University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

2002

A pre and post data warehouse cleaning technique.

Timothy Emenike. Ohanekwu
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

Ohanekwu, Timothy Emenike., "A pre and post data warehouse cleaning technique." (2002). *Electronic Theses and Dissertations*. 705.
https://scholar.uwindsor.ca/etd/705

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Pre and Post Data Warehouse Cleaning Technique

By

Timothy E. Ohanekwu

A Thesis
Submitted to the Faculty of Graduate Studies and Research through
the School of Computer Science in Partial Fulfillment of the Requirements
for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2002

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-75801-X

Canada

962153

Timothy E. Ohanekwu 2002
© *All Rights Reserved*

Abstract

A data warehousing system is a single data repository, which integrates already existing information from different data sources belonging to an enterprise over a long time period. One of the main tasks in building a data warehouse is to ensure that data drawn from several data sources contain no structural and semantic conflicts before being loaded into the data warehouse. Representing the same real world object in numerous ways is just one form of data disparity (dirt) to be resolved in a data warehouse. Data cleaning is a complex process, which uses multidisciplinary techniques to remove all the conflicts inherent in warehouse data.

This thesis proposes two data cleaning algorithms. The first algorithm, designed for initial data warehouse cleaning, uses the token keys composed from record fields for comparison of records. The second algorithm is designed to subsequently clean an existing data warehouse in a timely fashion. The algorithms achieve optimal cleaning correctness in a good time.

Key Words: Data Warehousing System, Data Cleaning, Pre-data Warehouse Cleaning Algorithm, Post-data Warehouse Cleaning Algorithm, "Dirty Fields", Data Disparity, Semantic and Structural Conflicts.

To My Dearly Beloved Wife, Oby

Acknowledgement

There are a number of people I would like to thank for their immense contributions to the completion of this work.

First, I would like to express my profound appreciation to my Supervisor, Dr. Christie Ezeife for her superlative supervision, comments, encouragements, motivation, guidance and supports throughout the time of the thesis work without which this thesis would not have been completed.

I would also like to thank my External Reader, Dr. Robert Schurko, my Internal Reader, Dr. Y.H. Tsin and the thesis committee Chair, Dr. A.C. Sodan for their time to read the thesis document and corrections. Thank you very much.

To this end, I want to express my ineffable gratitude to my principal sponsor, Mr. Ben Emeka Ohanekwu for all his financial and moral supports. Without him, this dream could not have come through. I give special thanks to my twin brother, Mr. Raymond Uchenna Ohanekwu who also contributed greatly to the actualization of this goal. My deep appreciation to Rev Emma Ohanekwu for all his prayers and moral supports as well as to my mum, Mrs. Virginia Ohanekwu for her prayers. I will not forget to thank my wife, Juliet Oby Ohanekwu for her prayers, encouragements and moral supports. I love you all.

Finally, I would like to thank all the members of the Varsity Christian Warriors, University of Windsor for their prayers and supports. God bless you all.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

1 Data Warehousing Background

A data warehouse is a consolidated database, which contains a huge integrated amount of data organized around major subject areas of an organization that span over a period of time to serve a historic purpose [Ma96]. W.H Inmon [In96] defines data warehouse as a collection of **integrated, subject-oriented, time-variant** and **nonvolatile** databases designed to aid decision support functions. Data warehouse data come from numerous data sources. These data sources are normally built to satisfy the day-to-day activities of an enterprise. Hence, the data contained in these operational data sources do not have time stamps (not historical), and are updateable (volatile), since they are designed around functions (not subject-oriented). In other words, a data warehouse is primarily built to integrate operational databases and other legacy systems over a long period of time for decision support and analytical data querying purposes [BS97, CD00, In96].

It is apparent from the above definitions of a data warehouse that the data contained in a data warehouse are drawn from different sources. The different data sources might have been implemented on different computer hardware and software platforms [Ez01]. For example, a branch of a bank in a given city, say Windsor, may have a number of units. One unit may be in charge of the checking accounts, which is mounted on an XML technology shown in Figure 1.1 and deployed on an IBM PC-based computer. Another unit, in charge of savings accounts may maintain its data in a relational data model-based source shown in Figure 1.2, which runs on an Apple Macintosh PC. The bank may yet have another unit, which takes care of the credit card businesses and may have its data stored in a flat file deployed on a Unix-controlled mini computer. Apart from the differences in hardware, data model and software, there are likely to be differences in designs and data formatting or representation. Figures 1.1 and 1.2 are two different data sources, which use different data management models to describe the same domain objects. Some pieces of information contained in such function-oriented data sources undergo frequent updates to reflect customers' activities. For example, the information about customers' occupations, addresses, and telephone numbers may most likely change from time to time, though not frequently.

However, such data as the customers' balances will definitely change from transaction to transaction.

```
<CheckingAccounts>
    <Customer>
        <Cid>1001</Cid>
        <Cname>S John </Cname>
        <Cbirth> 12-25-1970 </Cbirth>
        <Cphone>256-1234 </Cphone>
        <Caddress> 995 Sunset Avenue, n9b 3p4 </Caddress>
        <Coccupation> Student </Coccupation>
        <Csex> Male </Csex>
    </Customer>
    <Customer>
        <Cid> 1003</Cid>
        <Cname>A D Diana</Cname>
        <Cbirth>10-11-1972</Cbirth>
        <Cphone>566-5555</Cphone>
        <Caddress>Church Street. # 4, n8k 6t6</Caddress>
        <Coccupation>Politics</Coccupation>
        <Csex>Female</Csex>
    </Customer>

</CheckingAccounts>
```

Figure 1.1: An Instance of an XML Data Source

Id	Name	Birth	Phone	Address	Occupation	Sex
S001	John Smith O	25-Dec-70	5605678	Sunset # 995 N9B3P4	Student	M
S002	Tim E Ohans	01-Jan-75	2566416	ABCD St. 695 n9b 2t7	Researcher	M
S003	Colette Johnen	08-Aug-64	123-4567	600 XYZ apt 5a5 N7C4K4	Business	F

Figure 1.2: An Instance of a Relational Model Data Source

Generally, function-oriented data sources have a number of common characteristics [SF97], as follows: (1) they contain raw (atomic) and updateable (volatile) data, (2) they are inconsistent (not integrated) in the way they represent information (e.g., the data source in Figure 1.1 represents the gender of customers as a string, while that of Figure 1.2 represents the gender of the clients as a single character), (3) they are not designed for complex data analyses and reports (because they are not subject-oriented), (4) they do not store historic data that is needed for data analyses and decision support.

Consequently, such application-driven data sources can only adequately answer simple and one-dimensional queries, such as: (1) how much does a given customer have in his

or her checking accounts? (2) Which customers have balance greater than the overall average? (3) How many students maintain savings accounts?

If such one-dimensional queries are all that a profit-making enterprise needs in order to gain competitive advantage over rival organizations then it is not worthwhile building data warehouses [Ez01]. The fact however, is that one-dimensional queries (usually posed on single-point data sources) can no longer meet the needs of today's business, which requires inputs from many, but integrated data sources. For the bank in our example to break even in our highly competitive society, it needs multidimensional queries in order to know (1) when in a year customers withdraw huge amounts from their accounts, (2) the pattern and trend in customers' needs, (3) season-by-season analysis of customers' transactions, (4) when in a year does the credit card unit make the greatest profits, etc. Data warehousing is the only option since it has historical, integrated, subject-oriented, nonvolatile and summarized data of an enterprise [In96, Ki96a]. A data warehousing system therefore is the most appropriate option in situations where: (1) large amounts of heterogeneous data are created so rapidly that real-life query processing is impracticable, (2) processing and mining enterprise data is a resource-intensive job that is better carried out off-line in order not to interfere with the daily business operations, (3) there are frequently asked questions, which would be better stored in data warehouse for improved performance and efficiency.

Data warehousing is one of the fastest growing client-server applications [SF97], and its applicability is widespread, namely: (1) manufacturing for shipment ordering and customer support, (2) retail for user profiling and inventory management, (3) financial services for claims, credit card and risk analyses as well as fraud detection, (4) transportation for fleet management, (5) telecommunications for call analysis and fraud detection, (6) hospitals (healthcare) for patients' cases and outcomes analyses [CD97].

However, a data warehouse should deliver the *right information, in the right place, at the right time and at the right cost.* The extent to which a data warehouse meets the above objective depends exclusively on the **"cleanness of data it contains"**. It is widely believed among data cleaning researchers that the data from various data sources are **"dirty"** in nature, hence must be cleaned before loading them into the data warehouse. Data cleaning is a process that adopts statistical methods and other techniques to *eliminate variations in data contents*, as well as to *reduce data redundancy* aimed at *improving the overall data*

consistency [De97]. It is a computerized means of examining databases, detecting missing and incorrect data, and correcting errors [SL95]. Data cleaning is naturally difficult to automate, and it has been realized that the more "dirty" the source data are, the much harder it is to automate their cleaning with a predetermined set of tools [GF01a].

The rest of chapter one is organized as follows. Section 1.1 focuses on data warehouse architecture, while data source integration is the central theme of section 1.2. Types of integration and components of an integrated system are discussed in subsections 1.2.1 and 1.2.2 respectively, while numerous integration problems are described in subsection 1.2.3. A detailed discussion of data cleaning is done in section 1.3. The thesis motivation is given in section 1.4, while thesis problems and contributions are given in sections 1.5 and 1.6 respectively. Chapter 1 concludes with the outline of the rest of the thesis in section 1.7.

1.1 Data Warehouse Architecture

Figure 1.3 shows a simplified data warehouse architecture consisting of *external data sources*, *extraction, integration and transformation software*, *optional Operational Data Store (ODS)*, *Data warehouse*, *data warehouse metadata* and a number of *front-end tools*. Each of these components is described next.

The data in a data warehouse come from **external data** sources. The data contained in these sources could be structured (with metadata), semi-structured (self-describing) or unstructured (without metadata, e.g., flat files), and may be in different formats (though may be referring to the same real-life objects). For example, the Cid field in the XML-based data source of Figure 1.1 contains integer values. The same attribute (Id) in relational database source of Figure 1.2 is of alphanumeric type.

Data Extraction, Integration and Transformation Modules are a set of programs that extract data from the underlying data sources and integrate them into the desired format of the data warehouse. For instance, the Cid and the Id could be integrated to CustomerId of a data warehouse. Other roles performed by the integration and transformation modules are information filtering, data aggregation or summarization, data cleaning and merging. Example of data summarization include: (1) getting the total amount of money customers in savings accounts deposited in a given week, (2) deriving total monthly deposits from weekly deposits, (3) getting an annual deposit from monthly deposits. Summarization could be summing up

similar values along one or several business dimensions. Examples of integration and transformation programs are **wrappers** and **mediators** described in section 1.2.2 of this thesis.

Operational Data Store (ODS) is optional in the data warehouse. If available in a data warehouse architecture, ODS is used as a staging area for the data warehouse and the data from ODS undergo further transformation before loading into the warehouse. Unlike data in a data warehouse, data in ODS is (1) current and up-to-date, (2) at a detailed level (not summarized) and (3) volatile or dynamic. Like data warehouse data, ODS contains data that are subject-oriented and integrated.

Data Warehouse is a repository that contains huge, subject-oriented, integrated and historical data, which can be an off-the-shelf or special purpose database management system. It contains a reconciled, consolidated and materialized view of information residing in several data sources [CD00].

Metadata is a directory that describes data in a data warehouse to users of the data warehouse. It is data about data and provides information about the tables in the data warehouse, number of rows in each table, etc.

Front-end tools are application systems, such as OLAP, DSS (Decision Support System), EIS (Executive Information System), data mining tools, information delivery systems, etc., used to mine and query the data warehouse for business decision-making.

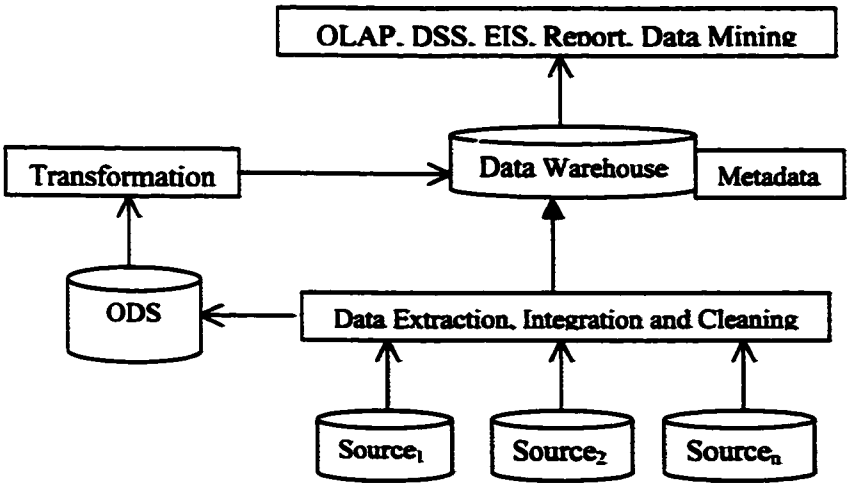

Figure 1.3: The basic architectural components of a data warehousing system

1.2 Database Integration

Database integration [TL99, ZH95] is a process that takes a number of databases as input and produces as output a single unified description of the input schemas and the associated mapping of information to support the integrated schema. Database integration is critical in the construction of data warehousing [AC93]. There are a number of reasons why an organization would want to integrate its data sources. These include: (1) Information sharing need - complex reasoning situations demand the solicitation of information from two or more different sources [Su94], (2) Intra- (or inter-) organizational information access facilitates the execution of functions, which may not be achieved by individual and isolated data sources [Gu89], (3) Many important scientific and engineering applications need input from multiple data sources in order to enhance productivity [BO86] and (4) It is an established fact that decision-makers more often than not need information from multiple sources.

Database integration process is a challenging task due to heterogeneities that characterize the underlying sources. Database heterogeneity within an organization stems from a number of reasons – (1) needs differ from one organizational unit to another, (2) different designers solve real-life problems differently, (3) different designers, most of the times, have differing perceptions of the same real-world entity and (4) operational environments and implementation platforms (hardware, software and models) differ across different departments of an enterprise. The numerous problems springing from data source heterogeneity are discussed in section 1.2.3.

1.2.1 Types of Integration

Four types of database integration discussed in the literature are – Virtual integration, materialized integration, schema integration and instance integration.

Virtual Integration – each of the participating systems retains its autonomy, but a global schema connects all the data sources. The global schema or view provides a representation of relevant data according to users' needs. With the global schema, users indirectly query the underlying data sources without having to know or understand their respective schemas. There is also an interface, which is a front-end tool that grants the users the flexibility of simultaneously retrieving data from the constituting data sources. This way,

the users of the system are freed from learning more than one query language, but are still permitted access to varied resources. One noticeable obstacle in virtual integration design is the high cost involved in accessing the dispersed sources. This form of integration is rarely used in data warehousing system.

Materialized Integration – this form of integration maintains a replicated view of the data at the sources [GM95, In96], and is a typical integration technique in use for information systems re-engineering and data warehousing [CG98]. One formidable obstacle to grapple with using materialized integration is the cost involved in view refreshment, because the views must be updated from time to time in response to changes at the base sources [CG98].

Schema Integration - involves merging of two or more database schemas into one integrated database schema [BL86, De89, LF97a]. The big task facing schema integration designers is to resolve the attendant problems arising from *naming, structural, key and constraint conflicts* [LT95, LT97].

Instance Integration - involves deciding that several instances from various sources refer to the same entity followed by merging [KS91]. The main instance integration task is how to reconcile partial or inconsistent data values from the matching instances [Le96].

Data warehousing system may involve some (if not all) of these types of integration depending on the nature of the data sources as well as the data warehouse design objectives.

1.2.2 Components of Data Integration System

Figure 1.4 shows a simplified, but representative architecture of a data integration system. The typical architecture of a data integration system is described in terms of two types of modules, namely, **wrappers** and **mediators** [Ul97]. Wrappers and mediators play such significant roles in data source integration that it is worthwhile to describe each in detail.

Wrappers are software modules that are able to access a data source and retrieve its data in a form that is coherent with the logical specification of the source. A wrapper (also called translator) is a software layer that translates the information from a specific data source into some standard form understood by the data integration system. For instance, a wrapper will convert the XML-based data model shown in Figure 1.1 to the relational data model of the data warehouse. The same "data model" conversion will be performed on any other data source whose data model is not originally relational. The indication is that every data source

that is not relational model will have a wrapper module for conversion. In other words, once the data source is screened through the wrapper layer, it would have assumed a data model close to (if not the same as) the data model of the target system. Thus, wrappers provide a common query language, which allows user queries to be converted into source-specific queries. Wrappers are usually equipped to detect changes in the sources and propagate the changes to the mediators. Wrappers play an enviable role in data integration. The application programmers are relieved the burden of having to redesign the entire system when the underlying data sources change, because only the wrapper module would be recoded if there is a change in the internal model of the constituting sources. This way, the layers above the wrappers will not be affected by the change. Thus, wrappers are designed to access a source, extract the relevant data, and present such data in a specified format and/or data model.

Mediators are software modules that receive as input sets of data from either wrappers or other mediators, refine the data by integrating and resolving conflicts, and produce another set of data as output, which correspond to the desirable result of a given query. The most pronounced task of mediators is to perform object fusion [PA97]. Object fusion entails grouping together information (from the same or different sources) about real-world entity. Fusing information from several sources involves "refinement" by way of removing redundancies, and resolving inconsistencies in data. For example, an attempt to integrate the XML-based source of Figure 1.1 and the relational model source of Figure 1.2 will need a mediator (or mediators) that would reconcile some of the inconsistencies noticeable in Figures 1.1 and 1.2. In a nutshell, a *mediator* plays the essential role of collecting, combining and **cleaning** data produced by different wrappers or mediators.

Mediators and wrappers are inevitable in an integration system due to the structural and semantic problems (described next) associated with database integration.

1.2.3 Data Integration Problems

Many problems arise when collapsing two or more data sources into a unified schema. Data integration problems could be schema integration problems [LG96], entity identification problems [WM89], record linkage problems [FS69], instance identification problems [WM89] or semantic integration problems [SB91]. In a data warehouse arena, the numerous data semantic and/or schematic conflicts encountered when merging two or more data sources are

referred to as *data cleaning* problems [HS98, Mo98] or *merge/purge* problems [HS95a, HS98].

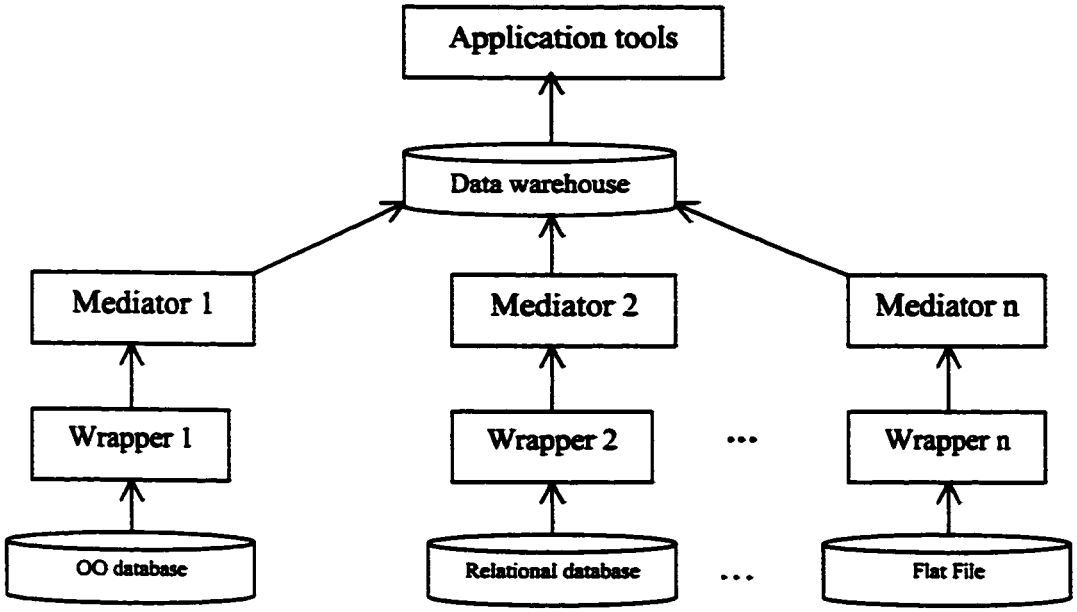

Figure 1.4: A Simplified Integration Architecture

Key: ⟶ information flow

One of the causes of integration problems is missing or partial information, which in turn makes the databases incomplete [Le96, TW84]. Another cause of integration problems is that **semantics of data are usually implicit and ambiguous in data sources**, and the integration complexity grows exponentially as the number of data sources increases [LF97a]. However, it is widely believed that all the semantic and non-semantic problems must be solved for a meaningful sharing of information across multitudes of data sources. The causes of integration problems are either due to semantic or structural incompatibilities.

