ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

View Update Translation: Estimation and Rejection

Alby Issac

PG Scholar, Dept. of CSE, VJCET, Vazhakulam, Kerala, India

ABSTRACT: Views and their updates are an essential technology used in a broad range of database operations. However, updating database through views is an open problem. This work solves this problem using a data-oriented approach. It proposes a method that summarizes the source database of views, which functions as an update filter. The update filter tries to systematically reject view updates that are untranslatable by estimating the side-effects of the updates. This work focuses on SPJ views. The errors are estimated using join cardinality summary (JCard) derived from cardinality equivalence. An estimation algorithm is proposed. Finally, performance evaluation of the update operation is conducted and represented using different parameters.

KEYWORDS**:** View update, side-effect estimation, and relational database**.**

I. INTRODUCTION

In DBMSs, views can be defined as a prime feature that allows access to distinct parts of a database. Views are used in a broad range of emerging applications, such as data publishing, XML or RDF query rewriting [26], [1], and tracing facility in P2P networks. Since views are query to the databases, these applications make queries to the databases and also updates the views in database. The views that are updated by the user are then transferred to the source databases of the views, and after the translation, views and its source databases must be compatible.

Updating databases through views have been one of the principle problems in DBMS. In this system, a data-based perspective that profoundly exploits data summaries in source databases has been proposed (e.g., [27]).

Analysis of the views that are being updated is very expensive. In most of the cases, the view updates under different situations are NP-hard [16], [1].Two major steps in analyzing the view update are: side-effect analysis and view update translation. Most of the work directly translates view updates and it may be inefficient and time consuming. Different from that, this method concentrates on side-effect analysis. It has been noted that, most of the view updates causes side-effects. This work proposes to reject in advance, such view updates in the side effect detection. The updates that are free with side-effects are only allowed to translate to the source database. Hence, the method results in rejecting the view updates that causes side-effects and this helps to reduce the cost of update translation. The views created by select, project and join queries (SPJ) are discussed and the operations insert and delete of the view tuple are described in this paper.

Example 1. To illustrate the view update problem [1], let us consider an example as shown in Fig. 1. Suppose we have a view V that joins Product, Supplier, Order, and Agency. (For illustration purposes, this example omits integrity constraints.) Suppose we insert (S4, FL, P6, C3, A5) into V as indicated by u1. The only way is to insert (S4, FL) into Supplier and (A5, P6) into Agency. However, since tuple p2 is also joinable with tuple s1, the insertion of (S4, FL) and (A5, P6) cause an unspecified effect of inserting (S4, LA, P6, C3, A5) into V. In fact, we cannot translate u1 without causing extraneous tuple(s) in V.

This work which focuses on view update analysis consists of two major steps: Phase 1- side-effect analysis and Phase 2 - view update translation. In Phase1 the filter which is built using the database summary detects the side-effects i.e., predetermining the errors that are going to be happening in the source database after the translation.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

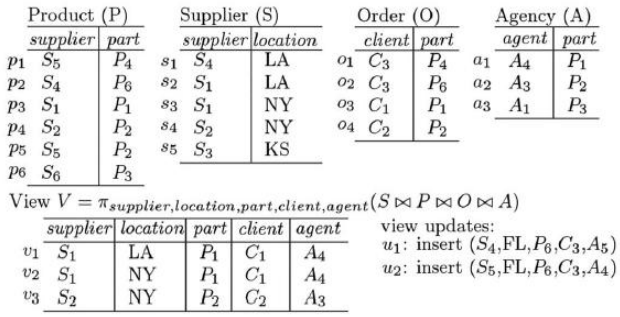

Fig. 1 Example of a view and source database tables [1].

The filter will check for the corresponding errors and list the aliases of the tuples that will be affected by the updating the view. The aliases will help to understand the tuples that are affected by the update operation. In phase2, the side effect free view updates are translated to the corresponding source databases.

View Update Problem: Given a view definition V of a relational database instance I, where the view V is V(I), and an update u on the view V, find a translated update u' on I such that u $\oplus$ V(I) = V(I $\oplus$ u') [1], [5], [26].

II. RELATED WORK

The view update problem is one of the principle problems in databases [17], [19], [20]. Most of the recent research and applications involves view updates [2], [14], [20], [24]. Due to space constraints this section presents works relevant to our approach. One may refer the literature for more complete reviews of the problem [6].

