-X Springer Berlin Heidelberg New York
ISBN-13	978-3-540-38877-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11841920 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4156

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

The theme of the XML Database Symposium (XSym) is the convergence of database technology with XML technology. Since the first International XML Symposium in 2003, XSym has continued to provide a forum for academics, practitioners, users and vendors to discuss the use of and synergy between advanced XML technologies.

XSym 2006 received 32 full paper submissions. Each submitted paper underwent a rigorous review by independent referees. These proceedings represent a collection of eight excellent research papers. Their focus is on building XML repositories and covers the following topics: XML query processing, caching, indexing and navigation support, structural matching, temporal XML, and XML updates.

The organizers would like to express their gratitude to the XSym program committee for their efforts in providing very thorough evaluations of the submitted papers under significant time constraints. We also would like to thank Microsoft for their sponsorship and for the use of the Microsoft Conference Management Toolkit, and the local organizers, especially, Kyuseok Shim, for all they did to make XSym a pleasant and successful event.

July 2006

Sihem Amer-Yahia
Zohra Bellahsene
Jeffrey Xu Yu
Ela Hunt
Rainer Unland

Organization

General Chair

Zohra Bellahsene, LIRMM (France)

Local Chair

Kyuseok Shim, Seoul National University (Korea)

Program Committee Chairs

Sihem Amer-Yahia, Yahoo (USA)
Jeffrey Xu Yu, Chinese University of Hong Kong (China)

Proceedings

Rainer Unland, University of Duisburg-Essen (Germany)

Communications and Sponsorship

Ela Hunt, ETH Zurich (Switzerland)

Members of the International Program Committee

Ashraf Aboulnaga, University of Waterloo (Canada)
Bernd Amann, Université Paris 6 (France)
Denilson Barbosa, University of Calgary (Canada)
Omar Benjelloun, Stanford University (USA)
Veronique Benzaken, Université Paris-Sud (France)
Philip Bernstein, Microsoft Research (USA)
Philip Bohannon, Bell Laboratories - Lucent Technologies (USA)
Jihad Boulos, American University of Beirut (Lebanon)
Stephane Bressan, National University of Singapore (Singapore)
Yi Chen, Arizona State University (USA)
Alex Dekhtayar, University of Kentucky (USA)
Alin Deutsch, University of California at San Diego (USA)
Yanlei Diao, University of Massachusetts at Amherst (USA)
Irini Fundulaki, University of Edinburgh (UK)
Minos Garofalakis, Intel Research (USA)

Giorgio Ghelli, Università di Pisa (Italy)
Torsten Grust, Technical University of Munich (Germany)
Giovanna Guerrini, Università di Genova (Italy)
Ela Hunt, ETH Zurich (Switzerland)
Ihab Ilyas, University of Waterloo (Canada)
Zack Ives, University of Pennsylvania (USA)
Vanja Josifovski, Yahoo Research (USA)
Carl Christian Kanne, University of Mannheim (Germany)
Yaron Kanza, University of Toronto (Canada)
Raghav Kaushik, Microsoft Research (USA)
Laks Lakshmanan, University of British Columbia (Canada)
Dongwon Lee, Pennsylvania State University (USA)
Mong Li Lee, National University of Singapore (Singapore)
Qiong Luo, Hong Kong University of Science and Technology (China)
Murali Mani, Worcester Polytechnic Institute (USA)
Amelie Marian, Rutgers University (USA)
Peter McBrien, Imperial College - London (UK)
Tova Milo, Tel Aviv University (Israel)
Atsuyuki Morishima, University of Tsukuba (Japan)
Fatma Ozcan, IBM Almaden Research Center (USA)
Tamer Ozsu, University of Waterloo (Canada)
Tadeusz Pankowski, Poznan University of Technology (Poland)
Alkis Polyzotis, University of California at Santa Cruz (USA)
Philippe Pucheral, INRIA (France)
Prakash Ramanan, Wichita State University (USA)
Michael Rys, Microsoft (USA)
Monica Scannapieco, Università di Roma "La Sapienza" (Italy)
Jayavel Shanmugasundaram, Cornell University (USA)
Jerome Simeon, IBM Research (USA)
Divesh Srivastava, AT&T Research (USA)
Martin Theobald, Max-Planck-Institut für Informatik (Germany)
Vasilis Vassalos, Athens University of Economics and Business (Greece)
Stratis Viglas, University of Edinburgh (UK)
Yuqing Melanie Wu, Indiana University (USA)
Jun Yang, Duke University (USA)

Lecture Notes in Computer Science

For information about Vols. 1–4060

please contact your bookseller or Springer

Vol. 4185: R. Mizoguchi, Z. Shi, F. Giunchiglia (Eds.), The Semantic Web – ASWC 2006. XX, 778 pages. 2006.

