A Framework for Records Management in Relational Database Systems

by

Ahmed Ayaz Ataullah

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

© Ahmed Ayaz Ataullah 2008

المنارة للاستشارات

www.manaraa.com

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

المنارة للاستشارات
www.manaraa.com

Abstract

The problem of records retention is often viewed as simply deleting records when they have outlived their purpose. However, in the world of relational databases there is no standardized notion of a business record and its retention obligations. Unlike physical documents such as forms and reports, information in databases is organized such that one item of data may be part of various legal records and consequently subject to several (and possibly conflicting) retention policies. This thesis proposes a framework for records retention in relational database systems. It presents a mechanism through which users can specify a broad range of protective and destructive data retention policies for relational records. Compared to naïve solutions for enforcing records management policies, our framework is not only significantly more efficient but it also addresses several unanswered questions about how policies can be mapped from given legal requirements to actions on relational data. The novelty in our approach is that we defined a record in a relational database as an arbitrary logical view, effectively allowing us to reduce several challenges in enforcing data retention policies to well-studied problems in database theory. We argue that our expression based approach of tracking records management obligations is not only easier for records managers to use but also far more space/time efficient compared to traditional metadata approaches discussed in the literature. The thesis concludes with a thorough examination of the limitations of the proposed framework and suggestion for future research in the area of records management for relational database management systems.

المنارة للاستشارات
www.manaraa.com

Acknowledgments

I would like to acknowledge the support of friends and family without which this thesis and the work represented therein would not have been possible.

المنارة للاستشارات
www.manaraa.com

Contents

1 Introduction **1**

1.1 Records Management . . . 1

1.1.1 Records Retention . . . 2

1.2 Policy Management . . . 3

1.2.1 Identification and Classification . . . 3

1.2.2 Policy Conflicts . . . 4

1.3 Legal Requirements and Implications . . . 5

1.4 State of the Art in Records Retention . . . 7

1.5 Problem Statement . . . 8

2 Records in Relational Database Systems **11**

2.1 Practical Implications . . . 11

2.1.1 Proposed Definition . . . 13

2.2 Temporal Records . . . 14

2.3 Physical Documents as Sets of Views . . . 15

3 Protecting Relational Records **19**

3.1 Requirements . . . 19

3.2 Protective Policy Specification . . . 20

3.3 Syntax and Examples . . . 21

3.3.1 Protection Levels . . . 22

المنارة للاستشارات
www.manaraa.com

3.3.2 Temporal Records and Protection Levels 23
3.4 Framework for Implementation 24
3.4.1 Non-Temporal views 24
3.4.2 Temporal Views 26
3.4.3 Expected Performance 27

4 Timely Destruction of Relational Records 29

4.1 Requirements 29
4.2 Flexibility and Proof of Compliance 32
4.3 Correct Enforcement 33
4.4 Weak Correctness 35
4.5 Integrity Preservation 37
4.5.1 Primary Key and Uniqueness 38
4.5.2 Foreign Keys 38
4.5.3 Other Issues in Integrity Preservation 40
4.6 Schema and Policy Set Evolution 41

5 Making it Work 43

5.1 Inter-Policy Conflicts 43
5.1.1 Detecting Conflicts 43
5.1.2 Source of Conflicts 44
5.2 Conflict Avoidance 45
5.3 Implementation 48
5.4 Experimental Evaluation 50

6 The Broader Picture in Records Retention 57

6.1 Existing Solutions 57
6.2 Guaranteed Destruction of Records 60
6.2.1 Backups and Offline Databases 60

المنارة للاستشارات
www.manaraa.com

6.2.2 Distributed Systems 61
6.2.3 Unwarranted Data Retention 62
6.3 Future Work 62
6.4 Summary of Contributions 64
A Policy Descriptions and View Definitions (DB2) 65
A.1 Policy 1 65
A.1.1 View Definition 65
A.1.2 Trigger on Orders 66
A.1.3 Trigger on Lineitem 66
A.2 Policy 2 67
A.2.1 View Definition 67
A.2.2 Trigger on Orders 68
A.2.3 Trigger on Lineitem 68
A.3 Policy 3 69
A.3.1 View Definition 69
A.3.2 Trigger on Orders 69
A.3.3 Trigger on Lineitem 70
A.4 Policy 4 70
A.4.1 View Definition 70
A.4.2 Trigger on Orders 71
A.4.3 Trigger on Lineitem 72
A.5 Policy 5 72
A.5.1 View Definition 72
A.5.2 Trigger on Orders 73
A.5.3 Trigger on Lineitem 73
A.6 Policy 6 74
A.6.1 View Definition 74