In 1974, Codd first reports the view update problem. Then the problem has been researched widely and deeply in database community. Previous methods [2] that used to solve the view update problem includes constant complement, clean source, collecting semantic information at view definition or update time, identity preservation, abstract data and bidirectional transformation.

Bancilhon and Spyratos proposed the seminal work on view complements [1], [17], [3]. He suggested that view updates could be translated without side-effects if the updates resulted in unchanged view complements (translation under a constant complement). The results of view complements were followed up in the proposal of "consistent views" [22], [23].

Yoshifumi Llasunaga [4] proposed a relational database view update translation mechanism. The view update translator consists of a translator body and four semantic ambiguity problem solvers of different types. The translation capability depends on the solvers available to the translator body and the problem solving capability they offer. Although, the four semantic ambiguity problems were explained, the design of such solvers is an open problem.

Keller [5], [6] proposed different algorithms for translating view updates for views involving selections, projections and joins.

Foster *et al.* [7] introduced combinators for bidirectional tree transformations. This provides a linguistic approach to the view update problem. This uses a pair of lens to work as the combinators.

Rom Langerak [8] proposed view updates in relational databases with an independent scheme. The independent scheme provides single view-tuple update.

Wang *et al.* [9] introduced the updating of xquery views published over relational data.

V. Braganholo *et al.* [10] in 2006 introduced a framework called PATAXO that help to update through XML views. The architecture uses an UXQuery processor and Update Manager. The UXQuery processor helps to process and translate the view definition query and the update manager validate the corresponding update and the result is given to RDBMS.

Bohannon *et al.* [11] developed a language for updating the views, the relational lenses. It is derived from the bi-directional transformations. This works as a pair of lenses which helps to perform some transformations such as Putget and Getput. The most important thing is the implementation primitive of those lenses.

After Braganholo's attempt [10], Kotidis *et al.* expressed a new hope for the update of views. He developed and produced physical ID's for view updates, which needs an intrusion to the physical layer of DBMS. For determining

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

updates of XML view updates, Boneva *et al*. [14] proposed tree automata techniques. It includes techniques with and without constraints. Liu *et al*. [24] proposed the view update analysis in exact XML context. Cong *et al*. [16], [28] studied the time complexities of various versions of view update.

Regarding filtering, Luo *et al*. [1], [29] proposed a filter for the view maintenance problem.

Y. Peng *et al*. [1] derived a filter called JCard, implemented using summary of the source database tuples and the cardinality equivalence. This work proposes an optimization of side-effect estimation to extend DBMSs with the capability to support practical view updates.

III. PROPOSED SYSTEM

Many researches are being done in the field of view update problem and several solutions have been suggested by different researchers. Most of the works focused on recognizing the problem and suggesting different methods for overcoming this problem. Recent progress on view updates includes bi-directional transformation of trees that permits operations for universal data synchronization and relational view updates.

After analyzing a few works which focuses on the view update problem it can be seen that it only focuses on the problem identification. The proposed system aims at identifying the problem and also detects the side-effects that are caused by different view updates in relational databases.

The proposed system also mainly focuses on the side effect estimation of view updates. It is seen that this has been an important work that focuses on estimating the error of these view updates. The work mainly does with select, project and joins queries. As an extension to this work, side effect estimation on a larger class of view definitions such as views with unions and selections with inequality can be taken into account.

The system works as follows:

1. The database which needs to be updated is selected.
2. The view of the corresponding database is created using SPJ query.
3. Then the update to the view is done by giving an insert tuple to the view.
4. The view update is taken by the filter and it checks all the tuples, using the tree structure of the database.

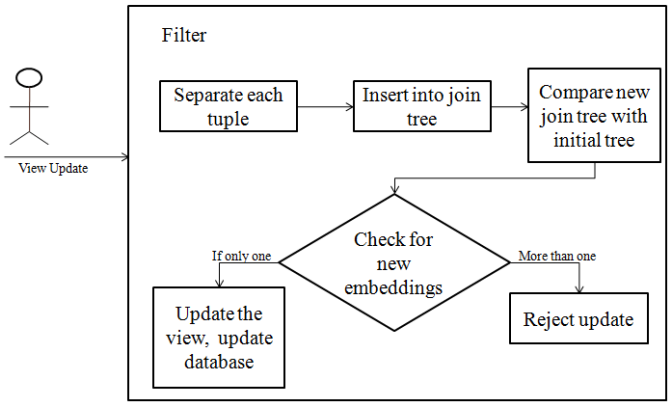

Fig. 2 Overview of the filter.