Semantic incompatibility

This is the case when there is a disagreement about the meaning, interpretation or intended use of the same or related data. For example, there is semantic discordance between database DB_1 and database DB_2 when:

> DB_1 has a **30-character-name field** with the **given name first**, and DB_2 has a **20-character-name field** with the **family name first**, and the two are to be coalesced.

Devising a general solution to handle semantic conflicts is not usually feasible [Be91]. Different authors use different terms to express problems in semantics. For instance, there is *data value conflict* [LF97b], value conflict or data inconsistency [AK95], semantic discrepancy [SB91], or semantic discordance [Ha00] when:

> Two data sources have attributes A_1 and A_2 respectively, which refer to the same property, say, Date of birth of a particular person, but with different values in them, say "*10-10-75*" in A_1 and "*November, 19 1957*" in A_2

Data heterogeneity [BL86] or type conflict [De89, PSU98] exists between Record R_A in data source A and Record R_B in data source B when:

> *R_A and R_B refer to the same real-life object, but R_A is of type entity in database A and R_B is of type relationship in database B, or R_A is an attribute in A but R_B is an entity in B.*

There is also data heterogeneity between databases A and B if:

> *There is an attribute, say, Event-Dates in A and an attribute, say, Activity-Dates in B, which are meant to serve the same purpose but Event-Dates attribute is declared as an integer, e.g., 120199, to mean same as Dec-01-1999 contained in Activity-Dates of B, which is of type date.*

There is therefore data heterogeneity between the "Cid" fields of the XML-based source of Figure 1.1 and the "Id" field of the relational model source of Figure 1.2, since the former contains integer values, while the latter contains alphanumeric values.

A process that identifies and resolves the semantic discrepancy is called *semantic reconciliation* [SB91]. Semantic reconciliation means the same as data cleaning or scrubbing in data warehousing arena. All the aforementioned types of conflicts could be caused by: (1) **Name differences** [Ch98], i.e., Synonyms (e.g., 99% in one data source and "A+" in another) and homonyms (e.g., "John Smith" in data source A and "John Smith" in source B may be referring to different persons), (2) **Abbreviations** (e.g. "ACM" versus "Association of Computing Machinery"), (3) **Different scales or units of measurements** (e.g., one database might indicate weights in "kilograms" (Kg), while another may indicate the same in "tons", "Fahrenheit" may be a unit of temperature measurement in one source, while "Celsius" may be used in another), (4) **Recording (or typo) errors** (misspellings including transpositions) – e.g., "Distributed" might be misspelled as "Distribted", "66031" instead of "66013" is an example of transposition, (5) **Incomplete or missing information** - Null, Unknown, Not Applicable, Not available entries in some columns of various databases (e.g., City = null), (6) **Mis-fielding** (e.g., the value of "Canada" in a "province" field), (7)

Noise or Contradicting data – values outside the accepted range of a given attribute domain (e.g., "31st September 19999" in a date field, "-$3500.50¢" in a salary field), (8) **Differing representations** (e.g., "M" for "male" gender in one data source, "F" for female gender in another source, "0" for "male" in another, "1" for female in yet another source), (9) **Format differences, mismatched attribute domain**, e.g., Date could be formatted as "DD/MM/YY" in one data source and as "MM/DD/YY" in another or even as "YY/DD/MM" yet in another, (10) **Dummy values**, e.g., "999" in the age field or "$999,999.99" in the monthly salary field of an employee, (11) **Multipurpose Fields** - Data field may be used for multiple purposes which have varying meanings across different units of an organization, (12) **Inconsistent value naming conventions** – arising from different field entry format and use of abbreviations or acronyms (e.g., "pages" versus "pp"), (13) **Type mismatch**, e.g., Customer Id# declared as an "string" in the checking accounts data source, but as an "integer" in the savings accounts database, (14) **Embedded values** – Multiple values found in a single (free-form) field, e.g., "Ohanekwu Tim, WODD Systems Administrator" in a "job description" field, (15) **Values spread across fields** – this is common when two slots (Addr1 and Addr2) are provided for addresses, e.g., Addr1: 695 Randolph", Addr2: "Avenue, Windsor", (16) **Scanning errors**, e.g., "I" for "1" and vice versa, (17) **Keypunch errors**, e.g., "O" for "P", "R" for "T".

Structural incompatibility

Structural incompatibility on the other hand arises when attributes are defined differently in different databases. For example, a situation where entity or attribute names may be labeled differently among different databases, e.g. "**Company**" versus "**Comp-name**", has to do with structural disparity, and is appropriately termed "*schema-level integration problem*" in [BL86, LA86]. There are a lot of structural heterogeneities in Figures 1.1 and 1.2 since almost all their attributes are labeled differently (e.g., "**Name**" in Figure 1.2 versus "**Cname**" in Figure 1.1).

Most of the data cleaning problems are due to semantic-related discrepancies. Structural heterogeneity belongs to a different area of database research, and therefore will not be discussed further in this thesis.

1.3 Data Cleaning

1.3.1 Definition, Objectives and Significance of Data Cleaning

P*re-data warehouse data are dirty*, and must be cleaned if the data warehouse is to be trusted. Most (if not all) of the conflicts enumerated above are responsible for "dirtiness" in pre-data warehouse data. From the data warehouse standpoint, "995 Sunset Avenue, n9b 3p4" contained in "**Caddress**" field of Figure 1.1 and "Sunset # 995 N9B3P4" contained in "**Address**" field of Figure 1.2 are dirty. Converting one to the other in order to achieve consistency is a form of **data cleaning**. Using different identifying data formats to refer to potentially the same customers in different sources is subtle dirt in Figures 1.1 and 1.2. Data cleaning consists of removing inconsistencies and errors from source data sets [GF01a]. Data cleaning determines whether two or more records referring to the same real-life entity, but represented differently are the same followed by any (or combination) of the following activities: (1) collapsing them to get a consolidated whole, (2) unifying them to enhance joining operations and (3) retaining only one copy of the duplicates. By unification, it is meant generating one representative identity number for all the tuples that refer to the same entity. Example, "122570JOS" may be a unifying Warehouse Id for record one in Figure 1.1 and record one in Figure 1.2. Data cleaning is a process that identifies and corrects incomplete and incorrect information in databases [SL95].

The central objective of cleaning data is to realize consistent and correct data [LT00], and *is more than simply updating a record with good data.* A serious data cleaning involves **decomposing and reassembling data** [Ki96b], and may involve **semantic enrichment** [PS98], which is a process of acquiring additional information *(*from external source file e.g. companies' registry, birth registry, etc*)* in order to fix some ambiguities. It is also a complex process that uses different techniques to solve the diverse data cleaning problems. Data cleaning aspect of data warehousing project is so important that its omission is like "*building a new and costly car and putting a 10-year old engine without testing it"* [Ma99]. It is also likened to **a pollution treatment plant downstream on a river to clean up toxins dumped upstream** [Ne98]. The danger of omitting data cleaning aspect in data warehousing project is highlighted with an example.

Example: An information-based company built a data warehouse with the XML-based system of Figure 1.1 and the relational model system of Figure 1.2 constituting the underlying data sources. The data from the sources were loaded into the data warehouse without cleaning. The data warehousing system was queried as follows:

"Get all the customers who maintain both savings and checking accounts"

Obviously, this query will fail to identify customer with Id "1001" in Figure 1.1 as being the same as the customer with Id "S001" in Figure 1.2. This kind of false result will definitely lead to wrong analysis, which in turn leads to wrong decisions, such as incorrect customer behavior analysis. Similarly, it would be prohibitively costly allowing a typographical error of "$10050" instead of "$100.50" in the "deposit" field of a customer. Another consequence of omitting data cleaning is waste of memory resources in that many copies (duplicates) of the same entity (represented differently) are stored, thereby adding to the response time of the system. **The overall cost of allowing "dirt" into data warehouse is the emergence of a gigantic but worthless artifact that cannot be trusted, and which has little or zero utilization.**

1.3.2 Data Cleaning Tasks

Three main types of data cleaning tasks are identifiable.

(1) **Duplicate detection and elimination.** This is when a number of records have been identified as referring to the same entity. The task is to eliminate all except one.

(2) **Duplicate detection and merging.** This is the case when a number of matching records are seen as a partial source of information. The main task is to combine pieces of information from each matching records to obtain one record with complete information.

(3) **Entity Unification.** This is needed in situations where two or more records referring to the real-life entity are identified with different unique keys. The task is to generate a common identification code for the matching records.

The present work is concerned with tasks (1) and (3), which is described fully in chapter 3.

1.3.3 Data Cleaning Phases

Data cleaning is a many-phased process. Each phase involves a definite task. Six data cleaning phases are recognized in [Ki96b], namely, elementizing, standardizing, verifying, matching, householding and documenting.

The data to be cleaned are parsed or tokenized during the **Elementizing phase**, while the **Standardizing stage** is concerned with standardizing some of the tokens, e.g., "Dr." may be standardized to "Doctor" (if the domain is medicine), "Str" to "Street". The **Verifying step** attempts to verify the consistency of the standardized tokens from Standardizing stage, while the **Matching phase** consists of finding other records that are identical to a given token or group of tokens. The **Householding stage** focuses on grouping elements belonging to the same household. More often than not, other information (internal and/or external) sources are consulted to ascertain whether two or more elements belong to the same household. The **Documentation step** involves documenting the results of the previous phases in a metadata mainly for future cleaning exercise.

Similar data integration phases are described in [BL86, LT95] as follows - *preintegration, comparison, conformation, merging and restructuring*. In **Pre-integration or investigation phase,** the source schemas are inspected to decide on the integration policy. The **comparison phase** comprises further schema analyses and comparisons to determine the correspondence among concepts and possibly spot out schema conflicts. It is also called *schema-conditioning phase*. The main task in **conformation phase** is to resolve any conflicts arising from schema (as well as other) discrepancies. This phase can be further split into more steps as the development environments demand. In the **merging and restructuring phase,** schemas (also records) are superimposed. The superimposition of schemas or records may lead to an intermediate schema or record, which may still go through a number of fine-tuning before the desired quality is arrived at.

Comment: it should be pointed out that the phases described above should not be **hard-and-fast**. They should serve as guides other than rules, because the *diversity and unpredictable natures of data cleaning problems demands flexibility rather than hard-and-fast rules.*

1.3.4 Where to Apply Data Cleaning

There are three places where data cleaning techniques can be applied, namely, at the **application level**, at the **integration level**, and right inside the **data warehouse**.

The proponents of Data Cleaning at the **application level** believe that once data is clean at the operational or legacy level, they need not be cleaned elsewhere. At first glance, the argument above sounds valid, but a careful analysis will refute the idea altogether, because, even though there is nothing wrong in cleaning data at the operational level, but the question is whether it is worth doing, since operational data are needed for day-to-day activities only. Besides, some disparities in application data may not be discovered until there is need to merge them with data from other sources. The truth is that data cleaning elsewhere is unavoidable no matter how clean the operational data are.

Data cleaning at the **integration level** entails passing data from numerous sources through the data cleaning layer (i.e., screened or scrutinized) aimed at fixing any anomalies and inconsistencies inherent in the data sources. The data that emerges is cohesive and consistent to be loaded to the warehouse. The bulk of data cleaning task is performed at this level, and this level seems to be the best place to clean data.

Data cleaning **inside the data warehouse** is unavoidable since data therein are collected over a spectrum of time (usually between five and ten years); hence the data changes over time (or even become obsolete), and therefore must be cleaned occasionally.

1.4 Thesis Motivation

The direct proportionate dependency of runs on the number of fields in the algorithm described in [HS95b, HS98] was blamed on the extraction of keys from "dirty" fields. The penalty is the use of a time consuming transitive closure to combine results from different runs. [LH99] proposed an algorithm that uses external data sources (such as birth registry, patients data, company registry, etc.) to preprocess the "dirty" fields before extracting tokens from them. The use of external sources is not feasible in many cases based on a number of reasons: (1) Information contained in some external sources is so confidential that their accessibility is outlawed. (2) Even when some external sources are available, their accessibility is via a network and accessing the external source as many times as there are number of rows in the dirty dataset could be prohibitively slow and costly. (3) The

fundamental requirement for their algorithm that the primary key of the dirty dataset must agree with the primary key of the external sources is a serious setback considering the fact that most data sources exist independent of others.

1.5 Thesis Problems

In this work, we are interested in providing a better solution to some well-known problems in data cleaning arena. Three most prominent problems are:

(1) Given a large dataset of records containing various types of "dirt", how to achieve an optimal cleaning correctness (high recall) in a good response time.

(2) Given a large dataset of records containing various types of "dirt", finding a technique whose performance behavior depends less on threshold.

(3) Given an already cleaned data warehouse, how to carry out subsequent data cleaning in a less costly manner (in terms of time, computation and resources).

1.6 Thesis Contributions

The present work has contributed in a number of ways:

(1) Using both "exact" and "similarity" match techniques on the short lengthened token keys for record comparison significantly improves the recall ratio and greatly reduces the false positive matches.

(2) Reducing the number of runs on the dataset to a constant of 2 irrespective of the number of fields used for cleaning. This is better than proportionally varying the number of runs according to the number of fields used, as is the case in [HS95a, HS95b, HS98].

(3) Provision of a subsequent data cleaning solution in a timely and less costly fashion.

1.7 Outline of Thesis

The rest of the thesis proposal is organized as follows. A comprehensive description of existing data cleaning algorithms is given in chapter 2. The proposed algorithms are presented in chapter 3, while chapter 4 focuses on data cleaning performance issues. The thesis concludes in chapter 5 where some conclusions were reached.

CHAPTER 2
Previous and Related Work

In this chapter, previous and related work are described and discussed. The description will be presented in two broad sections. Section 2.1 covers the related cleaning algorithms before the data warehouse era, while section 2.2 covers the related algorithms that emerged after the data warehousing technology.

2.1 Data Cleaning Algorithms Prior to the Data Warehousing Era

The task of using matching techniques to decide if records match has been recognized as a research problem in both medical and business arenas since the 1950s. Most of the earliest medical research appeared under the name "record linkage" and were mostly concerned with identifying medical histories for the same person in different databases [NK59, NK62]. Both works agree that combining pieces of information from different sources to determine a complete whole is unavoidable when the main identifying attribute (e.g. Social Security Number) is missing or wrongly represented. The first work in business circle to detect duplicates of records appeared in 1973 [YG73]. The work by Senator et al. [SG95] highlights that record matching is relevant for fraud detection and money laundering. Two earliest algorithms are described in more detail below.

2.1.1 Duplicates Elimination in Large Data Files

Bitton et al. [BD83] originated the idea of sorting on some designated fields to bring potentially identical records together in a large data file. Two algorithms were described in their work. The first algorithm uses two way external merge sort, while the second variant uses one-way external merge sort to achieve the bringing of potentially identical records to close proximity. The first algorithm proceeds in two stages:

Stage 1: Sort the entire file on some (or part) of chosen fields
Stage 2: Sequentially scan the sorted file and compare a given record with any other record in the file. The matching technique used in this stage was not disclosed, but **exact matching** and **similarity matching** techniques are the two main options.

The second (more advanced) algorithm improves the first by gradually eliminating the duplicates as sorting proceeds. By tying both operations together, the number of steps involved, is reduced, hence the execution time, though at the expense of simplicity. The algorithm compares two input tuples and only one record is written to the output run if they are identical, while the duplicated copy is discarded *by simply advancing the appropriate pointer to the next record.*

Discussion: The algorithms described in [BD83] are precursory to most of the data cleaning work that emerged subsequently. However, two serious demerits reduced their desirability. The first downside is that the fields on which sorting is based may themselves be inconsistent, hence will fail to bring identical records together. Another weak point is that each record is compared with any other record in the dataset, a time consuming process that requires $N^2 - N * 1/2$ record comparisons (N being the number of records in the dataset).

2.1.2 Algorithm For Structural and Semantic Incompatibilities in Databases

A probability-based algorithm that solves inconsistencies arising from structural and semantic incompatibilities in databases is described in [CS91]. One strong characteristic of this work is that it can match records from several databases even when their primary keys are structurally and semantically incompatible. Specifically, this algorithm handles two problems:

Synonym problem: Two or more databases using different primary keys to identify the same real-life entity.

Homonym problem: Two or more databases identifying different real-life entities with similar identifiers.

In both cases, matching records on their primary keys alone can produce false results. This algorithm reduced such matching errors by involving other fields of the records being compared. The solution to **synonym problem** is hinged on the following reasoning:

> *IF two or more records describe the same real world entity, most of their other common attributes would agree even when their primary key attributes do not.*

Homonym problem is handled with a reasoning of the following nature:

> *IF two records agree in their primary key attributes, other common attributes might differ (significantly) if the instances do not refer to the same object or entity.*

The probabilistic algorithm solves the semantic and structural incompatibilities in three main steps. First, the useful attributes common to the data sources are determined. The values in the selected attributes are compared in the second step. The comparison values are estimated using the Bayesian probability model in the final step. Each step is briefly described next.

Step 1: Determination of common attributes: Given two relation schemas, say R and S, the attributes common to both R and S are determined by taking schematic intersection, ($R \cap S$) of R and S. For example, given two relational schemas – **CUSTOMER** {Cust#, Lname, Fname, Address, City, State, Zip, Total-Purchase-year-to-date, Date-last-purchased} and **CREDIT** {SSN, Lname, Fname, Address, City, State, Zip, Credit-Rating}, which are intended to be merged in order to create a list of customers with good credit rating, the useful common attributes are determined from **CUSTOMER** $\cap$ **CREDIT** = {Lname, Fname, Address, City, State, Zip.

Step 2: Entity-Join (E-join) Comparison: The values of each pair of common attributes are compared for a match using the entity join. The result, M of the E-join of two relations r(R) (with tuples r_k, k = 1, ... , i) and r(S) (with tuples, s_k, k = 1, ... , j) on the join attribute, $a_j \in R, S$ is expressed as:

$M = \mathbf{r(R)} \bowtie \mathbf{r(S) = r(R) \times r(S)}$, **such that** $\mathbf{r_i[a_j] \equiv s_j[a_j]}$ (i.e., the similarly named attributes in r(R) and r(S)), where $\equiv$ indicates equivalence. An instance of M from $R \cap S$ and joined on {Lname, Fname, Address, City, State, Zip} attributes may look thus:

R = {-, Smith, John, 51st street, New York, NY, 10006, -, -}

S = {-, Smith, Jorn, 51st street, New York, NY, 10006, -}.

The non-common attributes are not relevant and therefore were replaced with "-".

Step 3: Comparison value estimation: The outcome of the E-join comparison is used in the comparison value estimation. A comparison value is assigned depending on the number of matches between the pair of attributes. A value of 1 shows a match, while 0 means no match. That is, given two database schemas, R and S with useful common attributes $\{a_j, j = 1, ..., n\} \subseteq \{R \cap S\}$. Let $t = \{r_i, s_j\}$; $ri \in r(R)$, $s_j \in r(S)$. Note that $\{r_i, s_j\}$ is a pair of tuples belonging to $\{R \cap S\}$, where r_i is a tuple from R and s_j is a tuple from S. A vector $\gamma(t) = \{\gamma_1(t), \gamma_2(t), ..., \gamma_n(t)\}$ is

defined to hold the number of common attributes between R and S. For example, comparing R and S yields strings of 1s and 0s as follows:

Smith	John	51st street	New York	NY	10006
Smith	Jorn	51st street	New York	NY	10006
1	0	1	1	1	1

Therefore, $\gamma(R, S) = [1,0,1,1,1,1]$.

A tuple probability, Ptuple(t) is finally used to determine the correctness or otherwise of the record comparison. Ptuple(t), is a number between 0 and 1 and is a conditional probability that join attributes are matched correctly or otherwise. Intuitively, the tuple probability on [1,0,1,1,1,1] will be close to 1, meaning that the two records taken from two different sources may be actually referring to the same person.

Discussion: The probabilistic algorithm described above uses "exact string matching" technique, hence can only match two strings when they agree character-to-character. The implication is that it cannot solve inconsistencies arising from the use of "abbreviations or acronyms", e.g., "Dept" versus "Department". The use of the Bayesian probability theorem for final decision is also questionable and amounts to waste of effort and time as the strings of 1s and 0s that emerge after attribute comparisons are sufficient to decide whether the records are the same or not.

2.2 Data Cleaning Algorithms in Data Warehouse Era

2.2.1 Sorted Neighborhood Method (SNM)

SNM described in [HS95a, HS98] tackles the data cleaning (merge/purge) problems that arise when data are drawn from two or more sources. How SNM solves the merge/purge problem is summarized as follows: (1) Pool all the data sources into a single list of N size, (2) Create sort keys from some chosen fields of the dataset, (3) Sort the entire dataset based on the sort keys, (4) Partition the dataset such that all the records having exact keys are clustered together, (5) Compare records in each cluster and merge records if they are the same. Each phase is briefly described below. The two variants of SNM are described as well.