المنارة للاستشارات
www.manaraa.com

A.6.2 Trigger on Orders 74
A.6.3 Trigger on Lineitem 75
A.7 Policy 7 75
A.7.1 View Definition 75
A.7.2 Trigger on Orders 76
A.7.3 Trigger on Lineitem 76
A.8 Policy 8 77
A.8.1 View Definition 77
A.8.2 Trigger on Orders 77
A.8.3 Trigger on Lineitem 78
A.9 Policy 9 78
A.9.1 View Definition 78
A.9.2 Trigger on Orders 79
A.9.3 Trigger on Lineitem 79
A.10 Policy 10 80
A.10.1 View Definition 80
A.11 Policy 11 81
A.11.1 View Definition 81
A.12 Policy 12 81
A.12.1 View Definition 82

References **82**

المنارة للاستشارات
www.manaraa.com

المنارة للاستشارات
www.manaraa.com

Chapter 1

Introduction

1.1 Records Management

Storage of records has always been a fundamental objective of information systems. However in the past decade managing sensitive information throughout its lifecycle, from creation to destruction (or archival), has become of significant importance. The increasing awareness of the impact of technology on privacy has added momentum to the need to have better enforcement of records retention policies. Organizations today not only have to comply with regulations, but also have to maintain a balance between operational record keeping requirements, minimizing liability of storing private information, and customer privacy preferences.

There is no globally accepted definition of a record and one of the primary questions addressed in this thesis is to answer what is a record in the context of relational database systems. The ISO 15489 standard defines a record as "information created, received, and maintained as evidence and information by an organization or person, in pursuance of legal obligations or in the transaction of business." Unfortunately, like most attempts to define a record, it leaves much to the interpretation of the records manager. For example, the question of whether an organization's internal emails should be treated as business records, cannot be directly answered using this definition. It is often fundamentally impossible to mechanize the process of classifying a piece of information as a business record and to determine whether storing it will be beneficial for the business. The tendency in most organizations traditionally has been to play it safe and to lean towards a "store everything" approach.

This work will not attempt to define the term "record" in the broad context. Instead the term will be treated in all its generality and then applied to the world

المنارة للاستشارات
www.manaraa.com

of relational databases. Without attempting to differentiate terms such as data, knowledge, information and record, it is recommended that the reader maintain a simple but consistent definition of a record throughout this thesis. Since the proposed framework is heavily geared towards records generated using a typical relational database system, the examples used in the work will be those of transactional records. Examples include a sales report, an invoice, a telephone bill or a student's transcripts. Such records typically exist as physical documents, and they have a visible structure and strong correlation with the underlying relational data. Examples of non-typical transactional records include, a single number such as the total number of employees in a business or the number of employees with first name "James." Non-traditional records typically become part of other records and are quite often ignored for policy enforcement, especially when embedded in physical documents.

Records management as described by ISO 15489 is "the field of management responsible for the efficient and systematic control of the creation, receipt, maintenance, use and disposition of records, including the processes for capturing and maintaining evidence of and information about business activities and transactions in the form of records." In essence the task of a records manager is identifying sources of information where records are created and then managing them throughout their lifecycle. Typical duties of a records manager include determining and enforcing policies on records, such as access control, archiving and destruction.

1.1.1 Records Retention

Recent trends in privacy and data management have led to a retention paradigm different from the store-everything approach. Most organizations have realized that there can often be significant costs associated with storage of information. The protection of customers' personal and financial information, for example, is a duty and a potential source of liability for online businesses. As the size of a corporate record repository grows, so does the cost of an accidental information leak. Furthermore the general increase in privacy awareness is leading modern consumers to demand that their private information not be retained indefinitely by organizations. Consequently many businesses have voluntarily enforced limited retention practices where sensitive records are destroyed when the cost of retaining them outweighs the benefits.