Vol. 4180: M. Kohlhase, OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. XIX, 428 pages. 2006. (Sublibrary LNAI).

Vol. 4176: S.K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, B. Preneel (Eds.), Information Security. XIV, 548 pages. 2006.

Vol. 4168: Y. Azar, T. Erlebach (Eds.), Algorithms – ESA 2006. XVIII, 843 pages. 2006.

Vol. 4163: H. Bersini, J. Carneiro (Eds.), Artificial Immune Systems. XII, 460 pages. 2006.

Vol. 4162: R. Královič, P. Urzyczyn (Eds.), Mathematical Foundations of Computer Science 2006. XV, 814 pages. 2006.

Vol. 4159: J. Ma, H. Jin, L.T. Yang, J.J.-P. Tsai (Eds.), Ubiquitous Intelligence and Computing. XXII, 1190 pages. 2006.

Vol. 4158: L.T. Yang, H. Jin, J. Ma, T. Ungerer (Eds.), Autonomic and Trusted Computing. XIV, 613 pages. 2006.

Vol. 4156: S. Amer-Yahia, Z. Bellahsène, E. Hunt, R. Unland, J.X. Yu (Eds.), Database and XML Technologies. IX, 123 pages. 2006.

Vol. 4155: O. Stock, M. Schaerf (Eds.), Reasoning, Action and Interaction in AI Theories and Systems. XVIII, 343 pages. 2006. (Sublibrary LNAI).

Vol. 4153: N. Zheng, X. Jiang, X. Lan (Eds.), Advances in Machine Vision, Image Processing, and Pattern Analysis. XIII, 506 pages. 2006.

Vol. 4152: Y. Manolopoulos, J. Pokorný, T. Sellis (Eds.), Advances in Databases and Information Systems. XV, 448 pages. 2006.

Vol. 4151: A. Iglesias, N. Takayama (Eds.), Mathematical Software - ICMS 2006. XVII, 452 pages. 2006.

Vol. 4150: M. Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle (Eds.), Ant Colony Optimization and Swarm Intelligence. XVI, 526 pages. 2006.

Vol. 4146: J.C. Rajapakse, L. Wong, R. Acharya (Eds.), Pattern Recognition in Bioinformatics. XIV, 186 pages. 2006. (Sublibrary LNBI).

Vol. 4144: T. Ball, R.B. Jones (Eds.), Computer Aided Verification. XV, 564 pages. 2006.

Vol. 4139: T. Salakoski, F. Ginter, S. Pyysalo, T. Pahikkala, Advances in Natural Language Processing. XVI, 771 pages. 2006. (Sublibrary LNAI).

Vol. 4138: X. Cheng, W. Li, T. Znati (Eds.), Wireless Algorithms, Systems, and Applications. XVI, 709 pages. 2006.

Vol. 4137: C. Baier, H. Hermanns (Eds.), CONCUR 2006 – Concurrency Theory. XIII, 525 pages. 2006.

Vol. 4136: R.A. Schmidt (Ed.), Relations and Kleene Algebra in Computer Science. XI, 433 pages. 2006.

Vol. 4135: C.S. Calude, M.J. Dinneen, G. Păun, G. Rozenberg, S. Stepney (Eds.), Unconventional Computation. X, 267 pages. 2006.

Vol. 4134: K. Yi (Ed.), Static Analysis. XIII, 443 pages. 2006.

Vol. 4133: J. Gratch, M. Young, R. Aylett, D. Ballin, P. Olivier (Eds.), Intelligent Virtual Agents. XIV, 472 pages. 2006. (Sublibrary LNAI).

Vol. 4130: U. Furbach, N. Shankar (Eds.), Automated Reasoning. XV, 680 pages. 2006. (Sublibrary LNAI).

Vol. 4129: D. McGookin, S. Brewster (Eds.), Haptic and Audio Interaction Design. XII, 167 pages. 2006.

Vol. 4128: W.E. Nagel, W.V. Walter, W. Lehner (Eds.), Euro-Par 2006 Parallel Processing. XXXIII, 1221 pages. 2006.

Vol. 4127: E. Damiani, P. Liu (Eds.), Data and Applications Security XX. X, 319 pages. 2006.

Vol. 4126: P. Barahona, F. Bry, E. Franconi, N. Henze, U. Sattler, Reasoning Web. X, 269 pages. 2006.

Vol. 4124: H. de Meer, J.P. G. Sterbenz (Eds.), Self-Organizing Systems. XIV, 261 pages. 2006.

Vol. 4121: A. Biere, C.P. Gomes (Eds.), Theory and Applications of Satisfiability Testing - SAT 2006. XII, 438 pages. 2006.