Key-extraction phase: This phase requires O(N) operations (N being the size of the dataset). The keys are extracted from the designated fields or portions of fields. For example, given the following table:

Rec. No.	First-Name	Last-Name	Address	SSN
1	Sal	Stolfo	123 First Street	45678986
2	Sal	Stiles	123 Forest Street	45654321
3	Sal	Stolpho	123 First Street	45688987
4	Sal	Stolfo	123 First Street	45678987

The keys are obtained by concatenating the first three consonants of a last name with the first three letters of the first name, followed by the address number and all of the consonants of the street name, followed by the first three digits of the SSN, resulting in the following table.

Rec. No.	Key
1	STLSAL123FRST456
2	STLSAL123FRST456
3	STLSAL123FRST456
4	STLSAL123FRST456

Sorting Phase: The records in the dataset are sorted on the "keys" so that potentially identical records should be brought to close proximity. The time complexity for this phase is O(N log N).

Comparison and merging phase: A window of fixed size, w, is moved through the sequential list of records thereby limiting records comparisons to the records in the window. That is, each time a new record enters the window; the new entrant is compared with the previous w – 1 record to find a match. The first record in the window slips out of the window. The comparison of records during this stage to determine their equivalence is a complex inferential process that takes into consideration more information from the records being considered than the keys used to bring them close together. The equational theory is employed to dictate the logic of domain (not just value or string) equivalence. Rule-based model is used to determine if records close together are the same. A typical rule-based system is of the following nature:

Given two records r_1 and r_2,

IF r_1.Lastname = r_2.Lastname AND their first names differ slightly AND r_1.Address = r_2.Address THEN

r_1 is equivalent to r_2

This phase has a time complexity of O(wN), where w is the size of the window and N is the number of records to be compared.

The Single-pass and Multi-pass variants of SNM

The **single-pass** variant entails applying SNM once on the dataset. Its effectiveness largely depends on how powerful the first portion of the key is to discriminate one record from the other. The **Multi-pass** is the alternative approach in which several independent runs of the SNM are applied on the dataset with the head portion of the keys varied at each run. The argument in favor of this scheme is that no single key is adequate enough to bring all duplicates together. One run may use the student's contact address as the head of the key, the next run may be based on key in which the head comes from the student's ID#, and so on. Each run would yield a set of pairs of records that could be merged using transitivity rule of the form:

Transitivity Rule
If record X was identified as record Y's equivalence in one of the independent passes, **And** record Y was identified as an equivalence of record Z in another pass, **Then** record X is equivalent to record Z

Discussion: The combination of clustering and window scanning techniques is one of the enviable features of SNM. Clustering and window-restricted comparison obviously limits the number of comparison to items in the cluster and window. The idea of sorting on keys extracted from the chosen fields is also a plus to this algorithm. However, a number of demerits are identifiable. First, the fields from where the keys are extracted are dirty themselves, implying that the keys obtained from them are also dirty. The direct consequence of this is that the single-pass variant would not catch all the identical records. For example, date of birth written as "12976" in database A, but appeared as "91276" in database B might not be brought to a close neighborhood if date of birth is the head portion of the key. The multi-pass variant is too complex in principle and time. "Dusting" the fields before extracting keys will definitely reduce the number of passes needed to capture all the identical records in a large dataset.

2.2.2 Duplicate Elimination Sorted-Neighborhood Method (DE-SNM)

DE-SNM presented in [HS95b] improves SNM described previously. Its design is hinged on the observation that most of the duplicate records detected by the scanning window always

have equivalent sort keys, hence it is more efficient to find the matching records during the sorting phase, and retaining only one member (called the prime representative) for comparison. Another improvement is the splitting of the sorted dataset into two lists – duplicate list and no-duplicate list. The duplicate list contains all the records with exact duplicate keys, while no-duplicate list contains all the other records. The duplicate list is screened through a small window, which may result in further splitting of the original list into two smaller lists - the list of matched records and a list of unmatched records. The list containing the unmatched records is merged with the original no-duplicate list and a second window scan is performed. Figure 2.1 summarizes the DE-SNM operations. DE-SNM also has single-pass and multi-pass versions.

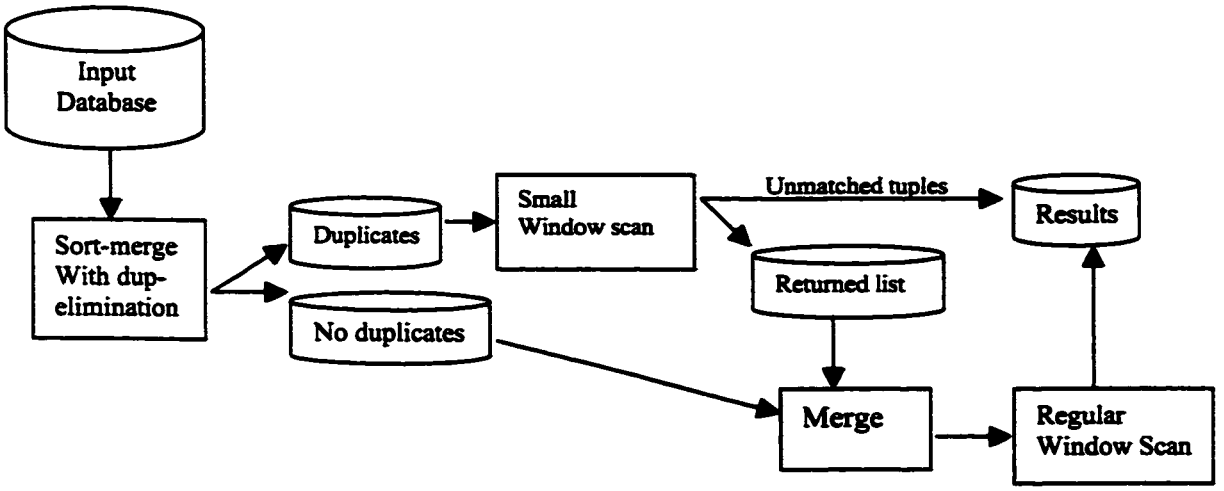

Figure 2.1: Duplicate Elimination Sorted Neighborhood Method

Discussion: DE-SNM inherits most (if not all) of the downsides of the SNM since the keys upon which sorting is based are extracted from fields, which are not preprocessed.

2.2.3 Data Value Conversion Rules

The set of rules described in [HW97] is designed to tackle context dependent conflicts, which are forms of data value conflicts due to systemic disparities stemming mainly from conflicting assumptions or interpretations in different data sources. In other words, these rules can be used to describe quantitative relationships among the attribute values from numerous data sources. Specifically, data value conversion rules remedy a situation where two attributes A_1 and A_2, which characterize the same property of an object in two data sources D_1 and D_2, but the content c_1 in A_1 differs from the content c_1 in A_2. Besides, the rules can equally handle aggregate

conflicts, which arise when an aggregate is used in one data source to indicate a set of values in another.

Generally, these rules adopt the datalog notation, thus:

Head ⇐= Body

The *head* predicate represents a relation in one data source, while the *body* is a conjunction of a number of predicates, which can either be inbuilt arithmetic predicates or aggregate functions or extensional relations present in the underlying data sources. In some cases (when conflicts are caused by synonyms), lookup tables are created to be part of the conversion rules. An example below helps to clarify the description given above.

Example: integrating data sources having different representations

Two data sources S.student with schema (sid, sname, major) and S.employee with schema (eid, ename, salary) are to be integrated into S.student_employee with schema (id, name, major, salary). The above task can be achieved through the following conversion rule:

S.student_employee (id, name, major, salary) **⇐= S.student (id, name, major),**
S.employee (eid, name, salary), S.same_person (id, eid).

The relation S.same_person shows the correlation between the student data source and the employee database. Example of this is where a professor used grades (e.g. A+, A, A-, etc.) to express students' performances in one database DS_1 while another professor used raw scores (e.g. 100, 93, 89, etc.) to express students' performances in the second database DS_2. The following conversion rule is valid if the information contained in DS_1 and DS_2 are to be merged.

DS_1.student_grade (id, name, grade) **⇐= DS_2 (id, name, score), score_grade (score, grade).**

The *score_grade (score, grade)* in the above rule is a lookup table that defines the relationship between scores and grades thus:

Score_grade (>= 95, 'A+')
Score_grade (>= 87, 'A')
Score_grade (>=80, 'A-')

Score_grade (>= 36, 'F')

2.2.4 The Field Matching Algorithms

Two algorithms are described in [ME96] to tackle problems of deciding if two field values are semantic alternatives. These are "the basic field matching algorithm" and "a recursive field matching algorithm".

2.2.4.1 Basic Field Matching Algorithm

This algorithm proceeds in two steps: (a) The extraction and sorting of the atomic strings of each field and (b) The atomic strings' comparison – the number of matches is noted and used in computing the basic matching score, whose value indicates whether or not the fields being matched are the same. The fields are first conditioned by stripping them of the common or stop words, such as {for, the, of, on, &, -, /}. Two atomic strings are taken to be a match if they are the same string or if one is a prefix of the other. The example below is meant to exemplify the description of this matching scheme.

Example: Consider a field whose value is "Comput. Sci. & Eng. Dept. University of California, San Diego" and another field B with content "Department of Computer Science, Univ. Calif., San Diego".

The preliminary stage involves removal of stop words from A and B thus:

A (without stop words) – "Comput. Sci. Eng. Dept., University California, San Diego".

B (without stop words) – "Department Computer Science, Univ. Calif. San Diego".

The next step is to sort the emerging atomic strings as follows:

A (sorted) – California Comput. Dept. Diego Eng. San Sci. University

B (sorted) – Calif. Computer Department Diego San Science Univ.

There are 6 strings k {Comput. Sci, San, Diego, Univ. Calif.} in "A" that match some strings in "B". The overall matching score is computed using the following formula.

$$\textbf{The overall matching score} = \frac{k/(|A| + |B|)}{2}$$

$$= \frac{6/(8 + 7)}{2} = \{(6 \times 2)/ 15\} = 0.8$$

The high value from the above computation implies that A and B are likely the same entity.

Discussion: The basic field-matching algorithm does not take into consideration abbreviations that are not prefixes. Also, it has no solution to a scenario where one word, e.g., "ACM",

abbreviates two or more words, e.g., "Association of Computer Machinery". The greater part of its time complexity comes from the sorting phase, which requires O(n log n), where n is the number of atoms common to the comparable tokens.

2.2.4.2 Recursive Field Matching Algorithm

This algorithm uses the recursive structure of typical textual fields. A degree of 1.0 is assigned if one string matches the other exactly or if one abbreviates the other. A degree of 0.0 is assigned if there is no match. Four patterns are used to resolve or match abbreviations.

(a) The abbreviation is a prefix of its expansion, e.g. "Univ." abbreviates "University", (b) The abbreviation combines a prefix and a suffix of its expansion, for instance, "Dept." is a match for "Department", (c) The abbreviation is an acronym for its expansion, example, "UCSD" abbreviates "University of California, San Diego" and (d) The abbreviation is a concatenation of prefixes from its expansion; for instance, "Caltech" and "California Institute of Technology" are matches.

Discussion: The recursive field-matching algorithm has a quadratic time complexity, meaning that every sub-field in A must be compared with every sub-field in B. The worst case arises when each atomic string in A compares with each atomic string of B.

2.2.5 External Source-enabled Field-preprocessing Algorithm

The data cleaning algorithm described in [LH99] introduces the idea of pre-processing the fields upon which sorting would be based prior to the sorting phase. It can be said to be an improvement over the SNM and DE-SNM described earlier on. It is believed that the chances of bringing potentially identical records to close neighborhood are increased by first pre-processing the individual fields before sorting. The task of cleaning data is carried out in five steps, namely, (a) Scrubbing of dirty data fields, (b) Tokenizing and sorting data fields, (c) Sorting of records, (d) Comparison of records, and (e) Merging of matched records

Scrubbing of dirty data fields: The pre-processing task is accomplished with an (or some) external data source(s). How this is done is shown in Tables 2.1-1, 2.1-2 and 2.1-3.

SSNO	Name	Age	Sex
0273632T	Koh Yiak Heng	43	M
3635290Y	Tan Kah Seng	16	M
5927356K	Vivian Chua	25	F

Table 2.1-1: External Data Source

SSNO	Name	Age	Sex
0273632T	Koh Y.H	42	M

Table 2.1-2: "Dirty" Record in Database

SSNO	Name	Age	Sex
0273632T	Koh Yiak Heng	43	M

Table 2.1-3: "Cleaned" record in the database

The "dirty" record in the database is simply replaced with the corresponding record from the external source file, which can be a birth registry, credit card database, etc.

Tokenizing and sorting data fields: This involves splitting of field values into meaningful tokens. Example, consider three name fields taken from different sources, A with value "Liu Kok Hong", B with value "Liu K.H" and C with value "Yap Kooi Shan". Tokenizing each yields: **A:** {Liu, Kok, Hong}, **B:** {Liu, K, H}, **C:** {Yap, Kooi, Shan}. Sorting within field in ascending order yields: A: {Hong, Kok, Liu}, B: {H, K, Liu}, C: {Kooi, Shan, Yap}.

Sorting of records: The entire database is sorted on the sorted tokens of the pre-processed fields. Example, sorting on the name field of the preceding example yields:

B	H K Liu
A	Hong, Kok, Liu
C	Kooi, Shan, Yap

Comparison of records: Records are compared to determine if they match. A window of fixed size w is used to limit comparison to only records in the window. The concept of *field weightage* (a value assigned by the user that indicates the relative importance of a field to computing the degree of similarity between two records) is used to compute the *degree of similarity* between two records. The computation of records' degree of similarity is an essential task in this phase. The degree of similarity shows the extent to which two records match or otherwise. Different kinds of similarity matching are used. These are, *exact string matching, single-error matching, abbreviation matching* and *prefix substring matching.*

(a) **Exact string matching** – A value of 1 is returned if string A exactly matches string B; Otherwise 0 is returned, (b) **Single-error matching** – The matching task includes: checking for additional characters, missing characters, substituted characters and transposition of adjacent characters (c) **Abbreviation matching** – An external source file containing the standard abbreviation of words may be consulted. Token A could be considered a likely abbreviation of token B if and only if either or both of the following conditions hold: *(i) A are encapsulated by B (ii) all the characters in A appear in the same order as in B.* The result of abbreviation matching is 1 only when one token is found to be an acronym of the other, (d) **Prefix substring matching** – This is concerned with finding two similar tokens where one is a leading substring of the other. The result is 1 *if all the characters in one are found in the other.* Example, a prefix substring matching between "Tech" and "Technology" (DoS_{Tech}) returns a degree similarity of 1, since all the characters in "Tech" are found in "Technology". However, $DoS_{Technology} = 0.4$, considering that there are 6 characters in "Technology" that cannot be found in "Tech." Generally, if the length of the first string is x, then 1/x is subtracted from 1, (i.e. the maximum degree of similarity) for each character not found in the second. For instance, the Degree of Similarity (DoS) of token "cat" relative to token "late" is:

$DoS_{cat} = 1 - 1/3 = 0.67$ (1/3 is as a result of character "c" in *cat* that is not found in *late*). Conversely, DoS of token "late" relative to token "cat" is: $DoS_{late} = 1 - (1/3 + 1/3) = 1 - 2/3 = 0.33$. (2/3 indicates that two characters – "l" and "e" in "late" are not found in "cat").

The results of the above series of comparisons are used to compute the similarity between the entire fields – called *field similarity*. Fields similarities and the assigned fields weightage are in turn used to calculate the *record similarity*. The formula used to compute the field similarities is given in proposition 1, while that for record similarity is expressed in proposition 2.

Proposition 1: Formula for Field Similarities

Given a field in Record X with tokens $t_{x1}, \ldots, t_{xn}$ and a corresponding field in record Y with tokens $t_{y1}, \ldots, t_{ym}$. Token t_{x1} is compared to the corresponding token t_{y1}. If $DoS_{x1}, \ldots, DoS_{xn}$, $DoS_{y1}, \ldots, DoS_{ym}$, are the maximum degree of similarities for tokens $t_{x1}, \ldots, t_{xn}$ and ty1, ..., t_{ym} respectively, then the field similarity of X and Y denoted by Sim_F (X, Y) is mathematically expressed as follows:

$$Sim_F(X, Y) = (\Sigma^{n}_{i=1} t_{xi} + \Sigma^{m}_{i=1} t_{yi}) / (n + m)$$

Proposition 1 is explained with the following example. Consider two records X and Y. X has a name field with the following tokens (sorted), E T Ohanekwy, while Y has a name field with the following tokens (sorted), Emenike Timothy Ohanekwu. Each token from X is compared with corresponding token from Y, and the degrees of similarities (DoS) are presented in the following table.

$X: x_1$	$Y: y_1$	DoS_{x1}	DoS_{y1}
E	Emenike	1.0	1.0 – (6/7) = 0.14
$X: x_2$	$Y: y_2$	DoS_{x2}	DoS_{y2}
Ohanekwy	Ohanekwu	1.0 – (1/8) = 0.875	1.0 – (1/8) = 0.875
$X: x_3$	$Y: y_3$	DoS_{x3}	DoS_{y3}
T	Timothy	1.0	1.0 – (6/7) = 0.14

The *field similarity* of X and Y Sim_F (X, Y) from the above table are based on the formula given under proposition 1, i.e. $Sim_F (X, Y) = (\Sigma^n_{i=1} t_{xi} + \Sigma^m_{i=1} t_{yi}) / (n + m)$ becomes:

$$\frac{1.0 + 0.875 + 1.0 + 0.14 + 0.875 + 0.14}{3 + 3}$$

$$= \quad 4.03 / 6 = 0.672$$

Proposition 2: Formula for Record Similarity

Suppose a database has fields $F_1, F_2, \ldots, F_n$ with field weightages $W_1, W_2, \ldots, W_n$ respectively. Given records X and Y, let $Sim_{F1} (X, Y), \ldots, Sim_{Fn} (X, Y)$ be the field similarities computed from proposition 1. Then record similarity for X and Y is expressed as follows:

$$\Sigma^n_{i=1} Sim_{Fi} (X, Y) * Wi$$

Assuming that records X and Y in the example given under proposition 1 have address fields, which are used for cleaning as well. Let the *field weightages* assigned by the user to the name and address fields be 0.55 and 0.45 respectively. Assuming that the address *field similarity* of X and Y had been computed as 0.46. Then record similarity for X and Y, $\Sigma^n_{i=1} Sim_{Fi} (X, Y) * Wi$ is:

$$(0.672 * 0.55) + (0.46 * 0.45) \approx 0.58$$

Merging of records: The last step is to merge matching records. Two records are duplicates and hence merged if the record similarity exceeds a certain threshold, say 0.8. How the threshold is determined was not disclosed in [LH99].

Discussion: The pre-processing stage introduced in this algorithm is a big plus, though the use of external data source is not feasible in many instances. Also, leaving the decision at to whether records being compared match or not to the end of the time consuming and processor-intensive processes has a great time overhead. This algorithm could be improved in a number of ways: (1) incorporating pre-processing procedure into the algorithm would eliminate or minimize the need to consult external source files, (2) Decision as to whether two records do not match can be determined at much earlier stage of their comparison, (3) Reducing the length of strings in records being compared, etc.

2.2.6 IntelliClean: A Knowledge-based Intelligent Data Cleaner

Expert System technique is used in the data cleaning algorithm described in [LT00]. The algorithm achieves its data cleaning task in three stages, namely, *Pre-processing stage, Processing stage* and *Validation and Verification stage.*

Pre-processing stage – Conditioning and scrubbing of records are the main tasks of this stage. These include "data type checks", "format standardization" as well as fixing of inconsistencies raised by the use of abbreviations or different coding schemes (e.g., "M" for male versus "F" for female). Reference functions and look-up tables are used in this stage. The conditioned records from this stage are fed to the next stage.

Processing stage – This stage contains an Expert System (ES) engine, which has a set of rules, which form the Knowledge Base mounted on Java Expert System Shell (JESS) language. Generally, a rule is of an "**IF** *<condition>* **THEN** *<action>*" form. The knowledge base consists of a number of rules, which can be grouped as follows: (i) Duplicate Identification Rules – These are rules that contain conditions for identifying duplicate of records, (ii) Merge/Purge Rules – The rules in this category stipulate how to merge records found to be duplicates, (iii) Update Rules – These rules specify what events can lead to data updating, (iv) Alert Rules – Contain conditions that raise alarm when satisfied.

Any of the above rules could be fired opportunistically when the ES engine receives data from the first stage. High Certainty Factor (CF) is assigned to rules that have high tendency to identify true duplicates. Prior to merging operation, users assign Threshold value (TH) to groups of records, which indicate their confidence that there will be a merge. CF and TH are compared before any merge. Any merge that will result in a CF less than TH will be refused. Sorted Neighborhood Method is also used to bring potential duplicates together. The duplicate records that satisfy the conditions will fire the merge/purge rules.