Equally concerning are the plethora of legal requirements that businesses face today. Globally, the number of regulatory requirements mandating businesses to

المنارة للاستشارات
www.manaraa.com

retain records for minimum periods have increased significantly. For example after the Enron scandal, the Sarbanes-Oxley Act in the United States has mandated that audit records of businesses be maintained for at least five years [MS02]. Similarly, in the wake of the Madrid and London bombings, the EU Directive on Data Retention has explicitly mandated strict record keeping requirements for businesses such as telecommunications companies and Internet Service Providers [DFK06].

Many businesses today faces the unique challenge of finding the correct balance between legal compliance, operational record keeping, and satisfying consumer privacy preferences. Destroying records too early can constitute a criminal offense and retaining them for too long can become a liability or constitute a violation of published policy.

1.2 Policy Management

1.2.1 Identification and Classification

There are several fundamental problems in retention policy management that make records management a difficult task. However, the most critical requirements for proper records retention are identification and classification of records.

Record identification is the task of determining which information or event needs to be documented and treated as a significant business record. The legal definition of a record, or an event leading to information being recognized as a record, is usually unclear and varies depending on the situation. As an example, consider a scenario where an employee is notifying his employer regarding a workplace safety hazard. Such first notices are recognized as critical legal records, and they are often used for litigation purposes. These records also have legislated minimum retention periods, regardless of the medium in which they are presented, which largely puts the burden of records identification and preservation on the corporation. In this case, the employee has the right to mention workplace safety related issues in an informal conversation with his supervisor, in a meeting, as a formally written complaint, as an email, or as part of a report. It is important to recognize that from a purely legal perspective, in many situations no physical document needs to be created for a record to be realized. Visible records with structure and purpose such as reports, invoices and requisition forms are relatively easy to identify. However, if a typical personal email or a daily status report contains a special notice about workplace safety, it needs to be treated very differently from other similar types

المنارة للاستشارات
www.manaraa.com

of communication. Consequently most businesses train and require employees to identify issues and to observe the relevant record keeping obligations related to them, so that the corporate records management policy can function effectively.

Since it is infeasible to design an individual retention policy for every record, policies are typically specified for types or classes of records. Once a particular record is identified the next step is to classify it into one of the pre-defined classes to ensure the correct retention policies are enforced on it. Although the task of classification is significantly easier when compared to identification, it leads to potential cross-classification conflicts as we begin to integrate policies originating from different areas of a business.

1.2.2 Policy Conflicts

Formulating records management policies within a particular department may be challenging, but integrating records and policies that involve several functional areas of an organization can be a significantly more complicated task. Various departments can use the same records differently and may specify retention policies without considering the company-wide implications. A retention conflict is essentially a conflict between two different actions (deletion and protection) required on a particular record. The primary source of conflicts are varying minimum and maximum retention periods among different types of business records. For example, a typical scenario in businesses is that of the human resource department warranting that employment details be deleted a fixed time after an employee leaves the company. An employment record on which such policy is defined may contain a substantial amount of detailed information such as the social insurance number, positions held, pay-cheques issued and taxes paid on behalf of the employee. Some of this information, such as taxation, may have its own independently legislated retention obligations, and other parts, such as salary paid, may be required to persist indefinitely by other departments of the organization.

The task of a records manager in such situations is to mediate policy requirements between various parts of the business. A conflict of policy without a predetermined mechanism for resolution represents a situation where user intervention is required. It is also worth noting that in most business scenarios, due to the complexity of business processes and ill-defined nature of records, it is almost impossible to enumerate all sources of policy conflicts.

المنارة للاستشارات
www.manaraa.com

1.3 Legal Requirements and Implications

Since there are no clear cut definitions of records over disparate sources of data, the ability to specify and enforce retention policies without ambiguity is severely diminished. Typically most legal requirements have to be interpreted by records managers and applied to each situation independently of other constraints. An interesting example of a vague definition, taken from the United States Health Insurance Portability and Accountability Act (HIPAA 2002), is that of "uniquely identifiable information." HIPAA mandates that any exchange of health related data of patients among organizations for non-medical purposes (such as research and statistical analysis) must adhere to principles of privacy and not disclose information that can lead to the unique identification of a patient. Unfortunately the question of which pieces of information can lead to unique identification of an individual is not addressed in the law. As pointed out by Sweeney in her work on k-anonymity [Swe02], the concepts of candidate and primary keys are largely inapplicable in this case. She noticed that the 3-tuple {ZIP, gender, date of birth}, although strictly not a candidate key, can easily and uniquely identify 87% of Americans. Similarly for different combinations of given information, such as ethnicities, age ranges and street addresses, it may be possible to identify many individuals precisely. The law however, puts the burden of defining and assessing which information (combination of tuples) can lead to unique identification of *every individual* on the record keeper. Needless to say that implementing mechanisms to comply with such regulations is an extremely challenging task.