Vol. 4119: C. Dony, J.L. Knudsen, A. Romanovsky, A. Tripathi (Eds.), Advanced Topics in Exception Handling Components. X, 302 pages. 2006.

Vol. 4117: C. Dwork (Ed.), Advances in Cryptology - CRYPTO 2006. XIII, 621 pages. 2006.

Vol. 4116: R. De Prisco, M. Yung (Eds.), Security and Cryptography for Networks. XI, 366 pages. 2006.

Vol. 4115: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Computational Intelligence and Bioinformatics, Part III. XXI, 803 pages. 2006. (Sublibrary LNBI).

Vol. 4114: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Computational Intelligence, Part II. XXVII, 1337 pages. 2006. (Sublibrary LNAI).

Vol. 4113: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Intelligent Computing, Part I. XXVII, 1331 pages. 2006.

Vol. 4112: D.Z. Chen, D. T. Lee (Eds.), Computing and Combinatorics. XIV, 528 pages. 2006.

Vol. 4111: F.S. de Boer, M.M. Bonsangue, S. Graf, W.-P. de Roever (Eds.), Formal Methods for Components and Objects. VIII, 447 pages. 2006.

Vol. 4110: J. Díaz, K. Jansen, J.D.P. Rolim, U. Zwick (Eds.), Approximation, Randomization, and Combinatorial Optimization. XII, 522 pages. 2006.

Vol. 4109: D.-Y. Yeung, J.T. Kwok, A. Fred, F. Roli, D. de Ridder (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. XXI, 939 pages. 2006.

Vol. 4108: J.M. Borwein, W.M. Farmer (Eds.), Mathematical Knowledge Management. VIII, 295 pages. 2006. (Sublibrary LNAI).

Vol. 4106: T.R. Roth-Berghofer, M.H. Göker, H. A. Güvenir (Eds.), Advances in Case-Based Reasoning. XIV, 566 pages. 2006. (Sublibrary LNAI).

Vol. 4104: T. Kunz, S.S. Ravi (Eds.), Ad-Hoc, Mobile, and Wireless Networks. XII, 474 pages. 2006.

Vol. 4099: Q. Yang, G. Webb (Eds.), PRICAI 2006: Trends in Artificial Intelligence. XXVIII, 1263 pages. 2006. (Sublibrary LNAI).

Vol. 4098: F. Pfenning (Ed.), Term Rewriting and Applications. XIII, 415 pages. 2006.

Vol. 4097: X. Zhou, O. Sokolsky, L. Yan, E.-S. Jung, Z. Shao, Y. Mu, D.C. Lee, D. Kim, Y.-S. Jeong, C.-Z. Xu (Eds.), Emerging Directions in Embedded and Ubiquitous Computing. XXVII, 1034 pages. 2006.

Vol. 4096: E. Sha, S.-K. Han, C.-Z. Xu, M.H. Kim, L.T. Yang, B. Xiao (Eds.), Embedded and Ubiquitous Computing. XXIV, 1170 pages. 2006.

Vol. 4095: S. Nolfi, G. Baldassare, R. Calabretta, D. Marocco, D. Parisi, J.C. T. Hallam, O. Miglino, J.-A. Meyer (Eds.), From Animals
to Animats 9. XV, 869 pages. 2006. (Sublibrary LNAI).

Vol. 4094: O. H. Ibarra, H.-C. Yen (Eds.), Implementation and Application of Automata. XIII, 291 pages. 2006.

Vol. 4093: X. Li, O.R. Zaïane, Z. Li (Eds.), Advanced Data Mining and Applications. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 4092: J. Lang, F. Lin, J. Wang (Eds.), Knowledge Science, Engineering and Management. XV, 664 pages. 2006. (Sublibrary LNAI).

Vol. 4091: G.-Z. Yang, T. Jiang, D. Shen, L. Gu, J. Yang (Eds.), Medical Imaging and Augmented Reality. XIII, 399 pages. 2006.

Vol. 4090: S. Spaccapietra, K. Aberer, P. Cudré-Mauroux (Eds.), Journal on Data Semantics VI. XI, 211 pages. 2006.

Vol. 4089: W. Löwe, M. Südholt (Eds.), Software Composition. X, 339 pages. 2006.

Vol. 4088: Z.-Z. Shi, R. Sadananda (Eds.), Agent Computing and Multi-Agent Systems. XVII, 827 pages. 2006. (Sublibrary LNAI).

Vol. 4087: F. Schwenker, S. Marinai (Eds.), Artificial Neural Networks in Pattern Recognition. IX, 299 pages. 2006. (Sublibrary LNAI).

Vol. 4085: J. Misra, T. Nipkow, E. Sekerinski (Eds.), FM 2006: Formal Methods. XV, 620 pages. 2006.

