An Empirical Data Cleaning Technique for CFDs

Satyanarayana Mummana[1], Ravi kiran Rompella [2]

[1] *Asst Professor, Dept of Computer Science and Engineering, Avanthi Institute of Engineering & Technology, Visakhapatnam, Andhra Pradesh.*

[2] *Final M.Tech Student, Dept of Computer Science and Engineering, Avanthi Institute of Engineering & Technology, Visakhapatnam, Andhra Pradesh.*

Abstract:- Data cleaning is a basic data preprocessing technique for before forwarding the data to data mining approach ,but it leads to an intresting research area in the field of data mining. Data cleaning is the process of finding and deleting noisy data/records from the database. The simplest technique used for data cleaning is based on Functional Dependencies. As FDs works on entire instance of a table we introduced a new technique called Conditional Functional Dependencies. CFDs are like if then rules. The dependence between the columns of a table are represented as conditions using functions.. For example if we consider a employee table which maintains the employee name,id,city,pincode and etc. In this table the employees who are belongs to the same city, are all may have the same pincode, So that we can generate a FD that city--->pincode. CFD means using specific condition for the FD. ex:city=vizag---->pincode=531005. The main agend of our project is to find the CFD violated rows in a table using the created CFDs. These CFDs violated rows are deleted to correct data.

I.INTRODUCTION

Data mining, is the extraction of hidden predictive information from large databases and powerful new technology with great potential to help companies focus on the most important information in their data warehouses. The tools predict future trends and their behaviours allowing businesses to make proactive, knowledge-driven decisions. The prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. The tools can answer business questions that traditionally were to resolve takes much time. They scour databases for finding predictive information that experts may miss because it lies outside their expectations Data mining (also known as Knowledge Discovery in Databases - KDD) has been defined as nontrivial extraction of implicit and potentially useful information from data and uses machine learning visualization techniques to discover and present knowledge in a form which is easily comprehensible to humans. The components of data mining technology have been under development for decades, in research areas such as artificial intelligence. Today, the maturity of these techniques and coupled with high-performance relational database engines and broad data integration efforts and these technologies practical for current data warehouse environments.

II.RELATED WORK

Data cleaning, also called data cleansing deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data. Problems of data are present in single data collections and those are files and databases due to misspellings during data entry and missing information or other invalid data. When multiple data sources need to integrated in data warehouses and database systems or global web-based information system and they need for data cleaning increases because the sources often contain redundant data in different representations. To provide access to accurate and consistent data and the consolidation of different data representations and elimination of duplicate information become necessary. Data warehouses require and provide extensive support for data cleaning and continuously refresh huge amounts of data from a variety of sources so the probability that some of the sources contain noisy data is high. Data warehouses are used for decision making, then that the correctness of their data is vital to avoid wrong conclusions.

For instance, duplicated or missing information will produce incorrect or misleading. Due to the wide range of possible data inconsistencies and the sheer data volume and the data cleaning is considered to be one of the biggest problems in data warehousing. The presence of errors and inconsistencies in data dramatically reduce the value of data, making it worthless and even harmful. The statistics estimated that data quality costs US businesses $600 billion annually is also estimated that data cleaning and complex process, accounts for 30 to 80% of the development time and budget in most data warehouse projects. Studies conducted by many researchers forecast that more than 50 percent of data warehouse projects will have limited success or will be outright failures and the result of the lack of attention to data quality issues. As a result of the previous statistics and studies and there is an increasing demand for data cleaning tools that automatically detect and effectively remove inconsistencies and errors from the data. Dirty data often arises due to changes in use and perception of the data and violation of integrity constraints (or lake of such constraints).The errors in a database often emerge as violations of integrity constraints.

Constraint-based data cleaning has mostly focused on two concepts and repair and a consistent answer to the query. A repair is to find another database that is consistent and differs minimally from the original database. Query

answer is to find an answer to a given query in every repair of the original database, without editing the data. The limitations in traditional dependencies lead the researchers in data cleaning to revive actions by considering extensions of FDs and INDs (Inclusion Dependencies) defined to as Conditional Functional Dependencies (CFDs) and Conditional Inclusion Dependencies (CINDs). The extended functional dependencies specifying patterns of semantically related values; these patterns impose conditions on what part of the relation the dependencies are to hold and which combinations of values should occur together (Fan et al., 2009).