Validation and Verification stage – Manual methods may be needed in a situation where the merge/purge rules fail to detect some duplicate cases. Wrong merges and incorrect updates could be reversed. All these could be aided by the information held in a *log report file*, which keeps track of all the actions of the system as well as what causes a given rule to fire. Another activity in this stage is the removal or adjustment of any rule discovered to consistently merge records wrongly or update data incorrectly.

2.2.7 An Adaptive Duplicate Detection Algorithm

An algorithm that aids in detecting approximately record duplicates is presented in [Mo97, Mo00]. The main contribution of this algorithm is the replacement of the fixed sized window of [HS95a, HS98] with a priority queue whose size varies (expands or shrinks) in response to the size and homogeneity of the discovered clusters in the database being scanned. The use of variable sized window provides a solution to the limitations of fixed sized window version, which are: (a) Insufficient number of comparisons made when there are more duplicate records than the size of the window, which leads to some duplicates passing undetected, and (b) Unnecessary comparisons are made when there are few or no duplicates.
The use of variable sized window significantly reduces (about 75%) the number of comparisons while competing favorably with the work in [HS95a, HS98] in terms of accuracy.

How this algorithm works is summarized as follows: the entire database is sorted and scanned with a priority queue of record subsets belonging to the last few clusters detected. The priority queue contains a fixed number of sets of records each of which contains one or more records from a detected cluster. A cluster of records would be saved in the priority queue only when such contributes to the variability of the cluster being represented. In trying to determine a match for a record, say R_k, the priority queue is scanned starting from the head (the cluster with

highest priority). The cluster to which R_k belongs is compared with each known cluster in the queue. There are three cases:

Case 1: Cluster-based comparisons are successful: In this case, R_k is found to be a member of a known cluster in the priority queue. The decision is to combine the incoming cluster of which R_k is a member with the matching cluster in the queue. The matching cluster becomes the highest priority cluster and therefore is moved to the head. The process continues with the next record, R_{k+1} in the sorted dataset.

Case 2: Cluster-based comparisons are unsuccessful: In this, the incoming cluster to which R_k belongs does not match any of the clusters in the priority queue, which leads to record-based comparison. Record-based comparison entails iterating through each cluster comparing each member of the cluster, R_j, with R_k. Union (R_j, R_k) operation is used to combine the clusters of R_j and R_k if R_k is found to match any record R_j. Besides, R_k may be included as a member of the cluster, which R_j is a member of. If the comparison between R_j and R_k did not return a high score then other members of that cluster are skipped while the next priority cluster is checked. The intuition is that if there is no match between R_j (which is the first record in a given cluster) and R_k then there is high probability of failure if comparison is done with the rest of the members (R_{i+n}, $n>=1$) of the cluster. The next member of a cluster is considered in an unsuccessful comparison only when the comparison score between R_j and R_k is very close to the matching threshold.

Case 3: Record based comparison is unsuccessful: This is the case when R_k is not found to match any record in all the clusters of a priority queue. The action is to create a one-member cluster for R_k, which is given the highest priority, as such becomes the head cluster. The cluster with the lowest priority is dropped from the priority queue if the creation of a new cluster increases the size of the queue beyond expectable limit.

2.2.8 The AJAX: the Extensible Data Cleaning System

The AJAX models the data cleaning logic as a directed graph of data transformations, which starts from some input source data and terminates by returning clean data. AJAX is

designed to accept as input a set of possibly erroneous and inconsistent data flows (e.g., files consisting of fixed-size records or relational tables) and return a set of consistent, error-free and formatted data flows. AJAX is designed to handle data cleaning problems in the context of a data warehouse construction and specifically in a telecommunication system domain. Another distinguished contribution of AJAX is the provision of a declarative and extensible language for specifying the desired transformations. The language specification comprised SQL statements enabled with a set of specific primitives. The declarativeness and extensibility of AJAX makes it easy to deploy and maintain. Another side of the system's extensibility is the involvement of the user during the data cleaning process. There are two points where the user is needed, namely, (a) exceptional situations that may arise during the data cleaning process, and (b) inspection of the intermediate results. AJAX incorporates a data lineage mechanism, which is used to explain the actions of the system. With the help of the lineage component, the user can backtrack within a data cleaning program execution in order to disclose the input records that generate a given output. The knowledge gained during system backtracking serves to refine the data cleaning criteria as well as to iterate the cleaning procedures. Metadata annotations dynamically generated during the running of the data cleaning system help to support the data lineage process. Therefore, AJAX is both *iterative and interactive*. AJAX uses a lot of optimization techniques, which are not discussed in this thesis, but which are fully described in [GF00, GF01a, GF01b].

2.2.9 The Potter's Wheel System

Another data cleaning framework called Potter's Wheel is presented in [RH00]. It integrates both transformation and discrepancy detection in a single interactive framework. This framework allows users to gradually build transformations by composing and debugging transforms in a stepwise fashion on a spreadsheet-like interface. It also allows the effect of any change or transformation by the user to be displayed immediately for either acceptance or rejection. It is not possible to describe all the data cleaning operations of this system. However, a selected few are described next.

Divide operation: this performs a conditional column division and is employed to solve problems where both first and last names are mapped unto the first name column of another. Figure 2.2 shows how the "divide operation" solves the problem of this nature. This operation simply separates the occurrence of both "the first and last names" on one column from where

each name occurs in a column. Adding another column for only records having their first and last names together does this.

Horizontal transforms: this is used to solve higher-order schematic heterogeneities, which arise when information is stored partly in data values, and partly in the schema. An example of a higher-order heterogeneity is where a student's grades are outlined in one row per course in one schema, and in multiple columns in another. A number of operations are employed in horizontal transformation. One operation is called "split", which splits value in one column into two or more columns. Another operation is called "fold", which converts a single row into multiple of rows. Figure 2.3 shows how both "split" and "fold" operations are used to solve a problem called a "high-order" disparity.

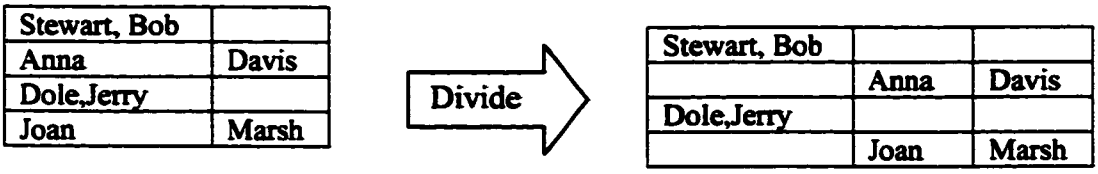

Figure 2.2: The divide operation of the Potter's Wheel system

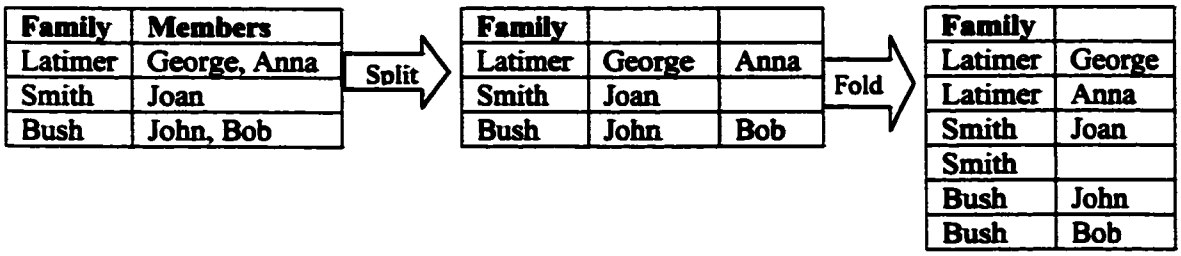

Figure 2.3: Fold and Split operations of Potter's Wheel System

The split arrow in Figure 2.3 is an operation that separates family members listed in a single column into individual members, where each member occupies a column. This increases the number of columns. The fold arrow reduces the number of columns, but increases the number of rows. It is an operation that create separate row for each family member occurring in a separate column.

CHAPTER 3

A Pre and Post Data Warehouse Cleaning Technique

This chapter gives comprehensive descriptions of the two data cleaning algorithms proposed in this thesis. The first algorithm is called "Pre-data Warehouse cleaning Algorithm" (PreWA), which is designed to clean data the first time a data warehouse is being constructed. The second algorithm, "Post-data Warehouse cleaning Algorithm" (PosWA) is designed for subsequent cleaning tasks on an existing data warehouse. Section 3.1 describes the problem domain. A detailed description of PreWA is covered in section 3.2, while PosWA is presented in section 3.3.

3.1 The Problem Domain

A good example of domain where PreWA and PosWA are needed is a financial institution such as a bank, where different units handle different transactions. For example, a savings account unit keeps track of the transactional activities of the customers in savings account. Another unit may solely be responsible for keeping the records of the activities of customers in the checking account. Assume that customers' personal information in the savings account is kept in a data source called SA, which is shown as Table 3.1-1 and that customers' personal information in the checking account is maintained in another data source called CA shown as Table 3.1-2. Assumed also that the customers' transactional activities are kept for a short period of time (e.g., 7 days) in two separate sources, TSA (for SA customers) and TCA (for CA customers). Instances of TSA and TCA are shown as Table 3.1-3 and Table 3.1-4 respectively. Both tables show the customer's identity numbers, the nature of their transactions (e.g., depositing money, withdrawing money, etc.), transaction time, and the amount of money involved in each transaction.

Data warehouse is needed in this arena so that the customers' transactional records could be kept for a longer period of time (e.g., 5 to 10 years). Building a data warehouse for the first time from the above sources requires initial cleaning, which is not obvious when the sources are viewed in isolation. Subsequent cleaning on a standing warehouse is unavoidable considering the fact that, (1) the data warehouse needs to be refreshed with the transactional activities done after the initial cleaning tasks and (2) the data warehouse could be expanded to

accommodate new units (e.g., the credit card unit), which were not part of the original data warehouse.

Although a banking environment is the primary implementation domain of our algorithms, yet they can work well in other domains such as bibliography, hospital, and transport.

Cid	CName	CBirth	CSex	CPhone	CAddress
S001	John Smith O	25-Dec-70	M	(519) 111-1234	Sunset # 995 N9B3P4
S002	Tim E. Ohanekwu	10-Jan-75	M	2566416	ABCD St. No. 695 n9b 2t7
S003	Colette Jones	08/Aug/64	M	123-4567	600 XYZ apt 5a5 N7C4K4
S004	Ambrose A. Diana	Nov/11/72	F	5196669999	4 Church Rd. N8K6t6
S005	Smith John	30-Oct-78	F	519 560 3626	182 Jesus Ave M9B3J7

Table 3.1-1: SA, Savings Account Customers

Cid	CName	CBirth	Sex	Phone	Address
1001	S. John	25-12-1970	M	1111234	995 Sunset Ave, n9b 3p4
1002	Jon Cole	08-08-1964	M	Null	XYZ No. 600 apt 585 n7c 4k4
1003	Ambo D. Dian	10-11-1972	F	566-5555	Church St. # 4 n8k 6t6
1004	Ohanekw T E	1-1-75	M	5192566416	# 695 abcd street N9B2T7
1005	Edema Tom Obi	23-Mar-1967	M	977-5950	98 Jesus Rd. M8C 8S4

Table 3.1-2: CA, Checking Account Customers

Cid	Activity	DateAndTime	Amount
S005	Deposit	1/1/02, 9.30AM	500
S001	Deposit	11/1/02, 4.30PM	1000.50
S005	Withdraw	15/3/02, 1.45PM	125
S004	Withdraw	6/4/02, 11.00AM	325.50

Table 3.1-3: TSA, a Short-lived Activity History For SA Customers

Cid	Activity	DateAndTime	Amount
1001	Deposit	2/1/02, 12.00PM	650
1004	Withdraw	5/3/02, 5.00PM	150
1005	Deposit	8/4/02, 9.45AM	1000
1002	Withdraw	15/2/02, 10.00AM	250
1003	Deposit	8/4/02, 7.00PM	450.50

Table 3.1-4: TCA, a Short-lived Activity History For CA Customers

3.1.1 The Data Warehouse Schema

Subject orientation is one of the most basic characteristics of a data warehousing system, meaning that data in it are organized around the major subject areas of the organization. For the banking environment in our example, the subjects of interest are, (1) the customers, (2) the nature of transactions carried out, (3) the source of the transaction (account type) and (4) the transaction time. Each of these subject areas constitutes a dimension table (DT) of the data warehouse. The data warehouse also contains huge fact table (FT). The primary key from each of the dimension tables appears as a foreign key in the fact table. This is what gives the data warehouse the star schema. Figure 3.1 shows the schemas of each of the dimension tables, while the star schema of the data warehouse is shown in Figure 3.2. Subsequent discussion in this thesis will be based on the customer dimension table instance shown as Table 3.1-5 and an instance of a fact table shown in Table 3.1-6.

Row	WID	Name	Sex	Phone	DBirth	Address
1	S001	John Smith O	M	(519) 111-1234	25-Dec-70	Sunset # 995 N9B3P4
2	S002	Tim E. Ohanekwu	M	2566416	01-Jan-75	ABCD St. No. 695 n9b 2t7
3	S003	Colette Jones	M	123-4567	08/Aug/64	600 XYZ apt 5a5 N7C4K4
4	S004	Ambrose A. Diana	F	5196669999	Nov/11/72	4 Church Rd. N8K6t6
5	S005	Smith John	F	519 560 3626	30-Oct-78	182 Jesus Ave M9B3J7
6	1001	S. John	M	1111234	25-12-1970	995 Sunset Ave, n9b 3p4
7	1002	Jon Cole	M	Null	08-08-1964	XYZ No. 600 apt 585 n7c 4k4
8	1003	Ambo D. Dian	F	566-5555	10-11-1972	Church St. # 4 n8k 6t6
9	1004	Ohanekw T E	M	5192566416	1-1-75	# 695 abcd street N9B2T7
10	1005	Edema Tom Obi	M	977-5950	23-Mar-1967	98 Jesus Rd. M8C 8S4

Table 3.1-5: An Instance of (yet to be cleaned) Customer Dimension Table

Row	WID	Trans-code	Account-code	Trans-time	Amount
1	S005	D	SA	570A	500
2	1005	D	CA	585A	1000
3	1004	W	CA	1020P	150
4	S005	W	SA	825P	125
5	1001	D	CA	720P	650
6	S004	W	SA	660A	325.50
7	1002	W	CA	600A	250
8	S001	D	SA	990P	1000.50
9	1003	D	CA	1140P	450.50

Table 3.1-6: An Instance of (yet to be cleaned) Fact Table

Fact-table (WID, Trans-code, Account-code, Trans-time, Amount)
Customers (WID, Name, Sex, Phone, DBirth, Address)
Transactions (Trans-code, Trans-name)
Accounts (Account-code, Account-name)
Times (Trans-time, Day, Month, Year)

Figure 3.1: Schemas of the Fact and Dimension Tables

3.1.2 The Dirt to Clean

Tables 3.1-5 and 3.1-6 contain data that show most of the data cleaning problems identified in chapter 1, and for which PreWA is being designed to clean. The dirt present in each table is described under different subsections.

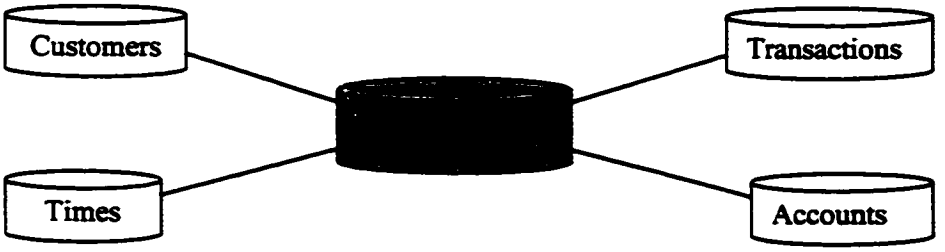

Figure 3.2: The Data Warehouse Star Schema

3.1.2.1 Dirt in the Customer Dimension Table

Two levels of dirt are identifiable from Table 3.1-5. These are *field or attribute level* dirt and *record level* dirt. By field level dirt, we mean "dirt" that occurs in each field of a given record. Record level dirt means collective fields' dirt on a given row of Table 3.1-5.

Field Level Dirt: One form of field level dirt apparent in WID field is "type mismatch", i.e., "string type" (e.g., S001) versus "integer type" (e.g., 1001). This leads to a subtle kind of synonym problem, because a given customer may be represented with different warehouse id (WID). The name field also presents a number of dirt, which includes: (1) inconsistent name values, e.g., "Colette Jones" in the name field of row 3 of Table 3.1-5 may be the same as "Jon Cole" in the same field of row 7, "S. John" (row 6) may be the same person called "John Smith O" (row 1). This is a different representation of the same entity, which is a synonym problem. (2) Typographical errors, e.g., "Ohanekw" (row 9) instead of "Ohanekwu" (row 2),

"Ambo" (row 8) instead of "Ambrose" (row 4). (3) There is also a possibility of the same name referring to two or more different entities, e.g., "John Smith" occurring in many rows may actually be referring to different persons. This gives rise to a homonym problem. There is also dirt in other fields of Table 3.1-5. A clear example of dirt in the "phone" field is format difference, e.g., "(999)-9999999" versus "999-999-9999" versus "9999999999" versus "9999999" versus "999-9999" versus "999 999 9999" versus "9-999-999-9999". The format difference is responsible for the different representations of the same telephone number, e.g., "2566416" (row 2) versus "5192566416" (row 9). Format difference is present in the "DBirth" field as well. The "Address" field contains some obvious dirt due to different addressing schemes, e.g., "Sunset # 995 N9B3P4" (row 1) versus "995 Sunset Ave, n9b 3p4" (row 6). The implication of field level dirt is that "*no particular field is clean enough to determine record duplicates*".

Record Level Dirt: The combination of all the fields' dirt produces record level dirt. For example, row 1 of Table 3.1-5 is a record with the following content (excluding the Row and WID fields), "*John Smith O. M. (519) 111-1234, 25-Dec-70, Sunset # 995 N9B3P4*". This appears to be the same person as row 6, with the following content (excluding the Row and WID fields), "*S. John, M. 1111234, 25-12-1970, 995 Sunset Ave, n9b 3p4*". An obvious implication of record level dirt is "*that duplicates are not easily determined*".

3.1.2.2 Dirt Present in the Fact Table

There is only field level dirt in Table 3.1-6 visible in the WID field. The type mismatch and lack of value uniformity in this field leads to the synonym problems in which the same customer is identified with different WID, thereby making it difficult (if not impossible) to decide all the transactions carried out by the same customer.

3.1.3 The Initial Data Cleaning Tasks

A number of initial data cleaning tasks to be carried out on Tables 3.1-5 and 3.1-6 are identified and described under different subheadings, as follows.

Data Cleaning Tasks on the Customer Dimension Table: Three main initial cleaning tasks to be performed on Table 3.1-5 by PreWA are: (1) duplicate detection, (2) duplicate elimination

and (3) proper identity generation. **Duplicate detection** task entails the combination of (pieces of) information from two or more fields to determine if two or more records showing the same or nearly the same features refer to the same entity or not. **Duplicate elimination** task is concerned with the retention of only a copy of the records detected to be the same. **Proper identity generation** task involves generating the identity on the WID field from the two or more fields that could be used to uniquely distinguish records. For example, "122570JOS" (generated from the combination of some tokens of both the "DBirth" and "Name" fields) is a proper WID for row 1 of Table 3.1-5. A target customer dimension table after the initial cleaning process on Table 3.1-5 is shown as Table 3.1-7.

Row	WID	Name	Sex	Phone	DBirth	Address
1	122570JOS	John Smith O	M	(519) 111-1234	25-Dec-70	Sunset # 995 N9B3P4
2	1175EOT	Tim E. Ohanekwu	M	2566416	01-Jan-75	ABCD St. No. 695 n9b 2t7
3	8864CJ3	Colette Jones	M	123-4567	08/Aug/64	600 XYZ apt 5a5 N7C4K4
4	111172ADD	Ambrose A. Diana	F	5196669999	Nov/11/72	4 Church Rd. N8K6t6
5	103078JS	Smith John	F	519 560 3626	30-Oct-78	182 Jesus Ave M9B3J7
6	32367EOT	Edema Tom Obi	M	977-5950	23-Mar-1967	98 Jesus Rd. M8C 8S4

Table 3.1-7: A Target Customer Dimension Table without Duplicates

Data Cleaning Tasks on the Fact Table: We need to establish a link between the customer dimension table and the fact table such that it will be possible to determine all the transactions carried out by a given entity. This is not possible with Tables 3.1-5 and 3.1-6 because different WIDs (e.g., S001 and 1001) are used to represent the same entity (John Smith O and S. John). One implication of this is that it is impossible to know that "John Smith O" with savings account also operates a checking account. This is also the case with "Tim E. Ohanekwu", "Ambrose A. Diana", etc. The solution to this problem is to use the same identity value for the same real world entity. For example, "122570JOS" will be used for all the occurrences of "S001" and "1001" in the fact table to reflect the fact that "John Smith O" and "S. John" are the same person. The same is done for records "S002" and "1004", "S004" and "1003", etc. The target fact table is shown as Table 3.1-8.