5. If the tuples can be inserted without any error, then it should be inserted as such into the view.
6. Then the update filter is updated using the new updated database.
7. If the update tuple is untranslatable, it shows as an error tuple.
8. The error tuple is rejected, and the filter shows the corresponding error that causes the untranslatable update.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

The proposed system as in Fig. 2 helps us to observe what changes are going to happen in the source database when an update is given. If the update causes side effect the system rejects the updates as well as it provides the errors that causes untranslatable updates in the database.

IV. SIDE-EFFECT ESTIMATION

Side-effect estimation of a view insertion tuple consists of two steps. The first step is to join the insert tuple with the source tuples in the source database graph. If the join result is not empty, then the inserted tuple forms additional view tuple(s) (*i.e.*, side-effects) and hence rejected. Otherwise, the second step estimates the join cardinality (JCard) of inserted tuple with the extended dangling tuples in the negative graph [1].

The JCard is specially designed for joins since joins are the most challenging in SPJ views. JCard has two components. (i) The first one is the summary of the dangling tuples [1] (ii) The second one is candidate tuples. These two components are used to estimate the size of the view.

Side-effect estimation by JCard is effectual because it summarizes the tuples by the equivalence classes [1]. Upon an insertion of a tuple, the average joinable tuples in a class is estimated by joining the insert tuple with the equivalence classes. The presence of side-effects is determined by the large number of new embeddings produced in each insertion.

A. *Side Effect Detector*

The side-effect estimation is done using a filter (side-effect detector) which is implemented using the summary of the database. Here the summary of the database is taken as a tree structure. This tree formed from the view is called the Join-tree and is stored as an initial tree (Fig 2.).

When an update is given by the user to the corresponding view that is formed from the SPJ query, the filter takes the input tuple and separates each of the tuple using the comma (,) separation. Then each of the comma separated tuples is taken into the tree Join-tree structure and it keeps a copy of the initial tree. The new tuples are then inserted into the tree and forms a new Join-tree which includes all the new tuples as nodes.

Now, the filter checks for the errors. The filter compares both the Join-trees, the initial tree and the updated tree, and if it finds more than one complete embedding on the updated tree than the initial tree, then it is considered as side-effect. When the insertion produces a large or moderate number of new embeddings (side-effects), the average count is sufficient to signify the presence of side-effects. If the filter finds only one new embedding, then it is considered as a side-effect free update. And then the corresponding update is given to the update translator which converts the tuples as new insertions into the view table as well as the source database tables. The updates with side-effects are caught by the detector and then the aliases of the tuples in the source database which will be affected by this update are displayed to the user.

The user can now understand the tuples which causes the risk and also helps to understand the risk of that corresponding update. The updates with side-effects are then rejected and the view and the source database remains the same. Those without side-effects are then passed to the view as well as the source database and the view and the source database will be consistent after the update.

B. *Implementation Of Join-Tree*

The tree implementation has been described with the help of the example database shown in Fig 1. For the ease of understandability, consider the database tables given in Fig: 1. It mainly has four tables namely supplier, product, order and agency. Each of the table has a set of rows and columns describing the values of each of the tables. Here each tuples in the row or column in one table is related to the tuple in the next table. The table information is taken out in order to build the tree structure. Now the view is created using the SPJ query and is created as a view table. The tuples from the view table is selected and form the tree.

The tree is implemented as follows:

For each of the tuples in the tables, aliases are created to easily recognize the pair of tuples by the filter.

1. For that, four set alias names are created namely S, P, O and A which represents Supplier, Product, Order and Agency.
2. The alias S will recognize the pair of tuples in the Supplier database i.e., supplier and location. Then P represents the tuples supplier and part, O represents client and part, and A represents agency and part. Here it is evident that S and P is related with supplier, P and O with part and O and A with part.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

3. The set of aliases is created by the source database name combined with the number of rows or the set of tuples in each database.