Some industries that are legally mandated to demonstrate effective retention policies in their record keeping practices include health-care, financial management (banks, credit issuing organizations and auditing/accounting firms), insurance, telecommunication and Internet services providers. These retention laws can be independently specified at the federal level, at the level of individual states and provinces and possibly (but rarely) at the level of local governing bodies. It is interesting to note that records can be subject to a variety of temporal and non-temporal conditions. A typical example in health care is that of the Florida Administrative Code for medical facilities with a pediatric program [ML03]: if a minor is treated in a medical facility that is subject to this code, then all medical records (diagnostic reports and treatments given) must be maintained until three years after the patient reaches the age of majority as specified in the state law. In 2002, with the introduction of the Sarbanes-Oxley Act, a much broader umbrella of record keeping requirements were put on all public companies in the United States. These

المنارة للاستشارات
www.manaraa.com

requirements mostly dealt with establishing mandatory minimum retention periods for corporate financial (primarily auditing related) records. The granularity of data on which retention periods are defined is still very vague and can be interpreted to go as far as mandating minimum retention periods for invoices, individual emails, and even voice-mail messages that may be significantly related to the financial operations of an organization. The rule of thumb for Sarbanes-Oxley compliance is that any information that is significant for financial auditing must be preserved (retained and protected from being tampered) for at least 7 years. Organizations facing these ambiguously defined record keeping requirements can not only be asked to present their "records" for examination but can also be faced with fines if they fail to satisfy an auditor of records that they have adequately managed their corporate records.

In many record keeping situations, such as law enforcement, no single piece of data can ever be truly deleted. Blanchette and Johnson point out in their examination of modern data retention practices [BJ98] that in situations such as juvenile criminal records, data retention can be a curse for many individuals throughout their life. Even though crimes can be pardoned, they are rarely deleted from criminal records: instead they are appended with a pardon-related entry. Her critical review of data retention practices in several industries leads to the conclusion that social-forgetfulness (which arguably may be beneficial for our society) is generally not supported by our rigid record keeping infrastructure.

From a different perspective data retention requirements, where not mandated legally, can be an operational business requirement. An interesting case is that of contractors of the Department of Defense in the United States (DoD). The DoD mandates that all electronic communications (especially emails) between DoD employees and any contractors must not be retained by the contractor for more than three years. Each organization that wishes to do business with the DoD must submit to a retention audit and, apart from hundreds of other security related requirements, must also demonstrate that electronic communications are properly deleted according to a transparent schedule. Recently other business have also begun implementing similar operational requirements. VISA, for example, is attempting to extract guarantees from its online credit card processing subsidiaries that warrant deletion of customers' data after a fixed and publicly available time after a credit card transaction is completed.

المنارة للاستشارات
www.manaraa.com

1.4 State of the Art in Records Retention

Although records management is a very mature field in itself, software solutions for effective management of business records have only emerged in the last two decades. In the realm of software, the term for managing business records that is widely recognized is Enterprise Content Management. Enterprise Content Management (ECM) can best be described as the use of computerized techniques for effective management of content generated through the activities of a business. The word 'content' is specifically used to encompass all forms of digital and non-digital information. ECM aims to simplify the overall task of creating, distributing and maintaining business related content, such as documents, videos, spreadsheets, presentations. Most ECM systems also attempt to aid in inter-team communications (groupware) and provide workflow management facilities. The largest players in the ECM market are OpenText with their LiveLink suite of products followed by EMC and IBM [Eid06].

The umbrella of content which current ECM solutions manage has been significantly extended to include emails, instant messages and even mobile messages between employees. Among the features offered in records management suites, the focus has been on collaboration (versioning and distribution), work flow management, and access control. Since the broader need to demonstrate records retention compliance has emerged only recently, retention is a relatively new feature in software based ECM systems. The functionality for retention offered in such systems is usually geared towards proper classification of records and then destroying them according to a certain fixed schedule.