Row	WID	Trans-code	Account-code	Trans-time	Amount
1	103078JS	D	SA	570A	500
2	32367EOT	D	CA	585A	1000
3	1175EOT	W	CA	1020P	150
4	103078JS	W	SA	825P	125
5	122570JOS	D	CA	720P	650
6	111172ADD	W	SA	660A	325.50
7	8864CJ	W	CA	600A	250
8	122570JOS	D	SA	990P	1000.50
9	111172ADD	D	CA	1140P	450.50

Table 3.1-8: A Target (cleaned) Fact Table

3.2 PreWA: Pre-data Warehouse cleaning Algorithm

PreWA is designed for cleaning data the first time a data warehouse is constructed. It takes dirty tables (such as Tables 3.1-5 and 3.1-6) and returns cleaned tables (such as Tables 3.1-7 and 3.1-8). PreWA accomplishes the cleaning tasks in a number of steps.

Step 1: The selection of fields by a user: This is the step where a user plays the role of selecting some fields that can uniquely distinguish one record from another. The cleaning of the data will be based on the fields selected by the user. Thus, the user is expected to select a good combination of fields that would perfectly differentiate records. The combination of fields that would uniquely identify records varies from one domain to another. For example, the "name" field, "DBirth" field and "address" field could be a perfect combination in our banking domain. On the other hand, the "author" field, the "title" field, the "volume-number" field and the "journal" field could perfectly identify publications in a bibliographic domain. In all cases, fields like "gender", "age" and "salary" will not be adequate for the cleaning process.

Step 2: Extraction of Token Keys: Each of the fields selected by the user is decomposed into its constituting important members. For example, a date element, (e.g., "19-Dec-1978") is decomposed into three members: (19) day, (Dec) month and (1978) year, which are the important members, while characters like "-", "/" are not useful. Generally, characters like "#", ".", "(", ")", "-", " ", are considered unimportant in date, address, and telephone values. Titles like "Pastor", "Mr.", "Ms", and "Dr" may be excluded in name values, while stop

words like "the", "of", "for", "in" are considered unimportant in publication titles. Further operations (e.g., conversion, further decomposition, etc.) on some members may be necessary before the re-composition operation. For example, "Dec" is converted to its numeric equivalence (12), while "1978" is further decomposed into the century (19) and year (78) parts. The century part is discarded. Thus, the surviving members from the original value (19-Dec-1978) are 19, 12 and 78. These are sorted in ascending order and recomposed to form the token key (121978) for a given record on the date field. Different rules are used to extract token keys for different types of fields. Further description of each of the token extraction functions is given in section 3.2.1.

Step 3: Sorting of the Token Key Table: Two tables are produced from the original token key table that emerged after step 2. The first table emerges by sorting the token key table on the token key field extracted from the most discriminating field (e.g., the DBirth field), while the second table comes from sorting the token key table on the token key field extracted from the next most discriminating field (e.g., the name field). Sorting the token key field of choice on ascending order produces both tables. Further discussion on this step is provided in subsection 3.2.2.

Step 4: Duplicates detection and elimination and WID Generation: There are three sub-steps here. The first is concerned with determining if records in closest proximity refer to the same entity or not. PreWA adopts both the **exact matching** and the **similarity matching** techniques for duplicate detection task. The exact-matching technique will consider two records as a match only when both match "character-to-character". In contrast, similarity matching does not insist that two records match "character-to-character" to be considered the same. Instead, all the (or substantial number of) characters in one must be contained in the other. Thus, "Tim" and "Timothy" are not the same when the exact matching scheme is used, but the two tokens would be identified as the same when the similarity-matching scheme is used. Duplicate elimination is the second sub-step and is simply concerned with the retention of one copy of the records detected to be the same. The third sub-step is to generate a WID for records by joining the two token keys used for sorting in step 3. More comprehensive description of step 4 (including duplicate detection rules and propositions) is given in subsection 3.2.3.

3.3: PreWA (Input: Dirty tables e.g., Tables 3.1-5 and 3.1-6)
Output: Clean Tables, e.g., Tables 3.1-7 and 3.1-8
BEGIN

Step 1:
Display a list of fields for a user to select at least two fields that could be combined to differentiate records from a pool of records. Also, select a proper field description from the field description options for each of the fields selected.

Step 2:
For each field selected do
Decompose the value into n (n >= 1) token member(s); apply further operations (e.g., conversion, further decomposition) to some members if necessary; extract some (or all the) parts of the members, sort them in a certain order (e.g., ascending) and combine them to get a single token key for the value

Step 3:
Sort the token key table on two token key fields extracted from the two most discriminating fields

Step 4:
For all the records in immediate neighborhood do
Apply duplicate detection procedures
If two or more records are found to be duplicates then
Apply WID generator for duplicates, eliminate duplicate copies, and apply results to the tables
Else
Apply WID generator for non-duplicates, and apply results to the tables

END

Figure 3.3: Main Steps of PreWA

Figure 3.3 shows the formal and compact version of the four steps given above. Three definitions of "*immediate neighborhood*" in step 4 of Figure 3.3 are given depending on the position of a record in a dataset where the number of records is greater than 1. A given record R could be in any of the following three positions: (1) first, (2) last and (3) anywhere between first and last position. A record R in the first position, fp, has only one neighboring record NR at position fp +1. A record R in the last position, lp, has only one neighboring record NR at position lp – 1. A record R at neither the first nor last position, mp, has two neighboring

records NRs at positions mp –1 and mp + 1. These definitions are employed in the duplicate detection phase covered in sub-section 3.2.3.1.

3.2.1 Extraction of Token Keys

Token keys are extracted from some parts or subparts of the sorted tokens. Three types of token keys are identified in this thesis, namely, numeric token keys (obtained from numeric-dominated attribute values), alphabetic token keys (obtained from alphabetic-dominated fields) and alphanumeric token keys (extracted from fields consisting of both numeric and alphabetic members, e.g., address field).

Numeric Token Keys: These keys consist of only digits (0 – 9) and are likely obtained from numeric fields like "telephone numbers", "dates of birth", "employee ids", "student ids", "social insurance numbers" (SSN). Three functions are defined to extract token keys from any numeric fields. For example, a "*telephone-token-extractor*" is a function that takes values in any of the following forms, "(519) 256 6416", "5192566416", "1-519-256-6416", "2566416", "256-6416", "256 6416", etc., and returns the last 7 digits, namely, "2566416". Similarly, a "*date-token-extractor*" is a defined function that extracts token keys from date-formatted values (like dates of birth, dates of publications, dates of employment, etc.). In other words, this function takes values of any of the following forms: "12-10-78", "10th, Dec, 1978", "10th, December, 78", "December, 10th, 78", etc., and returns numeric token key of the following nature: 101278. Extracting token keys from other regularly formatted numeric fields (e.g., SSN, student id, etc.) is achieved by yet another function called "*regular-numeric-token-extractor*", which when fed with values of the following nature: "999-666-666", "(999) (666) (666)", "999 666 666" etc. returns a numeric token of the form: 999666666.

Alphabetic Token Keys: These keys consist only of alphabets (aA – zZ) and are obtained from fields consisting of only alphabets, e.g., "name of persons", "company names", "journal names", "publication titles", etc. A function called "alphabet-token-extractor" is defined to take alphabet-dominated values like "Mr. Ohanekwu Tim Emenike" and return alphabetic token key of the form: "EOT". Each letter in "EOT" is taken from the first letter of the token

members: "E" from Emenike, "O" from Ohanekwu and "T" from Tim. The letters in the emerged token key are sorted in a given order (ascending in this case).

Alphanumeric Token Keys: The keys in this category may consist of both numeric and alphabetic parts. Such keys can be obtained from field values that contain both numbers and strings. A good example is an "address" field. A function called "*alphanumeric-token-extractor*" is defined, which (1) decomposes a given alphanumeric value into a number of members, (2) scans through the set of members and selects only tokens that are either numeric or alphanumeric in nature, (3) further decomposes each of the alphanumeric part to its numeric and alphabetic parts, and (4) sorts the set of tokens in certain order (e.g., ascending) to get an alphanumeric token key. Applying the above sub-steps on "600 XYZ blvd apt 5a5 N7C4K4" yields "55 600 744 A NCK".

Example: Given that Table 3.1-5 is the table from which a user selects fields for cleaning. Assuming that the user selects three fields, namely, "Name", 'DBirth" and "Address". Applying the token key extraction procedures already described would result in a token key table shown as Table 3.2-1.

Row	WID	NameKey	DBirthKey	AddressKey
1	S001	JOS	122570	934995NBP
2	S002	EOT	1175	927695NBT
3	S003	CJ	8864	74455600AKNC
4	S004	ADD	111172	4866NKT
5	S005	JS	103078	937182JMB
6	1001	JS	122570	934995NBP
7	1002	CJ	8864	744585600KNC
8	1003	ADD	101172	4866NKT
9	1004	EOT	1175	927695NBT
10	1005	EOT	52367	88498MCS

Table 3.2-1: Token Keys Table

Extracting tokens from n number of fields in N number of records requires time in the order of (nN), while sorting within tokens takes Log_2N time. Hence step 2 has overall time complexity of Θ(NLogN).

3.2.2 Sorting the Token Keys Table

One novel idea being introduced in this thesis is the use of the (short lengthened) token keys for record comparison purposes. Previous algorithms (HS95a, HS95b, HS98) use the extracted keys to bring likely duplicates to a close neighborhood only, though the keys are also used for clustering in [HS95b, HS98]. The concept of tokens mentioned in [LH99] differs significantly from ours in that their tokens are much longer than ours. It had long been discovered that sorting is the surest way to bring same record together [BD83]. Two tables, Tables 3.2-2 and 3.2-3 are produced by respectively sorting Table 3.2-1 on the two most discriminating token key fields. Table 3.2-2 is the outcome of sorting Table 3.2-1 on the "DBirthKey" field, while Table 3.2-3 results from sorting Table 3.2-1 on the "NameKey" field. The decision to sort the token keys table on the two most powerful token fields hinges on the fact that a singular token field may fail to bring potential duplicates together due to digit or letter differences in the tokens. These are the cases with records 4 and 8 in Table 3.2-1 and records 1 and 6 in Table 3.2-2. Records 4 and 8 are not in proximity due to a digit difference in the DBirthkey token upon which the sorting is based. The presence of a letter in the NameKey token of record 1 and its absence in record 6 is responsible for the spreading apart of the two possible duplicates. This means that there will be a maximum of 2 independent passes during the duplicate detection and elimination phase considered next. This is a significant improvement over multi-pass version of the algorithms in [HS95b, HS98] where the number of runs varies proportionally to the number of fields selected for cleaning.

The sorting of token key table in step 3 has a time complexity of (Log_2N), N being the number of records in the dataset.

Row	WID	NameKey	DBirthKey	AddressKey
2	S002	EOT	1175	927695NBT
9	1004	EOT	1175	927695NBT
3	S003	CJ	8864	74455600AKNC
7	1002	CJ	8864	744585600KNC
10	1005	EOT	52367	88498MCS
8	**1003**	**ADD**	**101172**	**4866NKT**
5	S005	JS	103078	937182JMB
4	**S004**	**ADD**	**111172**	**4866NKT**
1	S001	JOS	122570	934995NB P
6	1001	JS	122570	934995NB P

Table 3.2-2: Token keys Table sorted on the DBirthkey token field

Row	WID	NameKey	DBirthkey	AddressKey
4	S004	ADD	111172	4866NKT
8	1003	ADD	101172	4866NKT
3	S003	CJ	8864	74455600AKNC
7	1002	CJ	8864	744585600KNC
2	S002	EOT	1175	927695NBT
9	1004	EOT	1175	927695NBT
10	1005	EOT	52367	88498MCS
1	**S001**	**JOS**	**122570**	**934995NBP**
5	S005	JS	103078	937182JMB
6	**1001**	**JS**	**122570**	**934995NBP**

Table 3.2-3: Token keys Table Sorted on the NameKey field

3.2.3 Duplicate Detection, Elimination and WID Generation

This step comprises three sub-steps: (1) duplicates detection, (2) duplicate elimination and (3) generating of the warehouse identity.

3.2.3.1 Duplicate Detection

This is accomplished by comparing neighboring records on Tables 3.2-2 and 3.2-3. The motivation to use the token keys for record comparison is predicated on the following valid argument:

> *If the token keys are sufficiently strong enough to bring duplicates together, then they can equally be used to determine record match*

The above argument is more formally expressed as proposition 1:

Proposition 1:

Two or more records from different sources within the same application domain would most likely have the same or nearly the same token keys if such keys were extracted from the most uniquely identifying attributes of the records.

PreWA Record Matching Path: An illustration of the pathway followed by PreWA in an attempt to determine if two records being compared match or not is given next. Given two records, R_1 and R_2 being compared. R_1 and R_2 have m pairs of token key fields as follows:

$R_1t_1, R_1t_2, \ldots, R_1t_m; R_2t_1, R_2t_2, \ldots, R_2t_m$

The first task is for PreWA to determine a similarity match count (SMC), which indicates the number of corresponding token key pairs that match perfectly. The value of SMC determines whether the match is (1) perfect, (2) near perfect, (3) maybe and (4) no match at all.

A Perfect Match: SMC equals the number of fields used for cleaning. The conclusion is that R_1 and R_2 are the same. Perfect match could also be on field (token) level basis, which is explained as follows. Given an n-character token t_1 belonging to R_1 and an n-character token t_2 for R_2, there is a perfect match between tokens t_1 and t_2 if both are the same at all points. Mathematically:

$$R_1t_i = R_2t_i,\ i = 1, \ldots, n$$

Thus, there is a perfect match on the AddressKey token of records S002 and 1004 of Tables 3.2-2 and 3.2-3 since $S002_{\{927695NBT\}} = 1004_{\{927695NBT\}}$. Similarly, there is a perfect match on record level between records S002 and 1004 because all the corresponding fields match perfectly since $S002_{\{EOT,\ 1175,\ 927695NBT\}} = 1004_{\{EOT,\ 1175,\ 927695NBT\}}$. A similarity matching function on S002 and 1004 returns a similarity match ratio (SMR) of 1.0. Generally, a perfect match has an SMR of 1.0.How the SMR is determined is explained in the "maybe a match" section.

Near Perfect Match: SMC is not equal to the number of fields used in cleaning, but close to it. For any two records R_1 and R_2 to be considered a "near perfect match", their SMC must lie somewhere between 0.67 and 0.99. Two records, R_1 and R_2 showing a "near perfect match" features are taken as the same. It becomes evident with the help of the above definition of a "near perfect match" that records S003 and 1002 in both Tables 3.2-2 and 3.2-3 are near perfect match (hence the same entity) since 2 out of 3 token key fields match perfectly thereby recording a similarity match count (SMC) of 2/3 (0.67). The SMR function on the three token key fields of S003 and 1002 returns an SMR of 0.94. How this figure is obtained is given below.

Maybe a Match: Record R_1 "may be a match" to record R_2 if at least one of their corresponding tokens match perfectly. Formally, given two adjacent records R_1 and R_2 with n pairs of token keys (R_1t_1, R_1t_2, ... R_1t_n; R_2t_1, R_2t_2, ... R_2t_m), R_1 may-be a match to R_2 if at least one of these is true: $\mathbf{R_1t_1 = R_2t_1}$ **Or** $\mathbf{R_1t_2 = R_2t_2}$ **Or ...** $\mathbf{R_1t_n = R_2t_m}$. Maybe a match is not sufficient to conclude that R_1 is identical to R_2, but is an indicator that the records being compared may be referring to the same entity. Similarity matching function is applied on the tokens that did not match. The size of the similarity match ratio will determine whether R_1 and R_2 will be considered the same of not.

Similarity Match Ratio: Given two tokens t_1 and t_2 with m characters and n characters respectively. Assuming that n = 5 and m = 6. The Similarity Match Ratio (SMR) is the number of common characters in t_1 and t_2 divided by the average length of m and n. That is, SMR = (number-of-common-characters in t_1 and t_2) / (n + m)/2.

Example: It is evident from Table 3.2-3 that there is a maybe match situation for S005 and 1001 since their NameKey tokens match perfectly. The SMR function is applied to the other two token key pairs that did not match, namely, DBirthKey and AddressKey token keys. The SMR = 6/15 = 0.4. The conclusion from the low value of SMR is that S005 is not the same person as 1001, though their NameKey tokens are the same. PreWA can only match both records only when their SMR is as high as 0.8 or greater. High SMR is necessary in PreWA in order to avoid false matches at a "maybe match" situation.

No match at all: This is the case where SMC is 0, i.e., no token key pair match for R_1 and R_2. Formally, given records R_1 and R_2 with n pairs of token keys as follows: $\{(R_1t_1, R_2t_1), (R_1t_2, R_2t_2) \ldots (R_1t_n \ldots R_2t_n)\}$. R_1 does not match R_2 at all if: $\mathbf{R_1t_1 \neq R_2t_1}$ **And** $\mathbf{R_1t_2 \neq R_2t_2}$ **And ...** $\mathbf{R_1t_n \neq R_2t_n}$. Thus, PreWA will conclude that two records are not identical if their SMC is 0.

The best case for the duplicate detection is when the detection process ends at either a "perfect match", "near perfect match" or "not a match at all". The time required is n * (N-1), where n is the number of token fields, while N is the total number of records in a dataset. The worst case is when duplicate detection reaches "maybe a match" stage, in which the time required is n * (N-1) + m*l*, where m is the number of tokens fields involved in the similarity match, while *l* is the longest length of the tokens.

Duplicate Detection Results: The results of applying the duplicate detection procedures discussed so far separately on the sorted token key tables (Tables 3.2-2 and 3.2-3) are shown as Tables 3.2-4 and 3.2-5. Table 3.2-4 is the result of applying the duplicate detection procedures on Table 3.2-2 (token keys table sorted on the DBirthKey), while Table 3.2-5 emerges when the same procedures are applied on Table 3.2-3 (token keys table sorted on the NameKey).

The entries on the "Identified-As-Record-Above" and "Identified-As-Record-Below" columns of Tables 3.2-4 and 3.2-5 are explained as follows. Each record, R at a given row, j (where j is neither the first nor the last row) has two neighbors at rows j-1 (above) and j+1 (below). There are three possible entries in the "Identified-As-Record-Above" and "Identified-As-Record-Below" columns: (1) "/" means that the record at the row j does not match the neighboring record at either rows j-1 or j+1, (2) "NA" means that the record at row j is either the first or last record, therefore has no neighbor at either row j-1 or j+1 and (3) "an integer value" means that the record at row j is a match to the record at either row j-1 or j+1 whose Row value equals the integer value. For example, record S001 occupies position 9 (j = 9) in Table 3.2-2. Its two neighbors are records at positions j-1 (i.e., record S004) and j+1 (i.e., record 1001). The comparison of S001 and S004 shows that they are not a match, hence the entry "/" on the "Identified-As-Record-Above" of Table 3.2-4 of record S001. However,

the comparison of "S001" and "1001" reveals that both refer to the same entity, hence the entry "6" on the "Identified-As-Record-Below" of Table 3.2-4 of record S001.

Row	WID	Identified-As-Record-Above	Identified-As-Record-Below
1	S001	/	6
2	S002	NA	9
3	S003	/	7
4	S004	/	/
5	S005	/	/
6	1001	1	NA
7	1002	3	/
8	1003	/	/
9	1004	2	/
10	1005	/	/

Table 3.2-4: The Result of Duplicate Detection Procedures on Table 3.2-2

Row	WID	Identified-As-Record-Above	Identified-As-Record-Below
1	S001	/	/
2	S002	/	9
3	S003	/	7
4	S004	NA	8
5	S005	/	/
6	1001	/	NA
7	1002	3	/
8	1003	4	/
9	1004	2	/
10	1005	/	/

Table 3.2-5: The Result of Duplicate Detection Procedures on Table 3.2-3

The Integration of the Duplicate Detection Results: Tables 3.2-4 and 3.2-5 are combined resulting in Table 3.2-6. Simultaneously scanning Tables 3.2-4 and 3.2-5 and collecting the items on the "Identified-As-Record-Above" and "Identified-As-Record-Below" columns yields the entries on the ResultSet-0 of Table 3.2-6. Further operation is applied on the ResultSet-0 column culminating in the more integrated version shown in ResultSet-1 column. A procedure containing a number of rules examines each set on the ResultSet-0 column and removes duplicates, and entries like"/" and "NA". The number of elements in each set of the

ResultSet-1 column could be zero (null set), one or more depending on the entries on the "Identified-As-Record-Above" and "Identified-As-Record-Below" columns of Tables 3.2-4 and 3.2-5. The "Row", "WID" and "ResultSet-1" columns of Table 3.2-6 are used for the duplicate elimination and WID generation task explained next. The time required to collect the entries in ResultSet-0 is N, while the next step resulting in the entries in ResultSet-1 requires Nk.