III.EXISTING SYSTEM

Unfortunately little previous work has studied this issue. There has also been recent work on *constraint repair*, which specifies the consistency of data in terms of constraints and which detects inconsistencies in the data as violations of the constraints. The previous work on constraint repair is mostly based on traditional dependencies (*e.g.*, functional and full dependencies, etc) and they were developed mainly for schema design because are often insufficient to capture the semantics of the data, it shows in example below.

CC	AC	PN	NM	STR	CT	ZIP
1	890	1234566	Mike	Tree Ave	PHI	0891
1	890	2233445	Rick	Tree Ave	NYC	0891
1	212	2233445	Dick	Elm str	NYC	0891
1	212	2233445	Jack	Elm str	NYC	0210
1	215	3333333	Mack	Oak ave	EDI	0210
4	131	4444444	Lack	High st.	EDI	0238

Fig1:An instance of the cust relation

Example 1.1: Consider the following schema, which specifies a customer say that customer's phone (country code (CC), area code (AC), phone number (PN)), name (NM), and address (street (STR), city (CT), zip code (ZIP)). An instance of cust is shown in Fig. 1. Traditional functional dependencies (FDs) on a cust relation may include:

f1: [CC, AC, PN]→[STR, CT, ZIP]

f2: [CC, AC]→[CT]

(Recall the semantics of an FD: f2 requires that two customer records with the same country- and area-codes). Traditional FDs are to hold on all the tuples in the relation (indeed they do on Fig. 1). In contrast, the following constraint is supposed to hold only when the country code is 44. That is, for customers in the UK and ZIP determines STR:

φ0: [CC = 44, ZIP]→[STR]

In other words, φ0 is an FD that is to hold on the subset of tuples that satisfies the pattern "CC = 44" on the entire customized relation and generally not considered an FD in the standard definition since φ0 includes a pattern with data values in its specification. The constraints are again not considered FDs:

φ1: [CC = 01, AC = 908, PN]→[STR, CT = MH, ZIP]

φ2: [CC = 01, AC = 212, PN]→[STR, CT = NYC, ZIP]

φ3: [CC = 01, AC = 215] →[CT =PHI]

The first constraint φ1 assures that only in the US (country code 01) and for area code 908, if two tuples have the same PN and they must have the same STR and ZIP values and the city must be MH. Similarly and the φ2 assures that if the area code is 212 then the city must be NYC; and φ3 specifies that for all tuples in the US and with area code 215 and their respective PHI must be (irrespective of the values of the other attributes). Let us observe that φ1 and φ2 refine the FD f1 given above and then φ3 refines the FD f2. This refinement essentially enforces a binding of semantically related data values. Note that while tuples t1 and t2 in Fig. 1 do not violate f1, they violate its refined version φ1, since the city cannot be NYC if the area code is 908. In this example, the constraints φ0, φ1, φ2 and φ3 capture a initial requirement part of the semantics of the data. They cannot be expressed as standard FDs and are not considered in previous work on data cleaning. Indeed, constraints that hold conditionally may arise in a number of domains. For example, an employee's pay grade may determine her title in some parts of an organization but not in others; an individual's address may determine his tax rate in some countries while in others it may depend on his salary, etc. Further, dependencies that apply conditionally appear to be particularly needed when integrating data and dependencies that hold only in a subset of sources will hold only conditionally in the integrated data.

IV.PROPOSED SYSTEM

We introduce a novel extension of traditional methods referred to as conditional functional dependencies (CFDs) and they are capable of capturing the notion of "correct data" in these situations. The process goes like this, first we will generate some CFDs based on the existing data then we will construct a query to find the CFD violated rows this will result the set of rows which are violated CFDs and lastly violated rows will be treated as noisy data and those will be deleted to get the correct data.