Now the tree is created as it takes all the tuples from alias S and relates it with the tuples in the alias P. So a tree with link SP is created. Similarly AP, OP, SOP, SAP, OAP and SOAP are created. Then the tree which is having the complete embedding of the SPJ query is taken, which will be the SOAP tree. This is stored as the Join-tree.

C. *Creating Aliases*

For the ease of understanding and identification of the tuples which are affected by the update, a set of aliases is created by the system. The aliases are created by using the id of each of the tuples in the database. For example, consider Fig 1. In the figure for each of the rows, there is an id given to the tuples. For the tuples S1, S2, S3 and S4 the id is supplier and for the tuples LA, NY, and KS the id is location. These id's are under the Supplier database, thus the alias created for this database be S. Such that for each of the id's in the database order, agency and product four aliases can be created, such as S, P, O and A.

D. Inserting A Tuple And View Update

Pseudo code:

1. For each insert operation, takes the input tuple given by the user.
2. For each view update tuple, separate the tuples (using commas).
3. Create a new join tree using the view table tuples, including the new insertions.
4. Then compare the tree with the join tree of the view.
5. If the new tree is completely filled, and is having only one new embedding, then accept the update.
6. If the new tree is completely filled, and is having only one new embedding, then accept the update.
7. Each of the side-effect free update is then passed to the corresponding view and to the source database.
8. If the new tree is filling missing tuples, and also creates tree with large no. of new embeddings, then reject the update.
9. For each of the rejected update, display the aliases of the set of tuples in the database that will be affected by it.

E. *Deleting A Tuple*

Delete Operation works as same as the insert operation itself. If a particular tuple needs to be deleted from the source database, then a delete query is given to the view table. When the delete query is submitted to the view table, it checks whether there exists the corresponding database and the table to drop the values. If there exists, then the corresponding tuple gets deleted from the join tree. And if the dropping of tuple from the join tree shows any error then it is not translated to the table as well as the source database. If the tuples can be deleted without any error from the join tree, then the corresponding values are dropped from view table and are then translated to the source database. So that the query gets executed and the set of tuples will be dropped from the source database.

V. PERFORMANCE EVALUATION

The whole experiment is conducted using a synthetic dataset. Performance is evaluated with the number of updates (insertion, deletion).

Experimental Setting: The experiments are conducted on a dual-core 2.4GHz CPU running Window 7. Implementation was written in Python, using MySQL 5.1. The experiment was done using a Wamp server which uses Apache 2 and Python 2.7.

Error metrics: Let M be the set of insertions tested. Denote S+ and S- is the real-positive and real-negative insertions in M, respectively [1]. In this experiment, set |S+| = |S-|. S+ as true positives and S- as true negatives. Let A+ be the estimated positive insertions in S- and A- the estimated negative insertions in S+. Define the false negative (fn) to be $\frac{A-}{S+}$ and the false positive (fp) to be $\frac{A+}{S-}$.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Now define precision and recall as: Precision is the fraction of retrieved instances that are relevant i.e. $\frac{tp}{tp+fp}$ and recall is the fraction of relevant instances that are retrieved i.e. $\frac{tp}{tp+fn}$ where tp is true positives, fp is false positives and fn is false negatives.

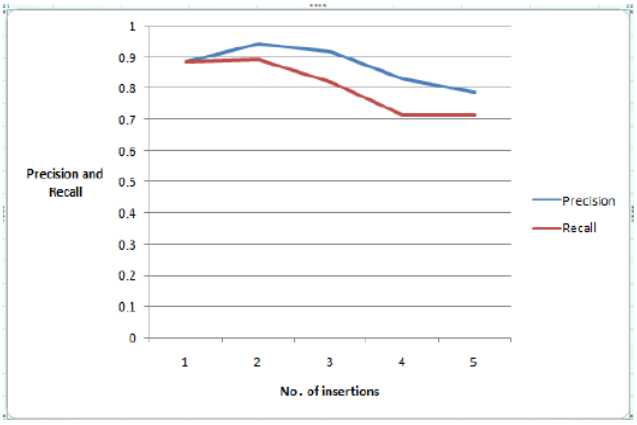

Fig. 3 No. of insertions V/S Precision and Recall.