Vol. 4084: M.A. Wimmer, H.J. Scholl, Å. Grönlund, K.V. Andersen (Eds.), Electronic Government. XV, 353 pages. 2006.

Vol. 4083: S. Fischer-Hübner, S. Furnell, C. Lambrinoudakis (Eds.), Trust and Privacy in Digital Business. XIII, 243 pages. 2006.

Vol. 4082: K. Bauknecht, B. Pröll, H. Werthner (Eds.), E-Commerce and Web Technologies. XIII, 243 pages. 2006.

Vol. 4081: A. M. Tjoa, J. Trujillo (Eds.), Data Warehousing and Knowledge Discovery. XVII, 578 pages. 2006.

Vol. 4080: S. Bressan, J. Küng, R. Wagner (Eds.), Database and Expert Systems Applications. XXI, 959 pages. 2006.

Vol. 4079: S. Etalle, M. Truszczyński (Eds.), Logic Programming. XIV, 474 pages. 2006.

Vol. 4077: M.-S. Kim, K. Shimada (Eds.), Geometric Modeling and Processing - GMP 2006. XVI, 696 pages. 2006.

Vol. 4076: F. Hess, S. Pauli, M. Pohst (Eds.), Algorithmic Number Theory. X, 599 pages. 2006.

Vol. 4075: U. Leser, F. Naumann, B. Eckman (Eds.), Data Integration in the Life Sciences. XI, 298 pages. 2006. (Sublibrary LNBI).

Vol. 4074: M. Burmester, A. Yasinsac (Eds.), Secure Mobile Ad-hoc Networks and Sensors. X, 193 pages. 2006.

Vol. 4073: A. Butz, B. Fisher, A. Krüger, P. Olivier (Eds.), Smart Graphics. XI, 263 pages. 2006.

Vol. 4072: M. Harders, G. Székely (Eds.), Biomedical Simulation. XI, 216 pages. 2006.

Vol. 4071: H. Sundaram, M. Naphade, J.R. Smith, Y. Rui (Eds.), Image and Video Retrieval. XII, 547 pages. 2006.

Vol. 4070: C. Priami, X. Hu, Y. Pan, T.Y. Lin (Eds.), Transactions on Computational Systems Biology V. IX, 129 pages. 2006. (Sublibrary LNBI).

Vol. 4069: F.J. Perales, R.B. Fisher (Eds.), Articulated Motion and Deformable Objects. XV, 526 pages. 2006.

Vol. 4068: H. Schärfe, P. Hitzler, P. Øhrstrøm (Eds.), Conceptual Structures: Inspiration and Application. XI, 455 pages. 2006. (Sublibrary LNAI).

Vol. 4067: D. Thomas (Ed.), ECOOP 2006 – Object-Oriented Programming. XIV, 527 pages. 2006.

Vol. 4066: A. Rensink, J. Warmer (Eds.), Model Driven Architecture – Foundations and Applications. XII, 392 pages. 2006.

Vol. 4065: P. Perner (Ed.), Advances in Data Mining. XI, 592 pages. 2006. (Sublibrary LNAI).

Vol. 4064: R. Büschkes, P. Laskov (Eds.), Detection of Intrusions and Malware & Vulnerability Assessment. X, 195 pages. 2006.

Vol. 4063: I. Gorton, G.T. Heineman, I. Crnkovic, H.W. Schmidt, J.A. Stafford, C.A. Szyperski, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 394 pages. 2006.

Vol. 4062: G. Wang, J.F. Peters, A. Skowron, Y. Yao (Eds.), Rough Sets and Knowledge Technology. XX, 810 pages. 2006. (Sublibrary LNAI).

Vol. 4061: K. Miesenberger, J. Klaus, W. Zagler, A.I. Karshmer (Eds.), Computers Helping People with Special Needs. XXIX, 1356 pages. 2006.

Table of Contents

Query Evaluation and Temporal XML

XPath and Twigs

XML Updates

Kappa-Join: Efficient Execution of Existential Quantification in XML Query Languages

Matthias Brantner[1,*], Sven Helmer[2], Carl-Christian Kanne[1], and Guido Moerkotte[1]

[1] University of Mannheim, Mannheim, Germany
{brantner, kanne, moerkotte}@informatik.uni-mannheim.de
[2] Birkbeck College, University of London, London, United Kingdom
sven@dcs.bbk.ac.uk

Abstract. XML query languages feature powerful primitives for formulating queries, involving comparison expressions which are existentially quantified. If such comparisons involve several scopes, they are correlated and, thus, become difficult to evaluate efficiently.

In this paper, we develop a new ternary operator, called Kappa-Join, for efficiently evaluating queries with existential quantification. In XML queries, a correlation predicate can occur conjunctively and disjunctively. Our decorrelation approach not only improves performance in the conjunctive case, but also allows decorrelation of the disjunctive case. The latter is not possible with any known technique. In an experimental evaluation, we compare the query execution times of the Kappa-Join with existing XPath evaluation techniques to demonstrate the effectiveness of our new operator.