A.Detecting CFD Violations

A first step for data cleaning is the efficient detection of constraint violations in the data. In this section we develop techniques to detect violations of CFDs. Given an instance I of a relation schema R and a set Σ of CFDs on R, it is to find all the inconsistent tuples in I, i.e., the tuples that (perhaps together with other tuples in I) violate some CFD in Σ. We first provide an SQL technique for finding violations of a single CFD, and then generalize it to validate multiple CFDs.

QCϕ 2 select t from cust t, T2 tp
where t[CC] _ tp[CC] AND t[AC] _ tp[AC] AND t[PN] _ tp[PN] AND
(t[STR] __ tp[STR] OR t[CT] __ tp[CT] OR t[ZIP] __ tp[ZIP])

QVϕ2 select distinct t[CC], t[AC], t[PN] from cust t, T2 tp
where t[CC] _ tp[CC] AND t[AC] _ tp[AC] AND t[PN] _ tp[PN]

group by t[CC], t[AC], t[PN]
having count(distinct t[STR], t[CT], t[ZIP])> 1
Figure 5. SQL queries for checking CFD ϕ2

B. Checking a Single CFD with SQL

Consider a CFD ϕ = (X → Y, Tp). The following two SQL queries suffice to find the tuples that violate ϕ:

QCϕ
select t from R t, Tp tp
where t[X1] _ tp[X1] AND . . . AND t[Xn] _ tp[Xn] AND
(t[Y1] __ tp[Y1] OR . . . OR t[Yn] __ tp[Yn])
QVϕ
select distinct t.X from R t, Tp tp
where t[X1] _ tp[X1] AND . . . AND t[Xn] _ tp[Xn]
group by t.X having count(distinct Y)> 1

where Xi (resp. Yj) ranges over attributes in X (resp. Y); t[Xi] , tp[Xi] is a short-cut for the Structured Query Language expression (t[Xi]
= tp[Xi] OR tp[Xi] = ‘ ’), while t[Yj] __ tp[Yj] is a shorthand
for (t[Yj] _= tp[Yj] AND tp[Yj] _= ‘ ’).

Intuitively, detection is a two-step process, each conducted by a query. Initially, query QCϕ detects single-tuple violations, i.e., the tuples t in I that match some pattern tuple tp ∈ Tp on the X attributes, but t does not match tp in the Y attributes due to a constant value tp[Yi] different from value t[Yi]. That is, QCϕ finds inconsistent tuples based on differences in the constants in the tuples and Tp patterns. On the other hand, query QVϕ finds multi-tuple violations, i.e., tuples t in I for which there exists a tuple t_ in I such that t[X] = t_[X] and moreover, both t and t_ match a pattern tp on the X attributes, but t[Yj] _= t_[Yj] for some attribute Yj in Y. Query QVϕ uses the group by clause to group tuples with the same value on X and it counts the number of distinct instantiations in Y . If there is more than one instantiation there is a violation. It catches both tuples t and t’ mentioned above as violations, although it is possible that both pass the test of query QCϕ. To be precise, QVϕ returns only the X attributes of inconsistent tuples (this is caused by the group by). However, this has the advantage that the ouput is more concise than when we would return the complete tuples. Moreover, the complete tuples can be easily obtained from the result of the two queries by means of a simple SQL query. Example 4.1: Recall CFD ϕ2 given in Fig. 2. Over a cust instance I, the SQL queries QCϕ2 and QVϕ2 shown in Fig. 5 determine whether or not I satisfies ϕ2. Executing these queries over the instance of Fig. 1, it returns tuples t1, t2(due to QCϕ2), and t3 and t4 (due to QVϕ2). _
ϕ4 = (cust:[CC, AC, PN]→[STR, CT, ZIP], T4), where T4 is

CC	AC	PN	STR	CT	ZIP
-	-	-	-	-	-
01	089	-	-	MH	-
01	202	-	-	NY	-
-	-	@	@	-	@
01	512	#	@	PHI	@

Fig:6 Merged Ω_2 and Ω_3

A salient feature of our SQL translation is that tableau Tp is treated an ordinary data table. Therefore, each query is bounded by the size of the embedded FD X → Y in the CFD, and is independent of the size (and contents) of the (possibly large) tableau Tp.