Row	WID	ResultSet-0	Result Set-1
1	S001	{/, 6, /, /}	{6}
2	S002	{NA, 9, /, 9}	{9}
3	S003	{/, 7, /, 7}	{7}
4	S004	{/, /, NA, 8}	{8}
5	S005	{/, /, /, /}	{}
6	1001	{1, NA, /, NA}	{1}
7	1002	{3, /, 3, /}	{3}
8	1003	{/, /, 4, /}	{4}
9	1004	{2, /, 2, /}	{2}
10	1005	{/, /, /, /}	{}

Table 3.2-6: Result of the Integration of Tables 3.2-4 and 3.2-5

Thus the integration of duplicate detection results has a time complexity of N + Nk, where N equals the number of records, while k is the number of entries in ResultSet-0 column.

3.2.3.2 Duplicate Elimination and WID Generation

The procedure that scans through Table 3.2-6 in order to collect duplicates, eliminate duplicates and generate WID is shown in Figure 3.4. Let us explain Figure 3.4 using the concept of linked list shown in Figure 3.5. All the nodes (except the rightmost ones) consist of two parts, the row part and the record Id part. The first node on each line of Figure 3.5 is the starting node, while the last node is the stopping node. Note that there are two loop-terminating conditions: (1) the value on the first part of the last node equals the value on the first part of the first node, and (2) the value on the first part of the last node is null. The rightmost node on each line shows the duplicate list for each loop. For example, the topmost node shows that records "S001" and "1001" are duplicates, while the lowest node shows that record "1005" has no duplicate. One WID is generated for the element(s) on the duplicate list.

For Each Row k in Table 3.2-6 that is not 0 do

(1) Put the token keys of k in a duplicate list and replace Row[k] value with 0

(2) Add the record pointed to by the integer value on the ResultSet of k into the duplicate list if non-empty otherwise go to step (5) below

(3) For the record pointed to do

(4) Check if the value on the ResultSet is {}

If "Yes" then

(5) **WID Generation Procedure:**

Generate WID for the records in the duplicate list; Remove duplicate copies from the dimension tables if the number of records in the duplicate list is more than 1 and update the retained copy with the value of the WID; Apply the generated WID to the corresponding records in the fact table; Update the log tables accordingly; Set duplicate list to null

Else if "No" then

If value equals k then

(6) Go to step (5) above

Else

(7) Repeat step (2)

Figure 3.4: Duplicate Elimination and WID Generating Procedure

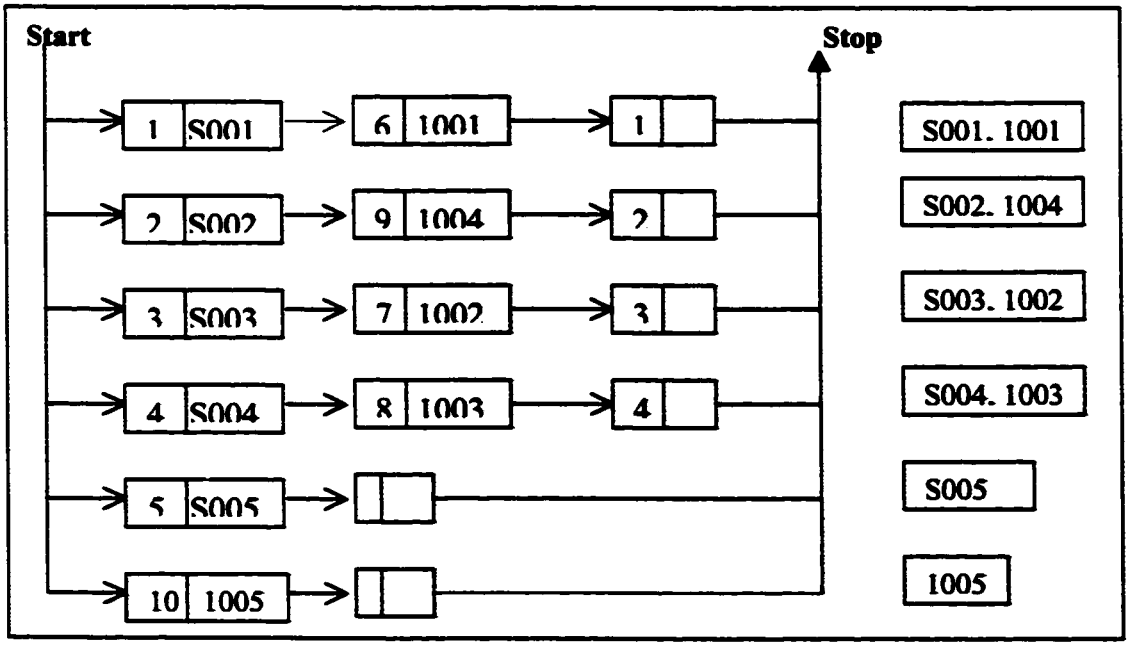

Figure 3.5: Linked List Version of Duplicate Collection and Elimination Procedure

Row	CID	NameKey	DBirthKey	AddressKey	WID
1	S001	JOS	122570	934995NBP	122570JOS
2	S002	EOT	1175	927695NBT	1175EOT
3	S003	CJ	8864	74455600AKNC	8864CJ
4	S004	ADD	111172	4866NKT	111172ADD
5	S005	JS	103078	937182JMB	103078JS
6	1001	JS	122570	934995NBP	122570JOS
7	1002	CJ	8864	744585600KNC	8864CJ
8	1003	ADD	101172	4866NKT	111172ADD
9	1004	EOT	1175	927695NBT	1175EOT
10	1005	EOT	52367	88498MCS	52367EOT

Table 3.2-7: A Log Table Generated by PreWA

The WID is obtained by concatenating the two most discriminating token keys used in sorting the token key table as already described. If the duplicate list has more than one element then the WID is generated from the first element. The record used for generating the WID is retained in the customer dimension table (Table 3.1-5), while the others are deleted from it. The fact Table (Table 3.1-6) is updated appropriately. At this point, the initial data cleaning will have been successfully accomplished and the duplicate-free dimension table (see Table 3.1-7) and a cleaned fact table (see Table 3.1-8) would be produced. In addition, a log table (Table 3.2-7) is generated and saved. Table 3.2-7 has CID column that contains the original customer id prior to cleaning process, as well as a WID column that contains the generated Warehouse Ids for the corresponding records. Table 3.2-7 also has token keys columns for the token keys extracted from the selected fields. Table 3.2-7 is required for the PosWA operations. The duplicate Elimination and WID generation has a time complexity of $\Theta(N)$, while the generation of Table 3.2-7 requires $\Theta(nN)$, where n is the number of fields and N is the number of records.

3.3 PosWA: Post-data Warehouse cleaning Algorithm

A data warehousing system should be both refreshable and expandable. Expandability of the data warehouse is necessitated by the fact that the underlying data sources may increase from time to time. For example, the initial data warehouse cleaned in section 3.2 has two starting data sources, SA and CA. The management of the bank may decide to include the credit card unit to the data warehouse. Two main options readily come to mind. The first option is to start the cleaning process from the scratch each time a new unit is added to the data warehouse. This option is fraught with a number of overheads and is not feasible at all, because it is both economically and computationally expensive. The second option is to reuse some of the resources generated by PreWA (e.g., Table 3.2-7) to accommodate any new entrant to the data warehouse. There is the problem of refreshing the already cleaned warehouse with subsequent transactions from Savings Accounts (SA) and Checking Accounts (CA) sources. The question we are seeking answer to is: given existing customer, say, "Tim E Ohanekwu" already having a WID of 1175TOE and token keys of "TOE", "1175" and "927695NBT", is there a way to use the information about this customer to quickly clean new records belonging to him? In this section, we detail a description of an algorithm called "PosWA", which can be used for subsequent cleaning of a standing data warehouse.

Assumptions: We assume that the data to refresh and/or expand the existing data warehouse with are fetched from Table 3.3-1. Table 3.3-2 is another table that gives the personal information of the records in Table 3.3-1. The contents of Table 3.3-1 reflect the activities carried out after the last refreshing or expanding operations on the data warehouse. PosWA will be halted if this table is empty.

CID	Source	Activity	Amount	Time	Date
S001	SA	W	250	779A	1/4/02
1006	CA	D	500	717P	3/3/02
S001	SA	D	300	771A	5/4/02
1001	CA	W	100	915P	7/4/02
CC001	CC	D	250.50	1015P	1/4/02
CC002	CC	W	100	666A	10/4/02
1006	CA	D	300	611P	8/4/02

Table 3.3-1: Data to Refresh or Expand the Data Warehouse with

CID	Name	Sex	Birth	Address
S001	John Smith O	M	25-Dec-70	Sunset # 995 N9B3P4
1006	Florence Mike	F	23-Nov-66	334 Rankin M7V 2J2
1001	S. John	M	25-12-1970	995 Sunset Ave, n9b 3p4
CC001	John Smith	M	December 25 1970	No. 995 Sunset Ave N9b 3p4
CC002	Zach Young	F	Jul-30-68	345 AP St P6C 6K1

Table 3.3-2: The Personal Information of the Records in Table 3.3-1.

3.3.1 Detailed Description of PosWA

The description of PosWA is summarized in Figure 3.6 and is explained further with an example as follows.

3.2-3: PosWA (Inputs: Table 3.3-1, Table 3.3-2)
Outputs: Expanded Warehouse Tables
Given a dataset, TA of records (such as Table 3.3-2)

(1) For each entry, R in TA do

(1.1) Compose token keys for R in the same way as in PreWA
(1.2) Form a cluster, c from the WH log table (Table 3.2-7) such that the record(s) in c has (have) same token keys as the ones composed in step 1.1 above.

(1.3) Examine c
If c has no element then

(1.3.1) Form token key(s) from other selected field(s) (if applicable); Form the WID accordingly; Insert R from Table 3.3-2 into the customer dimension table; Insert transactions belonging to R into the fact table; Update the WH log table accordingly.

Else if c has more than 1 element then

(1.3.2) Form token key(s) from other selected field(s) (if applicable); Use similarity match function to determine which record in c is R and insert transactions into the fact table with the WID of existing record that matched R.

Else if c has only one element then

(1.3.3) Insert transactions into the fact table with the WID of the existing record that matched R

Figure 3.6: Procedure for Subsequent Data Warehouse Cleaning by PosWA

Example: The procedure contained in Figure 3.6 is exemplified as follows. The inputs are Tables 3.3-1 and 3.3-2. The TA (dataset) is Table 3.3-2.

Let us go through the algorithm with the first record in Table 3.3-2.

1.1 WID Composition: Form token key, t_1 from the Name field is "JOS" and the token key, t_2 from the Birth field is "122570". The Warehouse ID, WID_1 for this record is $t_2 + t_1$ = "122570JOS".

1.2 Cluster Formation: A table, c is obtained by querying the WH log table as follows.

> Select * From Table 3.2-7 Where NameKey = t_1 and DBirthKey = t_2

The cluster formation above returns a table shown below.

Row	CID	NameKey	DBirthKey	AddressKey	WID
1	S001	JOS	122570	934995NBP	122570JOS

1.3 Determine number of elements in the table returned by the query in 1.2 above. Since there is one entry in the table, control is transferred to section 1.3.3.

1.3.3 Warehouse Table Update: Only the fact table will be affected with all the transactions belonging to record "S001" in Table 3.3-1.

The outcomes of following through with other rows in Table 3.3-2 will expand the Customers dimension table (Table 3.1-7), the fact table (Table 3.1-8) and the log table (Table 3.2-7) with the records respectively shown in Table 3.3-3, Table 3.3-4 and Table 3.3-5.

Row	WID	Name	Sex	Phone	DBirth	Address
7	112366FM	Florence Mike	F		23-Nov-66	334 Rankin M7V 2J2
8	73068YZ	Zach Young	F		Jul-30-68	345 AP St P6C 6K1

Table 3.3-3: The Customer Dimension Table after PosWA's operation on Tables 3.3-1 and 3.3-2

Row	WID	Trans-code	Account-code	Trans-time	Amount
10	122570JOS	W	SA	779A	250
11	122570JOS	D	SA	771A	300
12	112366FM	D	CA	717P	500
13	122570JOS	W	CA	915P	100
14	122570JOS	D	CC	1015P	250.50
15	73068YZ	W	CC	666A	100
16	112366FM	D	CA	611P	300

Table 3.3-4: The Fact Table after PosWA's operation on Tables 3.3-1 and 3.3-2

ow	CID	NameKey	DBirthKey	AddressKey	WID
11	1006	FM	112366	722334JMV	112366FM
12	CC001	JS	122570	934995NBP	122570JOS
13	CC002	YZ	73068	661345KPC	73068YZ

Table 3.3-5: The Log Table after PosWA's operation on Tables 3.3-1 and 3.3-2

CHAPTER 4
Implementation and Performance Analyses

In this chapter, we present some results of the experiment conducted to analyze the performance of PreWA and PosWA as well as two other data cleaning algorithms, namely, the "Basic Field Matching Algorithm (BFMA) [ME96] and the algorithm that uses external sources for field preprocessing (External-based) [LH99]. First, we describe the implementation environments in section 4.1. The parameters used to measure the performances of data cleaning algorithms are given in section 4.2. Four case experimental results are presented and analyzed in section 4.3. Finally, we give the limitation of PreWA and PosWA in section 4.4.

4.1 The Implementation Environments

We consider the hardware and software platforms on which PreWA, PosWA and other two comparable algorithms were implemented.

Hardware platforms: All the experiments were performed on a Pentium III with 733 MHZ, 128 RAM and 20.4 GB hard drive.

Software platforms: (1) Operating system: PreWA and PosWA were implemented on a computer controlled by Windows 2002 Professional. (2) Database Management System (DBMS): Both input and output data were stored in a database managed by Oracle 8i DBMS. (3) Programming language: The original implementation programming language for PreWA and PosWA is Java (SDK 1.3).

Input data: The data cleaned by PreWA and PosWA were real data taken from telephone directory and other official documents such as medical directory containing names of medical personnel. Some names were also taken from the list of research students in the database laboratory of the University of Windsor. We added some missing fields in the sample data to get the desired metadata. We also carefully introduced a wide range of "dirt" into the data. The dirt introduced include: (1) typographical errors, (2) transposition errors, (3) inconsistent use of initials in names, (4) different addressing schemes, (5) synonyms, (6) homonyms, (7) data duplications and (8) data format differences

4.2 Performance Measures for Data Cleaning Algorithms

The **degree to which data quality could be improved** is a clear effectiveness measure of any algorithmic solution to data cleaning problems. The two most popular performance measures are: r*ecall* (also known as percentage hits) and *false-positive error* (referred also as false merges). The third parameter used to measure performance is the r*esponse time*. In this thesis, we introduced the "*reverse false-positive error*" as one of the criteria to measure performances. Another parameter we used in this thesis to measure performances is "performance dependency on threshold". Each of these is described below.

Recall: this is the percentage of duplicate records correctly identified by a given data cleaning algorithm. In other words, given "m" number of duplicates in a dataset, if the algorithm identifies "n" number of duplicates then the recall is: (n/m) x 100. Therefore, the recall for Table 3.2-2 is **6/8 * 100 = 75%**, while the recall for Table 3.2-3 is **6/8 * 100 = 75%**, meaning that sorting on name token key produced the same result as sorting on date of birth token key. However, there is a 100% recall when both tables are combined. *A good data cleaning algorithm should have high recall.*

False-positive Error (FP-E): this is the opposite of *precision ratio* and indicates the percentage of records wrongly identified as duplicates. Formally, false-positive error = *(number of wrongly identified duplicates / total number of identified duplicates) x 100.* There is no false-positive error on the matching results shown in Tables 3.2-2 and 3.2-3. *A good data cleaning algorithm should have a very low (better if zero) false-positive error.*

Reverse False Positive-Error (RFP-E): in this thesis, we coin out a definition for a situation where a given cleaning algorithm fails to identify duplicate records. This is what we term "**Reverse False Positive-Error**" (RFP-E), and is formally defined as: (*number of duplicates not identified / total number of duplicate) x 100.* Both Tables 3.2-2 and 3.2-3 have RFP-E of 25%, but their combination has RFP-E of 0%. *A good data cleaning algorithm should have a very low (better if zero) RFP-E.*

Response time is defined in data cleaning context as *the time lapse between when the data cleaning algorithm is applied on the "dirty" dataset and when the cleaning is done.* Response time is largely affected by the frequency of secondary storage or external

source access. The more frequent external sources are accessed, the longer the response time. The length of strings involved in the record comparison phase could partly affect the response time. The longer the records, the more time it takes to compare them, hence the longer the response time. *A good data cleaning algorithm should have relatively short response time.*

Of all the criteria described above, the recall is the most significant. Most data cleaning algorithms sacrifice response time for recall, FP-E and RFP-E. The argument in favor of this is valid, namely, "*it is far better to arrive at a correct result in a longer time than getting spurious result in a very short time*". Nonetheless, it is still possible to achieve a correct result in a short period of time. Achieving an (or near) optimal cleaning correctness in good response time is the main target of PreWA and PosWA.

4.3 Four Case Experiments

In order to evaluate the performance of PreWA as well as to compare its performance with two other data cleaning algorithms, we conducted four different experiments. First, we started with input of small size containing some trivial "dirt". We gradually increased the size of the input data as well as the nature of "dirt". For each experiment, we varied the threshold trice, starting from a low threshold (0.25) then medium threshold (0.44) to high threshold (0.80). Each of the four case experiments is described below.

Case 1: 20 rows of records, 4 pairs of duplicates and other trivial dirt: the result of running the three algorithms on three different thresholds is shown in Table 4.1-1. This table shows the results on the "recall", "false positive error", "reverse false positive error" and the response time of the three algorithms being compared.

Case 2: 40 rows of records, 7 pairs of duplicates and slightly less trivial dirt: running the algorithms on the input data this time produced the result shown in Table 4.1-2.

Case 3: 80 rows of records, 10 pairs of duplicates and much less trivial dirt: we present the outcomes of running the three algorithms in this scenario in Table 4.1-3.

Case 4: 120 rows of records, 14 pairs of duplicates and nontrivial dirt: Table 4.1-4 contains the results recorded for the three algorithms under case 4.

		Recall	FP-E	RFP-E	Time (Sec)
Case 1	*Threshold = 0.25*				
	PreWA	4	0	0	3
	BFMA	3	5	1	3
	External-based	2	12	2	4
Case 2	*Threshold = 0.44*				
	PreWA	4	0	0	3
	BFMA	3	1	1	3
	External-based	3	2	1	4
Case 3	*Threshold = 0.80*				
	PreWA	4	0	0	3
	BFMA	0	0	4	3
	External-based	0	0	4	4

Table 4.1-1: 20 Rows of Records at Varied Thresholds

		Recall	FP-E	RFP-E	Time (Sec)
Case 1	*Threshold = 0.25*				
	PreWA	7	1	0	5
	BFMA	5	4	2	5
	External-based	5	17	2	6
Case 2	*Threshold = 0.44*				
	PreWA	7	1	0	5
	BFMA	6	0	1	5
	External-based	5	3	2	6
Case 3	*Threshold = 0.80*				
	PreWA	7	0	0	5
	BFMA	1	0	6	5
	External-based	0	0	7	6

Table 4.1-2: 40 Rows of Records at Varied Threshold

		Recall	FP-E	RFP-E	Time (Sec)
Case 1	*Threshold = 0.25*				
	PreWA	10	3	0	7
	BFMA	7	12	3	7
	External-based	7	58	3	9
Case 2	*Threshold = 0.44*				
	PreWA	10	2	0	7
	BFMA	7	2	3	7
	External-based	8	6	2	9
Case 3	*Threshold = 0.80*				
	PreWA	10	0	0	7
	BFMA	1	0	9	7
	External-based	0	0	10	9

Table 4.1-3: 80 Rows of Records at Varied Threshold

		Recall	FP-E	RFP-E	Time (Sec)
Case 1	*Threshold = 0.25*				
	PreWA	12	9	2	8
	BFMA	8	19	6	9
	External -based	8	111	6	11
Case 2	*Threshold = 0.44*				
	PreWA	13	7	1	8
	BFMA	8	3	6	9
	External-based	8	13	6	11
Case 3	*Threshold = 0.80*				
	PreWA	14	0	0	8
	BFMA	1	0	13	9
	External-based	0	0	14	11

Table 4.1-4: 120 Rows of Records at Varied Threshold

The recalls of the three algorithms at the three thresholds are graphically shown in Figures 4.1, 4.2, and 4.3. The patterns of FP-E and RFP-E are also shown in Figures 4.4 and 4.5 respectively.