1 Introduction

Almost every XML query language features a construct that allows to express an existentially quantified comparison of two node-set valued subexpressions in a concise manner. Unfortunately, current XML query processors lack efficiency and scalability when facing such constructs [5,20]. The corresponding semantics resembles that of nested and correlated subqueries, which are notoriously difficult to evaluate efficiently. To illustrate this point, let us consider the following query: For hiring a teaching assistant, we search the database for a student who took an exam that was graded better than 'B'.

```
for    $s                in  //student
let    $best             :=  //exam[grade < 'B']/@id
let    $exams            :=  $s/examination/@id
where  $exams = $best
return $s/name
```

Q1

Here, both sides of the comparison in the `where`-clause are set-valued because there are many good students and students take more than one exam. The existential semantics of the XQuery general comparison operator requires that all students are returned which have at least one exam also contained in the set `$best`.

A naïve evaluation technique evaluates the steps in order of appearance. In Q1, this means to reevaluate the value of `$best` and `$exams` for every iteration of the `for`

* This work was supported by the Deutsche Forschungsgemeinschaft under grant MO 507/10-1.

S. Amer-Yahia et al. (Eds.): XSym 2006, LNCS 4156, pp. 1–15, 2006.

loop, and then check for an item that occurs in both sets. This is a wasteful strategy: A closer look at Q1 reveals that, in contrast to `$exams`, the value of `$best` does not depend on the value of `$s`, making the reevaluation of `$best` unnecessary. A common strategy in such cases is to move the evaluation of `$best` out of the `for` loop, and to materialize and reuse the result. However, this only alleviates the problem of repeated evaluation of the expression to which `$best` is bound. To answer the query, it is still necessary to evaluate the `where`-predicate, which is a correlated nested query due to the existential semantics of the '=' operator and the fact that it refers to variables from two scopes, the independent `$best` and the dependent `$exams`.

In this paper, we are concerned with an efficient evaluation of existentially quantified *correlation predicates* such as the `where`-clause of Q1. While this area has received some attention in the past [5,20], we show that, even in simple cases like our Query Q1, there is still unexploited optimization potential for typical query patterns in XML query languages. We propose the novel Kappa-Join operator that fits naturally into execution plans for quantified correlation predicates, is easy to implement and yet features a decorrelated evaluation algorithm.

Q1 is "simple" because the correlation predicate occurs "alone". What if the correlation predicates become more complex? Assume that we consider *either* good *or* senior students to be eligible for assistantship, as in the following query:

If the two clauses were combined with **and**, we could use techniques to decorrelate queries with correlation predicates that occur conjunctively. If the clauses were not correlation predicates, we could use techniques to improve performance for disjunctive predicates (e.g. Bypass operators [8]). However, there is no published technique to decorrelate *disjunctively* occurring correlation predicates.

```
for   $s      in   //student
let   $best   :=   //exam[grade < 'B']/@id
let   $exams  :=   $s/examination/@id                    Q2
where $exams = $best or $s/semester>5
return $s/name
```

Hence, we also present a Bypass variant of the Kappa-Join. This allows a decorrelated evaluation of disjunctively occurring correlation predicates, which has not been accomplished for any query language so far.

The main contributions of this paper are as follows:

- We introduce the novel ternary Kappa-Join operator that, while simple to implement, can efficiently evaluate complex correlated queries where the correlation predicate occurs conjunctively.
- We introduce a Bypass variant of the Kappa-Join that allows us to extend our technique to queries where the correlation predicate occurs in a disjunction.
- We provide experimental results, demonstrating the superiority of both the Kappa-Join and the Bypass Kappa-Join compared to other evaluation techniques.

The remainder of this paper is organized as follows. In the next section, we discuss basic concepts, such as dependency and correlation in XPath. In Sec. 3, we discuss the drawbacks of existing decorrelation approaches for XML query languages. Further, we introduce our novel Kappa-Join operator to efficiently evaluate queries with conjunctive

correlation. Sec. 4 investigates the case of disjunctive correlation and presents the novel Bypass Kappa-Join. In Sec. 5 we experimentally confirm the efficiency of our approach. The last section concludes the paper.

2 XPath Query Processing

The problems discussed in the introduction affect most existing XML query languages. However, all of the involved issues occur even for the simple XPath language in its first version because it features nested predicates and existential semantics. In the following, we limit our discussion to XPath 1.0, because peripheral topics such as typing, query normalization, and translation into an algebraic representation can be presented in a simpler fashion for XPath than for the more powerful XQuery language. However, all of the techniques presented in this paper also apply for full-blown XQuery or similar languages, as long as they are evaluated algebraically (e.g. [23]). In fact, both queries from the introduction can be expressed in XPath syntax, as we will see below.