C. Validating Multiple CFDs

A naive way to validate a set Σ of CFDs is to use one query pair for each CFD ϕ in Σ. This approach requires2 × |Σ| passes of the underlying relation. We next present an alternative approach that only requires two passes. The key idea is to generate a single query pair to check constrains in Σ. Our proposed approach works in two phases. First phase the process starts performing a linear scan of all the tableaux belonging to CFDs in Σ and merges them, generating a single tableau called Tableau TΣ is such that it captures the constraints expressed by all the tableaux of the CFDs in Σ. Then, in its second phase, it generates a query pair that finds inconsistent tuples violating CFDs in Σ. 4.2.1 Merging Multiple CFDs Consider a set Σ which, without loss of generality, contains just two CFDs ϕ and ϕ_ on R, where ϕ = (X → Y, T) and ϕ = (X’ → Y ‘, T’). There are two main challenges for the generation of the merged tableau TΣ. The first challenge is that tableaux T and T _ may not be union-compatible, i.e., X _= X_ or Y _= Y _. We thus need to extend tableau T (resp. T _) with all the attributes in Z = (X∪Y)−(X_∪Y _) (resp. (X_ ∪ Y _) − (X ∪ Y) for T _). For each attribute A in Z and each tuple tc in the original tableau T, we set the value of tc[A] to be a special symbol denoted by ‘@’, which denotes intuitively a don’t care value. After this extension, the resulted tableaux are union-compatible. Then, tableau TΣ is defined to be their union. The presence of “@” and we reformulate CFD satisfaction. Consider a tuple tc[X, Y] in a tableau that includes ‘@’. We use Xfree tc and Y free tc to denote the subset of X and Y attributes of tc that is ‘@’-free, i.e., it has no ‘@’ symbol. I→R satisfies the set CFD ϕ , denoted by I |= ϕ, if for each pair of tuples t1 and t2 in the relation I and for every tuple tc in the pattern tableau Tp of ϕ, if t1[Xfree tc] = t2[Xfree tc] _ tc[Xfree tc], then t1[Y free tc] = t2[Y free tc] _ tc[Y free tc]. For the second challenge, consider the translation of a single CFD into an SQL query pair. Note that the translation assumes implicit knowledge of which attributes are in the

id	CC	AC	CT
1	-	-	@
2	1	1	@
3	4	--	-
4	@	@	-

id	CT	AC
1	-	@
2	PHI	@
3	GLA	@
4	@	-

Fig:7

X and Y sets and treats the translation of each attribute set differently. Now, consider two simple CFDs on