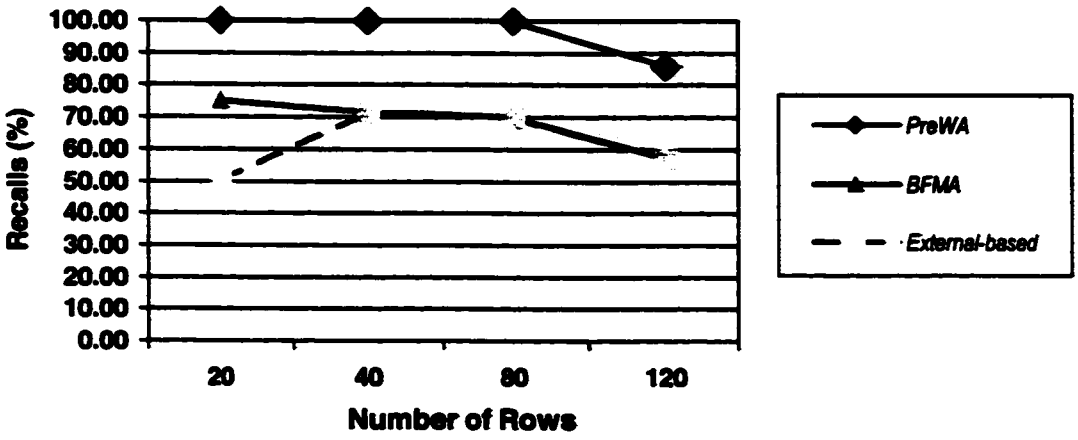

Figure 4.1: Recalls at a Threshold of 0.25

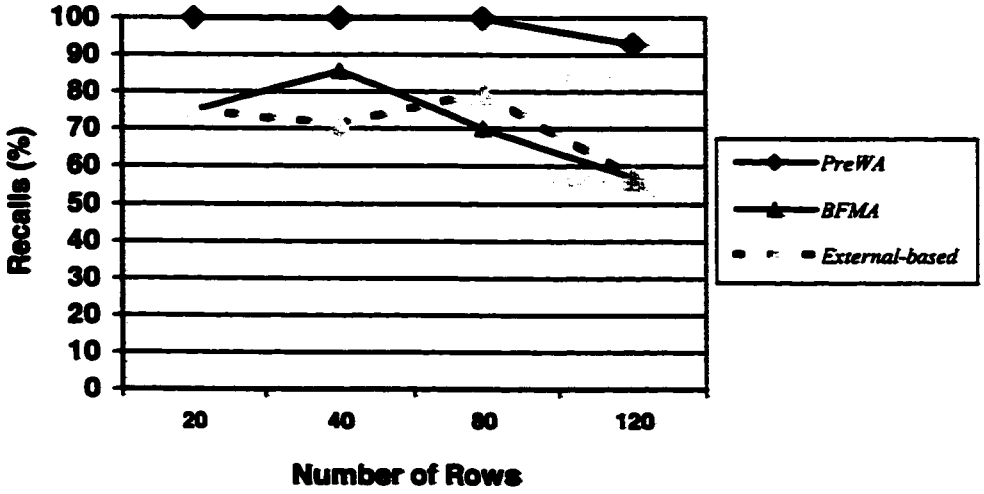

Figure 4.2: Recalls at a Threshold of 0.44

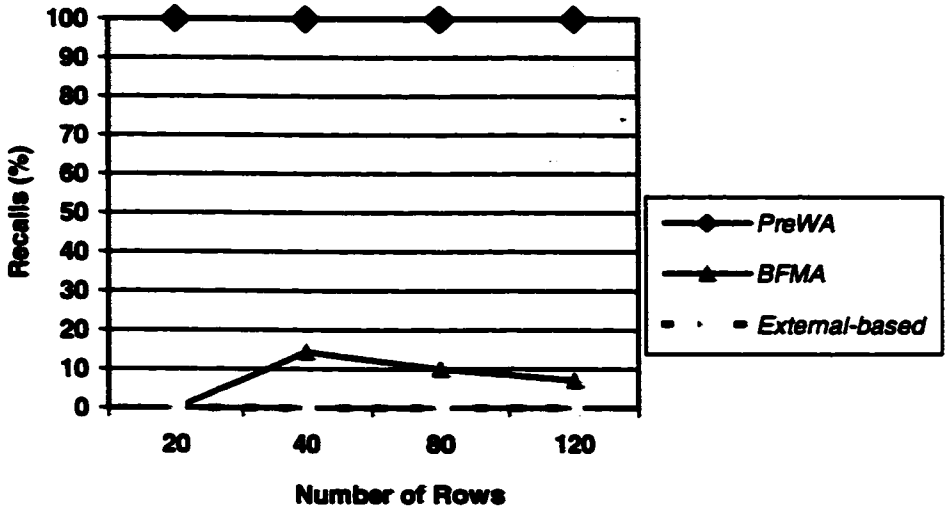

Figure 4.3: Recalls at a Threshold of 0.80

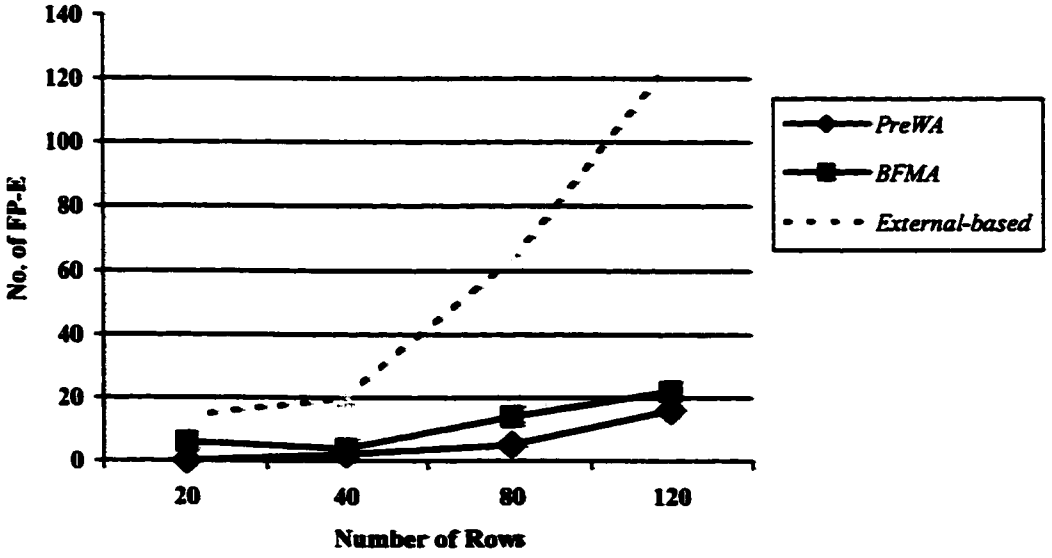

Figure 4.4: Patterns of the FP-E

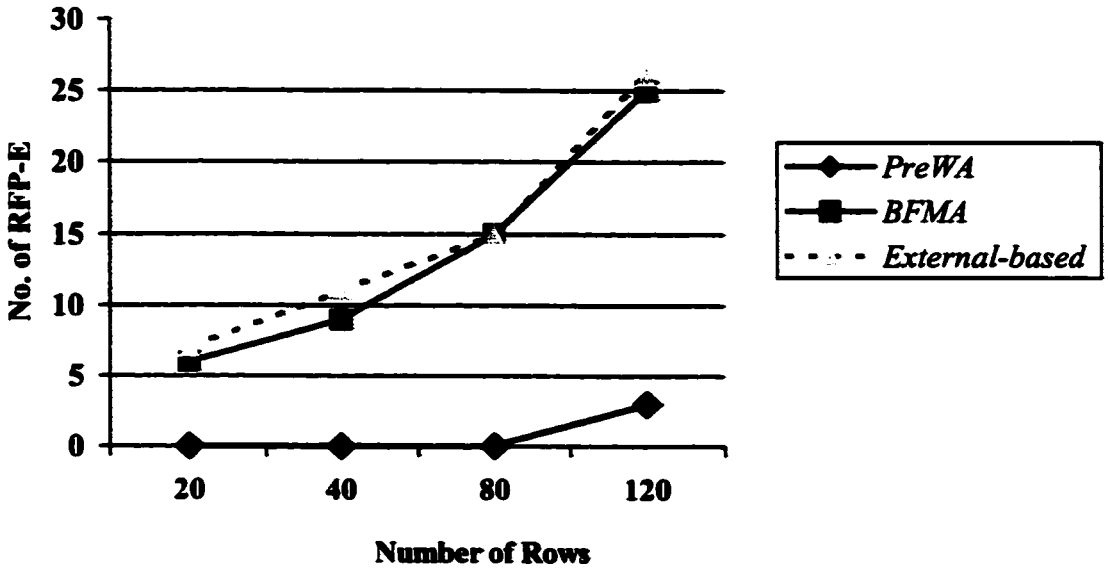

Figure 4.5: Patterns of RPF-E

Analyses of Results

It is evident from the tables given under the four experimental cases that PreWA reaches its optimal cleaning correctness at a threshold of 0.80 at all points in the four experiments. PreWA records its worst performance at the fourth experiment when the threshold is 0.25. The other two algorithms reached their best at a threshold of 0.44. PreWA also has a better performance at a threshold of 0.44 than any of the other algorithms. The results also show that the performance of PreWA is steady over the varied thresholds. In other words, the performance behavior of PreWA very slightly fluctuates as the threshold varies. The reason for this is that PreWA uses both an "exact" and a "similarity" match technique in record matching and the "similarity" match is only invoked at a "maybe match" scenario. This is not the case with the other two algorithms, which use a similarity match technique at all points, hence their performances depend so much on the threshold.

4.4 The Limitations of PreWA and PosWA

Our algorithms will fail to detect duplicates if their values at the token key fields used to sort the token key table differ.

CHAPTER 5
Conclusions and Future Research

5.1 Conclusions

Two data cleaning algorithms are described in this thesis. The first is designed to clean data the first time a data warehouse is built. The idea of using the token keys for record comparisons was introduced. Previous algorithms primarily use the keys extracted from the most discriminating fields of records for sorting and clustering purposes. The second algorithm is designed for subsequent or incremental cleaning of an existing data warehouse in a timely manner. The results of the experimental runs show (1) that the performance of PreWA does not depend on threshold, and (2) PreWA achieves the desired cleaning correctness in a good time. PosWA also achieved a 100% recall and 0% false positive error.

Conclusively, PreWA, in comparison to the Basic Field Matching Algorithm [ME96] and the algorithm in [LH99] has (1) a recall, which is very close to 100%, (2) negligible false positive and reverse false positive errors, and (3) a good response time. It also reduces the number of tables obtained from the token key table to a constant of 2 no matter the number of fields selected for cleaning, thereby greatly improving the algorithms in [HS95, HS98], where the number of extracted token key tables varies proportionally to the number of fields used for data cleaning.

5.2 Future Work

The present work has to do with the cleaning of data stored in a relational database system, which is highly structured. In our next work, we shall design a ubiquitous data cleaning algorithm that will be good for structured, semi-structured and unstructured data.

References

[AK95] S. Agarwal, A.M Keller, G. Wiederhold and K. Saraswai, *Flexible Relation: An Approach for Integrating Data from Multiple, possibly Inconsistent Databases*, In Proceedings of the 12th International Conference on Data Engineering, Taipei, Taiwan, 1995.

[AC93] Y. Arens, C.Y. Chee, C.N. Hsu and C.A. Knoblock, *Retrieving and Integrating Data from Multiple Information Sources*, Int'l Journal of Intelligent and Cooperative Information Systems, Vol. 2, No. 2, pp 127 – 158, June 1993.

[BD83] D. Bitton and D.J Dewitt, *Duplicate Record Elimination in Large Data Files*, ACM Transactions on Database Systems, Vol. 8, No. 2, pp 255 – 265, June 1983.

[Be91] E. Bertino, *Integration of Heterogeneous Data Repositories by Using Object-Oriented Views*, Proceedings of 1st Int'l Workshop on Interoperability in Multidatabase Systems, pp 22 – 29, 1991.

[BL86] C. Batini and M. Lenzerini, *A comparative Analysis of Methodologies for Database Schema Integration*, ACM Computing Surveys, Vol. 18, No. 4, pp 323 – 363, December 1986.

[BO86] Y. Breitbart, P.L Olson and G.R Thompson, *Database Integration in a Distributed Heterogeneous Database System*, Proceedings of 2nd Int'l IEEE Conference on Data Engineering, Los Angeles, February 1986.

[CD00] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, *Data Integration in Data Warehousing*, Int'l Journal of Cooperative Information Systems, 2000.

[CS91] A. Chatterjee and A. Segev, *Data Manipulation in Heterogeneous Databases*, SIGMOD RECORD, Vol. 20, No. 4, December 1991.

[CD97] S. Chaudhuri, U. Dayal, *An overview of data warehousing and OLAP technology* in: SIGMOD Record, Vol. 26 No. 1 March 1997.

[Ch98] A.I. Chen, *Integrating information from multiple independently developed data sources*, Proceedings of the 1998 ACM CIKM, seventh International conference on information and knowledge management, November 3 – 7 1998, Bethesda Maryland USA.

[De97] B. Delvin, *Data warehouse from architecture to implementation*, Addison-Wesley, 1997.

[De89] L.G DeMichiel, *Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains*, Proceedings of the 6th Int'l Conference on Data Engineering, 1989. IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 4, pp 485 – 493, December 1989.

[Ez01] C.I. Ezeife, *"Selecting and Materializing Horizontally Partitioned Warehouse Views"*, Elsevier journal of Data and Knowledge Engineering, Vol. 36, No. 2, pp. 185-210, 2001. Available at URL: http://www.cs.uwindsor.ca/users/c/cezeife/else36.pdf

[FS69] I.P Fellegi and AB Sunter, *A Theory for Record Linkage*, Journal of the American Statistical Association, Vol. 64, Issue 328, pp 1183 – 1210, December 1969.

[GF00] H. Galhardas, D. Florescu and D. Shasha, *An Extensible Framework for Data* Cleaning, In Proceedings of the International Conference on Data Engineering (ICDE), San Diego, CA 2000.

[GF01a] H. Galhardas, D. Florescu and et al., *Declarative Data Cleaning: Language, Model, and Algorithms*, Extended version of the VLDB'01 paper 2001.

[GF01b] H. Galhardas, D. Florescu and D. Shasha, *Improving Data Cleaning Quality using a Data Lineage Facility*, In Workshop on Design and Management of Data Warehouse (DMDW), Interlaken, Switzerland, Jun 01

[Gu89] A. Gupta, *Integration of Information Systems: Bridging Heterogeneous Databases.* IEEE Press, pp 2 – 5, 1989.

[GM95] A Gupta and I.S Mumick, *Maintenance of materialized views: problems, techniques and applications*, IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2), pp 3 – 18, June 1995.

[Ha00] D. Hackney, *Data warehouse delivery: CRM prerequisite #1: a data warehouse system*, DM Review, September 2000.

[HS95a] M.A Hernandez and S.J Stolfo, *The Merge/Purge Problem for Large Databases*, In Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, pp 127 – 138, May 1995.

[HS95b] M.A Hernandez and S.J Stolfo, *A generalization of Band Joins and the Merge/Purge Problem*, Technical Report CUCS-005-1995. Department of Computer Science, Columbia University, February 1995.

[HS98] M.A Hernandez and S.J Stolfo, *Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem*, Data Mining and Knowledge Discovery, 2, 9 – 37 1998.

[In96] W.H Inmon. *Building the Data Warehouse*, second edition, John Wiley, 1996.

[KS91] W. Kim and J. Seo, *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*. IEEE Computer, 24(12), pp 12 – 18 1991.

[Ki96a] R. Kimball, *The data warehousing tool kit: practical techniques for building dimensional data warehouse*, Wiley, New York, 1996.

[Ki96b] R. Kimball, *Dealing with Dirty Data*. DBMS Online, vol. 9, no. 10, September 1996. Available at URL http://www.dbmsmag.com/9609d14.htm

[LF97a] H. Lu, W. Fan, G.H Cheng, S.E Madnick and D.W Cheung, *Discovering and Reconciling Value Conflicts for Data Integration*, Proceedings of the 7th IFIP 2.6 Working Conference on Data Semantics (DS-7) 1997.

[LF97b] H. Lu, W. Fan, G.H Cheng, S.E Madnick and D.W Cheung, *Discovering and Reconciling Semantic Conflicts: A data mining perspective*. DS-7, 1997: Leysin, Switzerland.

[LG96] W.J. Labio and H. Garcia-Molina, *Efficient snapshot differential algorithms for data warehousing*, Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India 1996.

[LH99] M.L. Lee, L. Hongjun, W.L Tok and T.K Yee, *Cleansing Data for Mining and Warehousing*, In Proceedings of the 10th Int'l Conference on Database and Expert Systems Applications (DEXA 99), Florence, Italy, August 1999.

[LT95] M.L. Lee and W.L. Tok, *Resolving Structural Conflicts in the Integration of Entity-Relationship Schemas*, Proceedings of 14th Int'l Conference on Object-Oriented and Entity-Relationship Modeling (OOER95) 1995.

[LT97] M.L Lee and W.L. Tok, *Resolving Constraint Conflicts in the Integration of Entity-Relationship Schemas*, Proceedings of 16th Int'l Conference on Conceptual Modeling 1997.

[LT00] M.L. Lee and W.L Tok and W.L Low, *IntelliClean: A Knowledge-Based Intelligent Data Cleaner*, ACM SIGKDD 2000 Boston MA USA.

[Le96] A.Y Levy, *Obtaining Complete answers from Incomplete Databases*, Proceedings of 22nd Int'l Conference on VLDB, pp 402 – 412, 1996.

[LA86] W. Litwin and A. Abdellatif, *Multidatabase Interoperability*, IEEE Computer, Vol. 19, No. 12, pp 10 – 18, December 1986.

[Ma96] R. Mattison, *Data Warehousing: Strategies, Technologies and Techniques*, McGraw-Hill, page 5, 1996.

[Ma99] A. Maydanchil, *Challenges of efficient data cleansing*, DM Direct, September 1999.

[ME96] A.E Monge and C.P Elkan, *The Field Matching Problems: Algorithms and Applications*, Proceedings of the 2nd Int'l Conference on Knowledge and Data Mining pp 267 – 270, 1996.

[Mo97] A. Monge, *Adaptive detection of approximately duplicate database records and the database integration approach to information discovery*, PhD thesis, Department of Computer Science and Engineering, University of California, San Diego, 1997.

[Mo00] A.E. Monge, *Matching algorithms within a duplicate detection system*. IEEE 2000.

[Mo98] L. Moss, *Data Cleansing: A Dichotomy of data Warehousing*, DM Review, February 1998.

[Ne98] H. Neville, *Toxic Data*, DM Review, June 1998.

[NK59] H.B Newcombe, J.M. Kennedy, S.J. Axford, and A.P. James, *Automatic linkage of vital records*, Science, 130:954 – 959, October 1959.

[NK62] H.B Newcombe and J.M Kennedy, *Record Linkage*, Communications of the Association for Computing Machinery, 5 (1962), 563 – 566.

[PA96] Y. Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina, *Object Fusion in Mediator Systems*, Proceedings of VLDB conference, 1996. Available at URL: http://citeseer.nj.nec.com/cache/papers2/cs/1719/http:zSzzSzwww-db.stanford.eduzSzpubzSzpaperszSzfusion.pdf/papakonstantinou96object.pdf

[PS98] C. Parent and S. Spaccapietra, *Issues and Approaches of Database Integration*, The Communications of the ACM (CACM), Vol. 41, No. 5, 1998, pp 166 – 178. Available at URL: http://www.acm.org/pubs/articles/journal/cacm/1998-41-5e/p166-parent/p166-parent.pdf

[RH00] V. Raman and J.M. Hellerstein, *An Interactive Framework for Data Cleaning*, SIGMOD 2000 paper.

[SF97] T. Sakaguchi and M.N. Frolick, *A review of the data warehousing literature*, Journal of Data Warehousing, 2(1), 1997, 34-54. Available at URL: http://www.nku.edu/~sakaguch/dw-web.htm

[SG95] T.E. Senator, H.G. Goldberg, J. Wooton and M.A.C. et al., *The financial crimes enforcement network AI system (FAIS): identifying potential money laundering from reports of large cash transactions*, AI Magazine, vol. 16, no. 4, pp. 21 – 39, 1995.

[SB91] A. Sheth and Bellcore, *Semantics Issues in Multidatabase Systems*. SIGMOD RECORD, Vol. 20, No. 4, December 1991.

[SL95] E. Simoudis, B. Livezey and R. Kerber, *Using Recon for Data Cleaning*, In Proceedings of KDD 1995, pp 282-287.

[Su94] V.S Subrahmanian, *Amalgamating Knowledge Bases*, ACM Transactions on Database Systems, Vol. 19, No. 2, pp 291 – 331 June 1994.