2.1 Normalization

The techniques presented in this paper are mainly developed to optimize comparison expressions with one dependent and one independent operand. To correctly identify such expressions, the first step of our normalization is to rewrite a predicate such that it consists of literals and the logic operators $\wedge$ and $\vee$. After normalization, each literal l consists either of a comparison or a negation thereof, i.e. l is of the form $\mathrm{e}_1 \theta \mathrm{e}_2$ or $\mathrm{not}(\mathrm{e}_1 \theta \mathrm{e}_2)$, where $\theta \in \{=, <, \leq, >, \geq, \neq\}$.

One example for "hidden" comparisons are location paths or other node-set valued expressions when used as Boolean expressions. In such cases, we make the node-set valued expression the argument of an auxiliary *exists* function and compare its result to true, which yields a regular binary comparison expression.

Further, to provide efficient evaluation for disjunctively occurring comparison expressions, the second step of our normalization separates literals occurring in a conjunction from those that occur disjunctively. To this end, we employ an operation for collecting all literals that occur conjunctively: A literal l occurs *conjunctively* in a predicate p_k if $\mathrm{p}_\mathrm{k}[\mathrm{false}]$ can be simplified to false. That is, if we replace all occurrences of l inside p_k by false, the repeated (syntactical) application of the Boolean simplification rules to eliminate Boolean constants simplifies $\mathrm{p}_\mathrm{k}[\mathrm{false}]$ to false.

2.2 Context Dependency and Correlation

In this paper, we are concerned with efficient evaluation of existentially quantified comparison expressions that are *correlated*. In general, correlation occurs when a variable of a nested scope is compared with a variable from an enclosing scope. XPath does not have variables that can be declared by the user, but we can define correlation in terms of XPath *contexts*, as follows.

Every XPath expression is evaluated with respect to a given context, which is a sequence of *context items*. For our discussion, it is sufficient to use a definition of context

item that is slightly simpler than the original XPath context item. A context item is a tuple with three components: the *context node*, the *context position*, and the *context size*. In XPath, there is one global context, which must be specified as parameter of the evaluation process. The value of some constructs depends on *local contexts* that are generated by other subexpressions. The constructs that refer to the local context are location steps, relative location paths, and calls to `position()` and `last()`. We call these expressions *dependent* expressions. Expressions whose value is independent of the local context are called *independent* expressions.

If we apply this terminology to Queries Q1 and Q2 from the introduction, given in XPath syntax, we have

$$\underbrace{//\text{student}}_{\text{independent}}\underbrace{[\text{examination/@id}}_{\text{dependent}} = \underbrace{//\text{exam}[\text{grade} <' \text{B}']/@\text{id}]/}_{\text{independent}} \underbrace{\text{name}}_{\text{dependent}} \qquad \textbf{Q1}$$

$$//\text{student}\underbrace{[\text{examination/@id}}_{\text{dependent}} = \underbrace{//\text{exam}[\text{grade} <' \text{B}']/@\text{id}}_{\text{independent}} \text{or} \underbrace{\text{semester}}_{\text{dependent}} > \underbrace{5}_{\text{indep.}}]/\text{name} \qquad \textbf{Q2}$$

We now can define the term correlation for XPath as used in the remainder of this paper: A comparison expression with two node-set valued operands one being dependent and one being independent is called *correlation predicate*, because it compares a local context and an enclosing context. All example queries presented in the introduction and above contain correlation predicates. A correlation predicate can occur both conjunctively and disjunctively. We call the former case *conjunctive correlation* and the latter *disjunctive correlation*. In Q1 there is only one comparison expression which is a special case of conjunctive correlation, i.e. one with only a single literal. Q2 is an example with disjunctive correlation. The second comparison expression is not a correlation predicate, because the operand 5 is not node-set valued.

3 Kappa-Join for Conjunctive Correlation

The key to an efficient evaluation of correlated queries is to avoid redundant computation, e.g. the evaluation of the inner independent expression. Such techniques are called decorrelation techniques and have been studied extensively in the context of relational and object-oriented systems [9,11,12,17,18,26]. Similar techniques have been proposed for the evaluation of XQuery and XPath [5,20]. One of them is an approach that applies decorrelation to existentially quantified comparison expressions [5]. However, this approach is suboptimal because unnecessary duplicates are generated and must be removed at the end of the evaluation.

The optimizations developed in this paper are presented in the form of algebraic operators. Hence, we need an algebra capable of evaluating XPath. We have chosen NAL as a perfect fit, since a translation from XPath to NAL is also available [4]. However, our approach is not limited to NAL and the translation of XPath into it. For example, our techniques are also applicable to other algebraic evaluation strategies such as [25].