R, namely, ϕ = (A → B, T) and ϕ_ = (B → A, T _). Suppose that we have made the tableaux of the CFDs union-compatible. One might want to take the union of these two tableaux to generate TΣ. How can we translate tableau TΣ into a query and Clearly we cannot directly use the translation given earlier since we do not know how to treat the join of an attribute like, say, A. Attribute A is in X for tuples coming from empty set while it is part of Y for tuples coming from ϕ_. Thus we need to distinguish the two sets of tuples and treat the translation of each set separately. We accommodate this by splitting the tableau T of each CFD ϕ = (R : X → Y, T) into two parts, namely, TX and T Y , one tableau for the X and one for Y attributes of ϕ. Then, tableaux TX Σ (and similarly T Y Σ) is generated by making all the TX tableaux in Σ union-compatible. Note that an attribute can appear in both TX Σ and T Y Σ . To be able to restore pattern tuples from TX Σ and T Y Σ , we create a distinct tuple id t.id for each pattern tuple t, and associates it with the corresponding tuples in TX Σ and T Y Σ . For example, consider CFD ϕ3 shown in Fig. 2 and ϕ5 = (cust : [CT] →[AC], T5), where T5 consists of a single tuple (,). Figure 7 shows their merged TX Σ and T Y Σ tableaux. Note that attributes CT and AC appear in both tableaux. 4.2.2 Query Generation During the second phase of our approach, we translate tableau TΣ into a single SQL query pair. This translation, however, introduces new challenges. Recall that query QVϕ , for some CFD ϕ = (R : X → Y, T), requires a group by clause over all the X attributes. Let us consider tableau TX Σ in Fig. 7. It is not hard to see that if we use the group by clause over all the attributes in TX Σ , we are not going to detect all (if any) inconsistencies since, for example, for the first three tuples in TX Σ the '@' in attribute CT indicates that, while detecting inconsistencies, we should only group by the first two attributes and ignore the value of attribute CT. Similarly for the last tuple in TX Σ , the '@' in attributes CC and AC indicates that while detecting inconsistencies for these tuples then after we should only consider the value of CT. The example suggests that our SQL query should change the set of group by attributes, based on the contents of each tuple. In what follows, we show how this can be achieved while still keeping the query size bounded by the size of the embedded FD X → Y and independent of the size of the tableau. Central to our approach is the use of the case clause of SQL (supported by commercial DBMS like DB2). Consider the merged tableaux TX Σ and T Y Σ from a set Σ of CFDs over a relation schema R and let I be an instance of R. Then, the following two SQL queries can be used to detect inconsistent tuples of I violating ϕ:

QCΣ
select t from R t, TX Σ tXp , TY
Σ tYp
wheretXp.id = tYp.id AND
t[X1] _ tXp
[X1] AND . . . t[Xn] _ tXp
[Xn] AND (t[Y1] __ tYp[Y1] OR . . . t[Yn] __ tYp[Yn])
QV
Σ select distinct tM.X from Macro tM
group by tM.X
having count(distinct Y)> 1
where Macro is:
select (case tXp
[Xi] when "@" then "@" else t[Xi] end)AS Xi . . .
(case tYp
[Yj] when "@" then "@" else t[Yj] end)AS Yj . . .
from R t,TX Σ tXp, TY Σ tYp wheretXp.id = tYp.id AND
t[X1] _ tXp
[X1] AND . . . AND t[Xn] _ tXp [Xn]
where t[Xi] _ tp[Xi] now accounts for the '@' and is a short-hand for (t[Xi] = tp[Xi] OR tp[Xi] = ' ' OR tp[Xi] = '@'), while t[Yj] __ tp[Yj] is a short-hand for (t[Yj] _= tp[Yj] AND tp[Yj] _= ' ' AND tp[Yj] _= '@').

More specifically, query QCΣ is similar in spirit to the SQL query that checks for inconsistencies of constants between the relation and the tableau, for a single CFD. The only difference is that now the query has to account for the presence of the '@'symbol in the tableau. Now, we turn our attention to relation Macro which is of the same sort as TX Σ and T Y Σ (we rename attributes that appear in both tableaux so as not to appear twice). Relation Marco is essentially the join on X of relation I with the result of the join on tuple id t.id of the two tableaux. The value of each attribute, for each tuple tM in Marco, is determined by the case clause. In more detail, tM[Xi] is set to be '@' if tXp [Xi] is '@', and is t[Xi] otherwise find similarly for tM[Yj] value. Note that relation I is not joined on Y with the tableaux so if for some tuple t with t[X] _ tXp [X], there exists an attribute Yj with tYp [Yj] a constant and t[Yj] _= tYp[Yj] (i.e., t is inconsistent w.r.t. tp) then tM[Yj] is set to be t[Yj]. This creates no problems since this inconsistent tuple is already detected byQCΣ. Intuitively, Macro considers each tuple in the tableau, and uses it as a mask over the tuples of the relation. If the tableau tuple indicates a don't care value for an attribute, all the (possibly different) attribute values in the relation tuples are masked and replaced by an '@' in Macro. Figure 8 shows the result of joining the fourth tuple of tableaux TX Σ and TY Σ in Fig. 7 with the cust relation of Fig. 1. Note that the query masks the attributes values of CC and AC. This masking allows the subsequent group by over X to essentially consider, for each tuple, only the subset of X that does not have any don't care values. Note that although X = {CC, AC, CT}, the group by by queryQV Σ essentially performs a group by over only attribute CT. The query returns the NYC tuples which violate ϕ5.