[TW84] I. Tomasz and L. Witold Jr, *Incomplete Information in Relational Databases*, Journal of the Association of Computing Machinery, Vol. 31, No. 4, pp 761 – 791 October 1984.

[TL99] S.M Trisolini, M. Lenzerini and D. Nardi, *Data Integration and Warehousing in Telecom Italia*, In Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, 1999.

[Ul91] J.D Ullman, *Information Integration Using Logical Views*, In Proceedings of ICDT'97, Vol. 1186 of LNCS pp 19 – 40, Springer-Verlag 1997.

[WM89] Y.R Wang and S.E Madnick, *The Inter-Database Instance Identification Problem in Integration Autonomous Systems*, In Proceedings of the fifth Int'l Conference on Data Engineering, Los Angeles, California, pp 46 – 55, February 1989.

[YG73] M.I. Yampolskii and A.E. Gorbonsov, *Detection of duplicate secondary documents*, Nauchno-technical information, vol. 1, no. 8, pp. 3 – 6, 1973.

[ZH95] G. Zhou, R. Hull R. King and J. Franchitti, *Data Integration and Warehousing Using H2O*, IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, Vol. 18, No. 2, pp 29 – 40, June 1995.

Appendix A: Sample Outputs From PreWA

The Fact Table Before PreWA

WID	TransType	AccType	TransTime	AmountInTrans
1001	D	CA	711A	245.5
1005	W	CA	911P	500
S002	D	SA	1020P	120
S001	W	SA	555A	300
1003	W	CA	455P	250.4
1004	D	CA	1099P	350.5
S010	W	SA	651A	100
S009	D	SA	655P	150
1009	W	CA	601P	55.55
1007	W	CA	497A	50
1008	D	CA	447A	120.5
1008	W	CA	532P	50.5
S001	D	SA	877P	1000
S006	D	SA	788A	550
S003	D	SA	989A	600
S003	W	SA	998P	150
1007	D	CA	1001P	750.5

The Fact Table after PreWA

WID	TransType	AccType	TransTime	AmountInTrans
122570JOS	D	CA	711A	245.5
1979EOT	W	CA	911P	500
1175EOT	D	SA	1020P	120
122570JOS	W	SA	555A	300
111172ADD	W	CA	455P	250.4
1175EOT	D	CA	1099P	350.5
72356FS	W	SA	651A	100
42074DJO	D	SA	655P	150
31687EK	W	CA	601P	55.55
123182RR	W	CA	497A	50
21580EK	D	CA	447A	120.5
21580EK	W	CA	532P	50.5
122570JOS	D	SA	877P	1000
101275CO	D	SA	788A	550
8864CJ	D	SA	989A	600
8864CJ	W	SA	998P	150
123182RR	D	CA	1001P	750.5

The Customer Dimension Table before PreWA (Input)

CID	CNAME	SEX	CPHONE	CBIRTHDATE	CADDRESS
S001	John Smith O	M	519-111-1234	25-Dec-70	Sunset # 995, N9B3P4
S002	Tim E Ohanekwu	M	5603626	01-Jan-75	Randoph St. No. 685 n9b 2t7
S003	Colette Johnen	M	123-4567	08-Aug-64	600 XYZ apt 5a5 N7C4K4
S004	Diana D Ambrosion	F	5196669999	11-Nov-72	Church Rd. # 4 N8K6T6
S005	John Smith	M	(519)-560-3626	30-Oct-78	182 Jesus Ave M9B3J7
S006	Ogunbiyi Clement	M	519-9856488	October,12,1975	# 182 Josephine Windsor N9B 2K8
S007	Udechukwu Ajimobi	M	254-9851	14-9-1973	Electa Hall, Rm. 223M Patricia Road N9B 3P4 Windsor
S008	Sachwani Lubna	F	519) 258-8272	19/May/75	Rm. 213A, Electa Hall Patricia Rd. Widsor N9 34P
S009	Duru Juliet Oby	F	519-566-7890	20/April/74	Josephine Ave No. 18 Windsor M9C 4Y9
S010	Smith Florence	F	677-8990	23-Jul-1956	2423 Northwood Str. Windsor K9R4H6
1001	S. John	M	1111234	25-12-1970	995 Sunset Ave n9b 3p4
1002	Jon Collete	M		08-08-1964	XYZ, No. 600 apt 585 n7c 4k4
1003	D Diana Ambros	F	566-5555	10-11-1972	Church St. # 40, n8k 6t6
1004	Ohans E. Timothy	M	519-256-6416	01/01/1975	No. 695 Randolph Avenue Windsor Ontario N9B2T7
1005	Tim Emeka Obi	M	9856298	9-1-79	666 Rankin M9C 4Y8 Windsor ON
1006	Om Khan	F	971-0371	Dec, 20, 1982	530 Janette Close Windsor J2G5S9
1007	Ross Robert	M	591-948-1565	31-Dec-82	1518 StLuke Str. Widsor A8V 9G1
1008	Kenneth Ewans	M	519-688-9008	Feb/15/80	1023 Watson Rd. K8B 5O9
1009	Evans Kim	F	977-5950	16-Mar-1987	Watson Avenue No. 1020 Windsor K9C 1P9
1010	Adams Chris	M	591-978-3616	03/Jan/1981	Manchester Rd. # 685 Windsor L8B 5T6

The Customer Dimension Table After PreWA

CID	CNAME	SEX	CPHONE	CBIRTHDATE	CADDRESS
122570JOS	John Smith O	M	519-111-1234	25-Dec-70	Sunset # 995, N9B3P4
1175EOT	Tim E Ohanekwu	M	5603626	01-Jan-75	Randoph St. No. 685 n9b 2t7
8864CJ	Colette Johnen	M	123-4567	08-Aug-64	600 XYZ apt 5a5 N7C4K4
111172ADD	Diana D Ambrosion	F	5196669999	11-Nov-72	Church Rd. # 4 N8K6T6
103078JS	John Smith	M	(519)-560-3626	30-Oct-78	182 Jesus Ave M9B3J7
101275CO	Ogunbiyi Clement	M	519-9856488	October,12,1975	# 182 Josephine Windsor N9B 2K8
91473AU	Udechukwu Ajimobi	M	254-9851	14-9-1973	Electa Hall, Rm. 223M Patricia Road N9B 3P4 Windsor
51975LS	Sachwani Lubna	F	519) 258-8272	19/May/75	Rm. 213A, Electa Hall Patricia Rd. Widsor N9 34P
42074DJO	Duru Juliet Oby	F	519-566-7890	20/April/74	Josephine Ave No. 18 Windsor M9C 4Y9
72356FS	Smith Florence	F	677-8990	23-Jul-1956	2423 Northwood Str. Windsor K9R4H6
1979EOT	Tim Emeka Obi	M	9856298	9-1-79	666 Rankin M9C 4Y8 Windsor ON
122082KO	Om Khan	F	971-0371	Dec, 20, 1982	530 Janette Close Windsor J2G5S9
123182RR	Ross Robert	M	591-948-1565	31-Dec-82	1518 StLuke Str. Widsor A8V 9G1
21580EK	Kenneth Ewans	M	519-688-9008	Feb/15/80	1023 Watson Rd. K8B 5O9
31687EK	Evans Kim	F	977-5950	16-Mar-1987	Watson Avenue No. 1020 Windsor K9C 1P9
1381AC	Adams Chris	M	591-978-3616	03/Jan/1981	Manchester Rd. # 685 Windsor L8B 5T6

Record CIDs	Their WID	Their Row Nos.
S001, 1001	122570JOS	1,11
S002, 1004	1175EOT	2,14
S003, 1002	8864CJ	3,12
S004, 1003	111172ADD	4,13

Number of Pairs of Duplicates Detected is: 4

Cleaning Process Completed. Please, see WIDTABLE, CleanedCustomersTable and TheFactTable for the cleaning results.

The Comprehensive Warehouse Log Table (WIDTABLE) Generated After PreWA

CID	NAMEKEY	BIRTHKEY	ADDRESSKEY	WID
S005	JS	103078	937182JMB	103078JS
S006	CO	101275	928182KNB	101278CO
S007	AU	91473	934223MNBP	91473AU
S008	LS	51975	934213ANP	51975LS
S009	DJO	42074	91849MCY	42074DJO
S010	FS	72356	9462423HKR	72356FS
1005	EOT	1979	948666MCY	1979EOT
1006	KO	122082	259530JGS	122082KO
1007	RR	123182	8911518AVG	123182RR
1008	EK	21580	8591023KBO	21580EK
1009	EK	31687	9191020KCP	31687EK
1010	AC	1381	856685LBT	1381AC
S001	JOS	122570	934995NBP	122570JOS
1001	JS	122570	934995NBP	122570JOS
S002	EOT	1175	927685NBT	1175EOT
1004	EOT	1175	927695NBT	1175EOT
S003	CJ	8864	74455600AKNC	8864CJ
1002	CJ	8864	744585600KNC	8864CJ
S004	ADD	111172	866NKT	111172ADD
1003	ADD	101172	84066NKT	111172ADD

Appendix B: Sample Outputs From PosWA

Input: Personal Information Source to Refresh/Expand the Warehouse With

CID	CNAME	SEX	BIRTHDATE	PHONE	ADDRESS
CD001	Emenike T Oha	M	January 1 1975	519-2566416	696 abc avenue, N9B 2T7
CD003	Florence Joyce Johns	F	5-Aug-77	566-4085	1223 Windsor ave M1J 8H4
S001	Tim E Ohanekwu	M	01-Jan-75	2566416	ABCD St. No. 695 n9b 2t7
1005	Elema Tom Obi	M	23-Mar-1967	9856298	98 Jesus Rd. M8C 8S4
S011	Angela Rose M	F	Dec-25-78	253-3003	490 Rand N9B2T6 Windsor
1011	Favors Mandela	F	Jan-3-73	416-234-5678	Railway St. Toronto G7V1R5
CD002	Ray Uchenna O	M	June-04-1966	254-9853	# 2 XY Rd. H8U 2F4
1001	S. John	M	25-12-1970	1111234	995 Sunset Ave n9b 3p4
S012	Michael Rocky	M	16-12-53	971-6588	Mill St. 788 Nj92x2

Input: Transactional Data to Refresh/Expand the Warehouse With

CID	SOURCE	TRANSTYPE	AMOUNT	TRANSTIME
CD001	CD	D	500.99	889A
CD003	CD	D	150.00	651P
S001	SA	D	200.50	567A
CD001	CD	W	145.99	900P
S001	SA	W	100.50	657P
1005	CA	D	123.45	182A
1005	CA	W	223.00	482P
CD003	CD	W	50.00	659P
CD001	CD	W	100.99	809P
S011	SA	D	500.00	431P
1011	CA	D	550.00	512A
CD002	CD	D	450.50	774P
1001	CA	D	150.50	345P
S012	SA	D	560.00	423A

The Fact Table before PosWA

WID	TransType	AccType	TransTime	AmountInTrans
122570JOS	D	CA	711A	245.5
1979EOT	W	CA	911P	500
1175EOT	D	SA	1020P	120
122570JOS	W	SA	555A	300
111172ADD	W	CA	455P	250.4
1175EOT	D	CA	1099P	350.5
72356FS	W	SA	651A	100
42074DJO	D	SA	655P	150
31687EK	W	CA	601P	55.55
123182RR	W	CA	497A	50
21580EK	D	CA	447A	120.5
21580EK	W	CA	532P	50.5
122570JOS	D	SA	877P	1000
101275CO	D	SA	788A	550
8864CJ	D	SA	989A	600
8864CJ	W	SA	998P	150
123182RR	D	CA	1001P	750.5

The Warehouse Log Table before PosWA

CID	NAMEKEY	BIRTHKEY	ADDRESSKEY	WID
S005	JS	103078	937182JMB	103078JS
S006	CO	101275	928182KNB	101278CO
S007	AU	91473	934223MNBP	91473AU
S008	LS	51975	934213ANP	51975LS
S009	DJO	42074	91849MCY	42074DJO
S010	FS	72356	9462423HKR	72356FS
1005	EOT	1979	948666MCY	1979EOT
1006	KO	122082	259530JGS	122082KO
1007	RR	123182	8911518AVG	123182RR
1008	EK	21580	8591023KBO	21580EK
1009	EK	31687	9191020KCP	31687EK
1010	AC	1381	856685LBT	1381AC
S001	JOS	122570	934995NBP	122570JOS
1001	JS	122570	934995NBP	122570JOS
S002	EOT	1175	927685NBT	1175EOT
1004	EOT	1175	927695NBT	1175EOT
S003	CJ	8864	74455600AKNC	8864CJ
1002	CJ	8864	744585600KNC	8864CJ
S004	ADD	111172	866NKT	111172ADD
1003	ADD	101172	84066NKT	111172ADD

The Customers Dimension Table Before PosWA

CID	CNAME	SEX	CPHONE	CBIRTHDATE	CADDRESS
122570JOS	John Smith O	M	519-111-1234	25-Dec-70	Sunset # 995, N9B3P4
1175EOT	Tim E Ohanekwu	M	5603626	01-Jan-75	Randoph St. No. 685 n9b 2t7
8864CJ	Colette Johnen	M	123-4567	08-Aug-64	600 XYZ apt 5a5 N7C4K4
111172ADD	Diana D Ambrosion	F	5196669999	11-Nov-72	Church Rd. # 4 N8K6T6
103078JS	John Smith	M	(519)-560-3626	30-Oct-78	182 Jesus Ave M9B3J7
101275CO	Ogunbiyi Clement	M	519-9856488	October,12,1975	# 182 Josephine Windsor N9B 2K8
91473AU	Udechukwu Ajimobi	M	254-9851	14-9-1973	Electa Hall, Rm. 223M Patricia Road N9B 3P4 Windsor
51975LS	Sachwani Lubna	F	519) 258-8272	19/May/75	Rm. 213A, Electa Hall Patricia Rd. Widsor N9 34P
42074DJO	Duru Juliet Oby	F	519-566-7890	20/April/74	Josephine Ave No. 18 Windsor M9C 4Y9
72356FS	Smith Florence	F	677-8990	23-Jul-1956	2423 Northwood Str. Windsor K9R4H6
1979EOT	Tim Emeka Obi	M	9856298	9-1-79	666 Rankin M9C 4Y8 Windsor ON
122082KO	Om Khan	F	971-0371	Dec, 20, 1982	530 Janette Close Windsor J2G5S9
123182RR	Ross Robert	M	591-948-1565	31-Dec-82	1518 StLuke Str. Widsor A8V 9G1
21580EK	Kenneth Ewans	M	519-688-9008	Feb/15/80	1023 Watson Rd. K8B 5O9
31687EK	Evans Kim	F	977-5950	16-Mar-1987	Watson Avenue No. 1020 Windsor K9C 1P9
1381AC	Adams Chris	M	591-978-3616	03/Jan/1981	Manchester Rd. # 685 Windsor L8B 5T6

The Fact Table After PosWA

WID	TransType	AccType	TransTime	AmountInTrans
123182RR	D	CA	1001P	750.5
122570JOS	D	CA	458P	150.5
1979EOT	D	CA	185P	123.45
1979EOT	W	CA	458P	223
1373FM	D	CA	517A	550
1175EOT	W	CD	920P	145.99
1175EOT	W	CD	899P	100.99
4666ORU	D	CD	772A	450.5
5877FJJ	D	CD	625P	150
5877FJJ	W	CD	649P	50
122570JOS	D	SA	367A	200.5
122570JOS	W	SA	665P	100.5
122578AMR	D	SA	437P	500
122570JOS	D	CA	711A	245.5
1979EOT	W	CA	911P	500
1175EOT	D	SA	1020P	120
122570JOS	W	SA	555A	300
111172ADD	W	CA	455P	250.4
1175EOT	D	CA	1099P	350.5
72356FS	W	SA	651A	100
42074DJO	D	SA	655P	150
31687EK	W	CA	601P	55.55
123182RR	W	CA	497A	50
21580EK	D	CA	447A	120.5
21580EK	W	CA	532P	50.5
122570JOS	D	SA	877P	1000
101275CO	D	SA	788A	550
8864CJ	D	SA	989A	600
121653MR	D	SA	429A	560
8864CJ	W	SA	998P	150

The Customer Dimension Table After PosWA

CID	CNAME	SEX	CPHONE	CBIRTHDATE	CADDRESS
122570JOS	John Smith O	M	519-111-1234	25-Dec-70	Sunset # 995, N9B3P4
1175EOT	Tim E Ohanekwu	M	5603626	01-Jan-75	Randoph St. No. 685 n9b 2t7
8864CJ	Colette Johnen	M	123-4567	08-Aug-64	600 XYZ apt 5a5 N7C4K4
111172ADD	Diana D Ambrosion	F	5196669999	11-Nov-72	Church Rd. # 4 N8K6T6
103078JS	John Smith	M	(519)-560-3626	30-Oct-78	182 Jesus Ave M9B3J7
101275CO	Ogunbiyi Clement	M	519-9856488	October,12,1975	# 182 Josephine Windsor N9B 2K8
91473AU	Udechukwu Ajimobi	M	254-9851	14-9-1973	Electa Hall, Rm. 223M Patricia Road N9B 3P4 Windsor
51975LS	Sachwani Lubna	F	519) 258-8272	19/May/75	Rm. 213A, Electa Hall Patricia Rd. Widsor N9 34P
42074DJO	Duru Juliet Oby	F	519-566-7890	20/April/74	Josephine Ave No. 18 Windsor M9C 4Y9
72356FS	Smith Florence	F	677-8990	23-Jul-1956	2423 Northwood Str. Windsor K9R4H6
1373FM	Favors Mandela	F	416-234-5678	Jan-3-73	3 Railway St. Toronto G7V1R5
4666ORU	Ray Uchenna O	M	254-9853	June-04-1966	# 2 XY Rd. H8U 2F4
5877FJJ	Florence Joyce Johns	F	566-4085	5-Aug-77	1223 Windsor ave M1J 8H4
122578AMR	Angela Rose M	F	253-3003	Dec-25-78	490 Rand N9B2T6 Windsor
1979EOT	Tim Emeka Obi	M	9856298	9-1-79	666 Rankin M9C 4Y8 Windsor ON
122082KO	Om Khan	F	971-0371	Dec, 20, 1982	530 Janette Close Windsor J2G5S9
123182RR	Ross Robert	M	591-948-1565	31-Dec-82	1518 StLuke Str. Widsor A8V 9G1
21580EK	Kenneth Ewans	M	519-688-9008	Feb/15/80	1023 Watson Rd. K8B 5O9
31687EK	Evans Kim	F	977-5950	16-Mar-1987	Watson Avenue No. 1020 Windsor K9C 1P9
1381AC	Adams Chris	M	591-978-3616	03/Jan/1981	Manchester Rd. # 685 Windsor L8B 5T6
121653MR	Michael Rocky	M	971-6588	16-12-53	Mill St. 788 Nj92x2

The Warehouse Log Table After PosWA

CID	ASCN	DBIRTHKEY	ADDRESSKEY	WID
S006	CO	101275	928182KNB	101275CO
S007	AU	91473	934223MNBP	91473AU
S008	LS	51975	934213ANP	51975LS
S005	JS	103078	937182JMB	103078JS
S009	DJO	42074	91849MCY	42074DJO
S010	FS	72356	9462423HKR	72356FS
1005	EOT	1979	948666MCY	1979EOT
1006	KO	122082	259530JGS	122082KO
1007	RR	123182	8911518AVG	123182RR
1008	EK	21580	8591023KBO	21580EK
1009	EK	31687	9191020KCP	31687EK
1010	AC	1381	856685LBT	1381AC
S001	JOS	122570	934995NBP	122570JOS
1001	JS	122570	934995NBP	122570JOS
S002	EOT	1175	927685NBT	1175EOT
1004	EOT	1175	927695NBT	1175EOT
S003	CJ	8864	74455600AKNC	8864CJ
1002	CJ	8864	744585600KNC	8864CJ
S004	ADD	111172	866NKT	111172ADD
1003	ADD	101172	84066NKT	111172ADD
1011	FM	1373	715GVR	1373FM
CD002	ORU	4666	824FHU	4666ORU
CD003	FJJ	5877	1841223HMJ	5877FJJ
S011	AMR	122578	926490NBT	122578AMR
S012	MR	121653	922788NJX	121653MR

Vita Auctoris

Timothy obtained his Bachelor of Science honors degree in Computer Science, University of Ibadan, Ibadan, Nigeria in 1997. He also got his Master of Information Science (MINFSC) degree from the same University in 1998. He is presently a candidate for the Master of Science degree in Computer Science, University of Windsor, Windsor Ontario Canada. Timothy may most likely start his PhD degree program in the University of Calgary, Alberta effective from 2003 academic session, where he is expected to continue his research in databases.

Apart from academics, Timothy also has a great passion for church related ministry. Before now, Timothy had put in several years in pastoral vocation and might likely combine his secular career with the pastoral calling.