At the beginning of this section, we describe our assumptions about algebraic translation and evaluation in more detail. For a more elaborate treatment of these topics, please refer to [4,5]. We then recapitulate the decorrelation approach from [5] and discuss its

drawbacks. Afterwards, we introduce the novel Kappa-Join operator that features an efficient decorrelation algorithm avoiding these drawbacks.

3.1 Algebra and Translation

The universe of the NAL algebra for XPath 1.0 is the union of the domains of the atomic XPath 1.0 types (`string, number, boolean`) and the set of ordered sequences of tuples which represent XPath contexts. Each tuple represents one context node, position and size. Special attribute names are used to hold the context node (cn), the context position (cp) and the context size (cs).

The NAL algebra features well-known operators [4,20]. All the sequence-valued operators in the logical algebra have a corresponding implementation as an *iterator* [14] in the physical algebra. In the following, we primarily need the *Selection* σ, the *Projection* Π, the *Semi-Join* $\ltimes$, and the *D-Join* $\overset{\rightarrow}{\bowtie}$. All operators are described when they are needed for the first time. Additionally, they are formally defined in our technical report [21].

To convert XPath queries into algebraic expressions, we use the translation introduced in [4]. We briefly recapitulate the relevant part of the translation process by elaborating on the translation result for Q1 (see Fig. 1). However, we omit the translation of subexpressions that are orthogonal to our discussion and denote them by $\mathcal{T}[e]$, where e stands for any XPath expression. For instance, we denote the translation of the location path `//student` by $\mathcal{T}[$//student$]$. Its result is the sequence of context nodes produced by the location path.

For Q1 (see Fig. 1) the algebra expression provides a Selection (σ) for the only literal. In the subscript (denoted by a dashed line) of this Selection, there is an Aggregation operator ($\mathfrak{A}$) that aggregates the input sequence into a singleton sequence with a single attribute by applying the aggregation function $exists$. It returns true if there exists at least one tuple in the input sequence. This input sequence stems from a Semi-Join ($\ltimes$), whose input sequences in turn stem from the two operands of the comparison expression, i.e. the two (translated) location path expressions. For a comparison between two node-sets, as in the particular case of Q1, we have an existential semantics. In the equality case, this fact can be leveraged by using a Semi-Join.

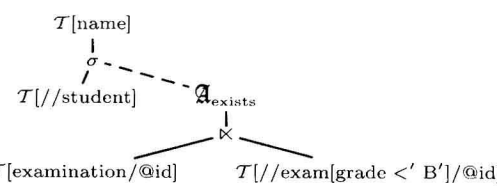

Fig. 1. Translation sketch for Q1

Because of the repeated evaluation of e_3 the worst-case complexity is $O(|e_1| \times |e_2| \times |e_3|)$ where $|e|$ denotes the cardinality of an expression e and, in this case, $e_1 =$ //student, $e_2 =$ examination/@id, and $e_3 =$ //exam . . ..

3.2 Existing Decorrelation Techniques

We now recapitulate the decorrelation approach introduced in [5] and discuss its drawbacks. Again, we take Query Q1 to illustrate this (see Fig. 1). The fundamental idea of decorrelation is to avoid unnecessary evaluations of an inner independent expression. In [5] this is achieved by pulling up the Semi-Join (see Fig. 1) into the top-level algebraic expression (see Fig. 2).

This expression needs some explanations. The dependent location path is connected to the outer expression using a D-Join ($\overset{\bowtie}{\rightarrow}$). The *D-Join* joins each tuple $t \in \mathcal{T}[//\text{student}]$ to all tuples returned by the evaluation of the dependent path $\mathcal{T}[\text{examination}/@\text{id}]$. For each t, $\mathcal{T}[\text{examination}/@\text{id}]$ is evaluated once, and free occurrences of variables in the dependent expression are substituted with the attribute values of t, i.e. the current context. At the end all resulting sequences are concatenated.[1]

The dependent expression, i.e. the evaluation using the D-Join, might produce duplicates for tuples from $\mathcal{T}[//\text{student}]$, hence the tid_A operator (tuple identifier) is needed to identify the tuples resulting from the outer expression.

The idea is to densely number the tuples, store this number in an attribute A, and use it later on to perform a duplicate elimination. To do this, we introduce an order-preserving duplicate elimination projection Π^{tid_A}, which removes multiple occurrences of the same tid-value A. It keeps the first tuple for a given A value and throws away the remaining tuples with the same value for A.