CC	AC	CT	CT'	AC'
@	@	NYC	@	908
@	@	NYC	@	212
@	@	PHI	@	215
@	@	EDI	@	131

Fig:8

In this way we generate a single pair of SQL queries to validate a set Σ of CFDs, while guaranteeing that the queries are bounded by the size of the embedded FDs in

Σ, independent of the size of the tableaux in Σ. Furthermore, to validate Σ only two passes of the database is required.

V.CONCLUSION

We have introduced CFDs and shown that CFDs can express semantics of data fundamental to data cleaning. For applications of CFDs in data cleaning, we have developed SQL-based techniques for detecting inconsistencies as violations of CFDs. We have also experimentally evaluated our detection techniques. Even the CFDs is a good technique for the data cleaning, it takes more computation time for query joining and record retrievals when compared with simple FDs. If we are able to decrease this computation time in CFDs we can produce a good data cleaning mechanism than this.

REFERENCES

[1] M. Arenas and L. Libkin, "A Normal Form for xml Documents," ACM Trans. Database Systems, vol. 29, pp. 195-232, 2004.
[2] P. Atzgeni and V.D. Antonellis, Relational Database Theory. The Benjamin/Cummings Publishing Company, Inc., 1993.
[3] J. Bauckmann, U. Leser, and F. Naumann, "Efficiently Computing Inclusion Dependencies for Schema Discovery," Proc. Second Int'l Workshop Database Interoperability, 2006.
[4] C. Beeri, M. Dowd, R. Fagin, and R. Statman, "On the Structure of Armstrong Relations for Functional Dependencies," J. Assoc. for Computing Machinery, vol. 31, no. 1, pp. 30-46, 1984.
[5] S. Bell, "Discovery and Maintenance of Functional Dependencies by Independencies," Proc. Workshop. Knowledge Discovery in Databases (KDD '95), pp. 27-32, 1995.
[6] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional Functional Dependencies for Data Cleaning," Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE), pp. 746-755, 2007.
[7] T. Calders, R.T. Ng, and J. Wijsen, "Searching for Dependencies at Multiple Abstraction Levels," ACM Trans. Database Systems, vol. 27, no. 3, pp. 229-260, 2002.
[8] F. Chiang and R.J. Miller, "Discovering Data Quality Rules,"VLDB Endowment, vol. 1, no. 1, pp. 1166-1177, 2008.
[9] G. Cormode, L. Golab, K. Flip, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the Confidence of Conditional Functional Dependencies," Proc. SIGKDD Int'l Conf., pp. 469-482, 2009.
[10] S.S. Cosmadakis, P.C. Kanellakis, and N. Spyratos, "Partition Semantics for Relations," Proc. Fourth ACM SIGACT-SIGMOD
Symp. Principles of Database Systems (PODS), pp. 261-275, 1985.
[11] W. Fan, F. Geerts, L.V.S. Lakshmanan, and M. Xiong, "Discovering Conditional Functional Dependencies," Proc. IEEE 25th Int'l Conf. Data Eng. (ICDE), pp. 1231-1234, 2009.
[12] P.A. Flach and I. Savnik, "Database Dependency Discovery: A Machine Learning Approach," Artificial Intelligence Comm., vol. 12, no. 3, pp. 139-160, 1999.
[13] C. Giannella and E. Robertson, "On Approximation Measures for Functional Dependencies," Information Systems, vol. 29, no. 6, pp. 483-507, 2004.
[14] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On Generating near-Optimal Tableaux for Conditional Functional Dependencies," Proc. Very Large Databases (VLDB) Conf., pp. 376-390, 2008.
[15] G. Gottlob and L. Libkin, "Investigations on Armstrong Relations, Dependency Inference, and Excluded Functional Dependencies," Acta Cybernetica, vol. 9, no. 4, pp. 395-402, 1990.
[16] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen, "Tane : An Efficient Algorithm for Discovering Functional and Approximate Dependencies," Computer J., vol. 42, no. 2, pp. 100-111, 1999.
[17] I.F. Ilyas, V. Mark, P. Haas, P. Brown, and A. Aboulnaga, "Cords: Automatic Discovery of Correlations Soft Functional Dependencies," Proc. SIGMOD Int'l Conf. Management of Data, 2004.
[18] M. Kantola, H. Mannila, K.-J. Ra¨iha¨, and H. Siirtola, "Discovering Functional and Inclusion Dependencies in Relational Databases,"
Int'l J. Intelligent Systems, vol. 7, no. 7, pp. 591-607, 1992.
[19] R.S. King and J. Oil, "Discovery of Functional and Approximate Functional Dependencies in Relational Databases," J. Applied Math. and Decision Sciences, vol. 7, no. 1, pp. 49-59, 2003.
[20] J. Kivinen and H. Mannila, "Approximate Dependency Inference From Relations," Proc. Fourth Int'l Conf. Database Theory (ICDT '92), pp. 86-98, 1992.
[21] A. Koeller and E.A. Rundensteiner, "Heuristic Strategies for Inclusion Dependency Discovery," On the Move to Meaningful Internet Systems 2004: Proc. Int'l Conf. CoopIS, DOA, and ODBASE, pp. 891-908, 2004.
[22] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient Discovery of Functional Dependencies and Armstrong Relations," Proc. Seventh Int'l Conf. Extending Database Technology (EDBT): Advances in Database Technology, vol. 1777, pp. 350-364, 2000.
[23] S. Lopes, J.-M. Petit, and L. Lakhal, "Functional and Approximate Dependency Mining: Database and Fca Points of View," J. Experimental and Theoretical Artificial Intelligence, vol. 14, no. 2, pp. 93-114, 2002.
[24] H. Mannila and K.-J. Ra¨iha¨, "Dependency Inference," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB), pp. 155-158, 1987.
[25] H. Mannila and K.-J. Ra¨iha¨, "On the Complexity of Inferring Functional Dependencies," Discrete Applied Math., vol. 40, pp. 237- 243, 1992.
[26] F. De Marchi, F. Flouvat, and J.-M. Petit, "Adaptive Strategies for Mining the Positive Border of Interesting Patterns: Application to Inclusion Dependencies in