The performance evaluated using this precision and recall is shown in Fig 3 and Fig 4. As the number of insertion increase, precision and recall increases. After a limited set of insertions, the precision will be decreasing, and recall stands to be stable. High precision [30] means that an algorithm returned substantially more relevant results than irrelevant, while high recall means that an algorithm returned most of the relevant results.

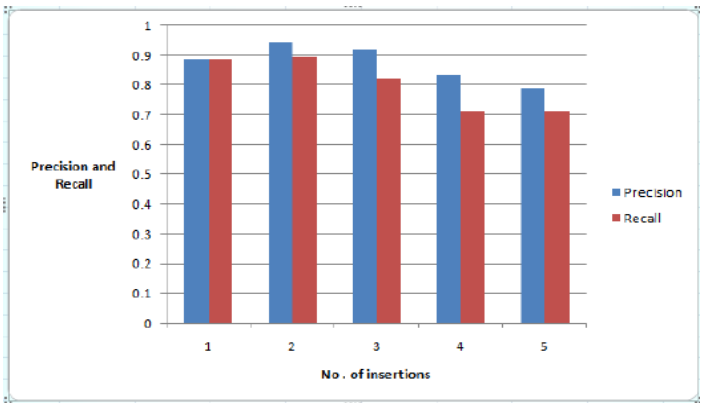

Fig. 4 Variations in Precision and Recall for each set of insertions.

VI. CONCLUSION

A data-based approach can offer practical assistance for the view update problem. Concretely, this work proposes a side-effect detector for SPJ views that estimates or detects side-effects caused by a view update and avoids costly update translations by rejecting the untranslatable updates early. The base of the detector was the update filter. The system mainly works with data manipulation of databases. The updates include insertion, deletion of the database tuple through the views. The update filter is a join cardinality summary JCard that consists of structures that summarize the source of view tuples. The system will help to, detect all the untranslatable updates, provides the error in database and helps to correct the updates.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

ACKNOWLEDGMENT

First and Foremost the authors thank God Almighty for His divine grace and blessings in making all this possible. The authors gratefully acknowledge all the staff members of Department of Computer Science and Engineering, Viswajyothi College of Engineering and Technology, for providing their motivation, constant support and encouragement and for guiding with patience and enthusiasm. Also the authors thank students of M.Tech, Department of Computer Science and Engineering, Viswajyothi College of Engineering and Technology, Vazhakulam, for their assistance and help during the course of this project.