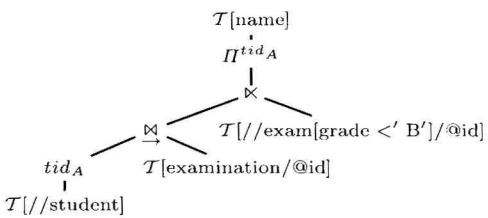

Fig. 2. Decorrelation for Query Q1

Clearly, the main advantage of this approach is that the independent expression is evaluated only once. In addition, if the Semi-Join's implementation uses a custom data structure (e.g. a hash-table) to improve performance, this data structure has to be initialized only once, compared to one initialization per student in the naïve correlated evaluation from Fig. 1. However, decorrelation comes at a price: The outer expression produces duplicates which have to be eliminated. Below, we show how we can avoid them using the novel Kappa-Join. Our evaluation in Sec. 5 confirms this claim.

3.3 Kappa-Join

To avoid the above-mentioned generation of duplicates, but nevertheless gain performance by avoiding unneeded evaluations of the independent expression, we introduce the Kappa-Join operator. It combines the advantages of the evaluation strategies from Fig. 2 and Fig. 1 into one operator and capitalizes on efficient implementation techniques.

Logical Definition. The *Kappa-Join* is a ternary operator, i.e. it has three argument expressions e_1, e_2, and e_3. It is defined by the equation

$$e_1 \kappa^{e_2}_{cn=cn'} e_3 := \sigma_{\mathfrak{A}_{x;exists}(e_2 \ltimes_{cn=cn'} e_3)}(e_1)$$

where cn is the context node resulting from the evaluation of e_2 and cn' the context node from e_3. As for conventional join operators, we denote the producer expressions

[1] In [23] this operator is called MapConcat.

e_1 and e_3 the as *outer producer* and *inner producer*, respectively. The second producer expression e_2 (in the superscript) is called *link producer* because it acts as a link between the outer and inner producer within the join predicate. The outer expression e_1 and the inner expression e_3 are independent expressions, i.e. they do not depend on any of the Kappa-Join's other arguments. In contrast, the expression e_2 is dependent on e_1.

Informally, the result sequence of the operator contains all tuples from the outer producer (e_1) for which there exists at least one tuple in the link producer (e_2), when evaluated with respect to the current tuple of e_1, that satisfies the predicate p which is a comparison from attributes of e_2 and attributes of the the inner producer (e_3).

Translation with Kappa-Join. There exist two alternatives to incorporate the Kappa-Join into an algebraic plan: (1) The Kappa-Join's definition matches the pattern that results from the canonical translation of correlation predicates (e.g. see Fig. 1). Hence, the Kappa-Join can *replace* this pattern after translation and, hence, already decorrelate during translation. (2) The other alternative is to *modify* the translation procedure such that a Kappa-Join is used for conjunctive correlation predicates.

Because our experiments (see Sec. 5) show that the Kappa-Join always dominates the canonical approach and simplifies the translation procedure, we have chosen the second alternative. Fig. 3 contains the resulting algebra expression for Q1. Here, the location path `//student` is mapped to the outer producer of the Kappa-Join . The inner location path `examination/@id` is the (dependent) link producer, and the independent expression `//exam[grade<'B']/@id` is mapped to the inner producer.

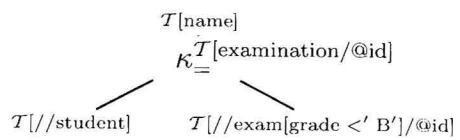

Fig. 3. Query Q1 with Kappa-Join

```
OPEN
1  while T ← INNERPRODUCER.NEXT
2      do HASHTABLE.INSERT(T)

NEXT
 1  while T1 ← OUTERPRODUCER.NEXT
 2      do
 3          LINKPRODUCER.OPEN(T1)
 4          while T2 ← LINKPRODUCER.NEXT
 5              do
 6                  if HASHTABLE.LOOKUP(T2)
 7                    then
 8                         LINKPRODUCER.CLOSE
 9                         return T1
10
11          LINKPRODUCER.CLOSE
12  return nil
```

Fig. 4. Pseudocode for the Kappa-Join

Implementation. The secret of the Kappa-Join lies in its simple, yet efficient implementation. It improves performance beyond that of the operator combination in its logical definition. Fig. 4 shows the pseudocode for the implementation of the Kappa-Join as an iterator [14].

In its `open` method, the Kappa-Join builds a data structure, e.g. a hash-table, containing the attributes from the inner producer that are part of the join predicate. In its `next` method, the Kappa-Join initializes the link producer for every tuple T_1 from its outer producer. Like a Semi-Join, it then probes the hash-table with tuples T_2 from the link producer until a matching one is found, and returns the outer tuple as soon as it finds a match. The Kappa-Join does not always enumerate all tuples from the dependent link producer, while building the hash-table once only. Hence, the worst-case complexity is $O(|e_1| \times |e_2| + |e_3|)$, assuming constant hash-table