Databases," Proc. Workshop Constraint- Based Mining and Inductive Databases, pp. 81-101, 2006.
[27] F. De marchi, S. Lopes, and J.-M. Petit, "Efficient Algorithms for Mining Inclusion Dependencies," Proc. Eighth Int'l Conf. Extending Database Technology (EDBT), pp. 199-214, 2002.
[28] F. De Marchi, S. Lopes, and J.-M. Petit, "Unary and N-Ary Inclusion Dependency Discovery in Relational Databases," J. Intelligent Information Systems, vol. 32, no. 1, pp. 53-73, 2009.
[29] F. De Marchi and J.-M. Petit, "Approximating a Set of Approximate Inclusion Dependencies," Advances in Soft Computing— Intelligent Information Processing and Web Mining, vol. 31, pp. 633- 640, 2005.

BIOGRAPHIES

Satyanarayana Mummana is working as an Asst. Professor in Avanthi Institute of Engineering & Technology, Visakhapatnam, Andhra Pradesh. He has received his Masters degree (MCA) from Gandhi Institute of Technology and Management (GITAM), Visakhapatnam and M.Tech (CSE) from Avanthi Institute of Engineering & Technology, Visakhapatnam. Andhra Pradesh. His research areas include Image Processing, Computer Networks, Data Mining, Distributed Systems, Cloud Computing.

I Ravi kiran Rompella completed My MCA in 2011 from university of Mysore and pursing my Mtech in Avanthi Institutions Narsipatnam vizag worked as a software engineer for 1 year on java and javascript and sql and my research areas include Data Mining, Distributed Systems, Cloud Computing.