REFERENCES

1. Yun Peng, Byron Choi, Jianliang Xu, Haibo Hu, and Sourav S. Bhowmick, "Side-E_ect Estimation: A Filtering Approach to the View Update Problem," IEEE Transactions On Knowledge And Data Engineering, VOL. 26, NO. 9, Sept. 2014.
2. H. Chen and H. Liao, "A comparative study of view update problem," in Proc. DSDE, Bangalore, India, 2010, pp. 83-89.
3. F. Bancilhon and N. Spyratos, "Update Semantics of Relational Views," Inria, France, ACM Transactions on Database Systems, Vol. 6, NO. 4, December 1981, Pages 557-575.
4. Yoshifumi Llasunaga, Yatabe-machi, Tsukuba-gun , Ibaraki-ken, "A Relational Database View Update Translation Mechanism," in Proc. DBLP, Singapore, August 27-31, 1984.
5. A. Keller, "Algorithms for translating view updates to database updates for involving selections, projections, and joins," in Proc. PODS, New York, NY, USA, 1985, pp. 154-163.
6. A. Keller, "The role of semantics in translating view updates," Computer, vol. 19, no. 1, pp. 63–73, Jan. 1986.
7. J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for bi-directional tree transformations: A linguistic approach to the view update problem," in Proc. POPL, 2005, pp. 233–246.
8. Rom Langerak, "View Updates in Relational Databases with an Independent Scheme, "in Proc. DBLP, ACM Transactions on Database Systems,1990, pp. 40-66.
9. L. Wang, M. Mulchandani, and E. Rundensteiner, "Updating XQuery views published over relational data: A round-trip case study," in Proc. XSym, 2003, pp. 223–237.
10. Vanessa Braganholo, Susan Davidson, Carlos A Heuser, "PATAXO: A framework to allow updates through XML views," in proc. DBLP ACM Trans. Database Systems, 2006, pp. 839-886.
11. A. Bohannon, B. C. Pierce, and J. A. Vaughan, "Relational lenses: A language for updatable views," in Proc. PODS, New York, NY, USA, 2006, pp. 42-67.
12. Y. Kotidis, D. Srivastava, and Y. Velegrakis, "Updates through views: A new hope," in Proc. ICDE, 2006, p. 2.
13. A. Bohannon, B. C. Pierce, and J. A. Vaughan, "Relational lenses: A language for updatable views," in Proc. PODS, New York, NY, USA, 2006, pp. 42–67.
14. I. Boneva et al., "View update translation for XML," in Proc. ICDT, New York, NY, USA, 2011, pp. 42–53.
15. P. Buneman, S. Khanna, and W.-C. Tan, "On propagation of deletions and annotations through views," in Proc. PODS, New York, NY, USA, 2002, pp. 150–158.
16. G. Cong,W. Fan, and F. Geerts, "Annotation propagation revisited for key preserving views," in Proc. CIKM, New York, NY, USA, 2006, pp. 632–641.
17. S. S. Cosmadakis and C. H. Papadimitriou, "Updates of relational views," J. ACM, vol. 31, no. 4, pp. 742–760, Oct. 1984.
18. Y. Cui and J. Widom, "Run-time translation of view tuple deletions using data lineage," Standford University, Standford, CA, USA, Tech. Rep., 2001.
19. U. Dayal and P. A. Bernstein, "On the updatability of relational views," in Proc. VLDB, 1978, pp. 368–377.
20. U. Dayal and P. A. Bernstein, "On the correct translation of update operations on relational views," ACM Trans. Database Syst., vol. 7, no. 3, pp. 381–416, Sep. 1982.
21. L. Fegaras, "Propagating updates through XML views using lineage tracing," in Proc. ICDE, Long Beach, CA, USA, 2010, pp. 309–320.
22. G. Gottlob, P. Paolini, and R. Zicari, "Properties and update semantics of consistent views," ACM Trans. Database Syst., vol. 13, no. 4, pp. 486–524, Dec. 1988.
23. J. Lechtenbörger, "The impact of the constant complement approach towards view updating," in Proc. PODS, New York, NY, USA, 2003, pp. 49–55.
24. J. Liu, C. Liu, T. Haerder, and J. X. Yu, "Updating typical XML views," in Proc. DASFAA, Busan, South Korea, 2012, pp. 126–140.
25. A. N. Swami and K. B. Schiefer, "On the estimation of join result sizes," in Proc. EDBT, 1994, pp. 287–300. B. Choi, G. Cong, W. Fan, and S. D. Viglas, "Updating recursive XML views of relations," in Proc. ICDE, Istanbul, Turkey, 2007, pp. 766–775.
26. B. Choi, G. Cong, W. Fan, and S. D. Viglas, "Updating recursive XML views of relations," in Proc. ICDE, Istanbul, Turkey, 2007, pp. 766–775.
27. B. Choi, G. Cong, W. Fan, and S. D. Viglas, "Updating recursive XML views of relations," in Proc. ICDE, Istanbul, Turkey, 2007, pp. 766–775.
28. G. Cong, W. Fan, F. Geerts, J. Li, and J. Luo, "On the complexity of view update analysis and its application to annotation propagation," IEEE Trans. Knowl. Data Eng., vol. 24, no. 3, pp. 506–519, Mar. 2012.
29. G. Luo and P. S. Yu, "Content-based filtering for efficient online materialized view maintenance," in Proc. CIKM, New York, NY, USA, 2008, pp. 163–172.
30. https://en.wikipedia.org/wiki/Recall_and_precision.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

BIOGRAPHY

Alby Issac received her B.TECH degree from University of Calicut, Malappuram, India in 2013. She is currently pursuing her M.TECH degree in Computer Science and Engineering at VJCET, MG University, Kerala, India. Her area of interest includes image processing, Data mining, Web Security etc.

Amel Austine did his Master's in Software Engineering from Karunya University in the year 2011. He joined Viswajyothi College of Engineering and Technology and is working as Assistant Professor in Department of Computer Science since 2008. His areas of interest are image processing, software engineering, data mining and computer networks. He has carried out many works related to his fields of interest during his academic studies as well as his professional life.

..