The expressiveness of retention policies in most ECM systems is sufficient for a broad range of user requirements. Content is typically managed (classified and retained) using metadata. For example, in the LiveLink eDOCS Email Management System, company-wide emails can be automatically classified based on message metadata such as date received, sender address and receiver address. Advanced features such as classification based on regular expression matches within the email body and attachments are also available, and classification on-demand can also be performed. Retention rules, for example, deleting emails five years after creation, can be specified for classes of similar emails. ECM systems are generally expected to track client and server copies of records (emails in this case) and ensure that any retention conflict arising through cross-classification is brought to the attention of the administrator.

المنارة للاستشارات
www.manaraa.com

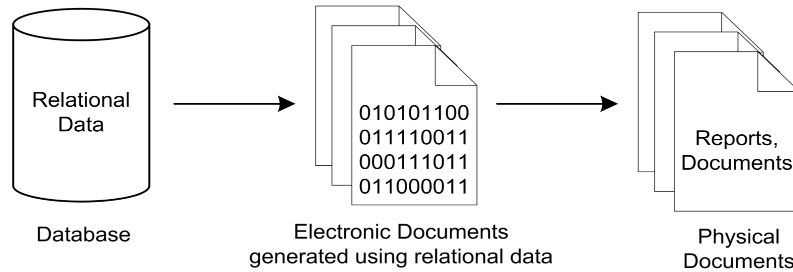

Figure 1.1: The broader picture in records management

1.5 Problem Statement

In this thesis we focus on several core problems of retention policy management for records in relational databases including identification, classification, enforcement, conflict detection and resolution. Our work is motivated by the fact that ECM systems do not consider databases as content and are therefore unable to support the enforcement of retention policies at the granularity of rows and attributes. We also note that timely destruction of documents (derived through relational data) does not necessarily imply that the information contained within those documents has been lost. Figure 1.1 depicts this disconnect between physical records and relational databases. It is obvious that shredding the relevant printed documents and deleting electronic versions of those documents is not sufficient if they can simply be recreated using the underlying data. All retention policies and obligations that are enforced on the paper or electronic versions of these documents must also be enforced in some way on the data which led to the creation of such records.

Furthermore, when we consider relational systems, there may be many documents or records that are never materialized. Reports involving relational data may be dynamically created and viewed by users, but never printed or persisted as files. However, depending on the legal circumstances, these virtual records may be subject to the same obligations as physical records.

There has not been any significant work done in the area of records retention for records encapsulated within relational databases. Since ECM systems view the database as a single object or file it is unlikely that effective retention policies can be specified and enforced using such a coarse perspective. It is stipulated that there are not many organizations that implement, monitor and maintain company wide relational record retention policies. Needless to say database systems do not have any native functionality for retention, and it has to be modeled as an elaborate

المنارة للاستشارات
www.manaraa.com

set of access control mechanisms (for enforcing minimum retention periods) and batch programs (for enforcing maximum retention periods). The problem is further exacerbated by the fact that schema evolution and introduction of new policies can pose significant overhead costs in manually updating the control mechanisms for retention.

It is obvious that the traditional ECM approach of simply attaching metadata to records containing policy requirements for data cannot be translated efficiently onto the relational world. Having a timestamp which denotes when data needs to be protected/removed, associated with every row/attribute is simply space inefficient and will most certainly not scale up for modern high performance database systems. Furthermore the issue of how these timestamps will be assigned and maintained by administrators is unclear. This thesis addresses the need for a much more efficient and systematic way of retaining records in relational database systems than a simple metadata approach. We re-examine the problem of records retention from a different perspective and present a new way of looking at records, which makes the task of record identification and classification in relational systems easy for records managers. The thesis examines efficient mechanisms for enforcing protective retention policies, such as those that mandate records be protected from unwarranted deletions, and also develops a framework for destruction of relational records as they outlive their retention period. Most importantly a formal layer of reasoning about records and detecting policy conflicts is presented. It is expected that using the proposed framework in situations involving complex and evolving database schemas will very likely minimize the cost of retention policy management.

المنارة للاستشارات
www.manaraa.com

المنارة للاستشارات
www.manaraa.com

Chapter 2

Records in Relational Database Systems

2.1 Practical Implications

From a non-technical perspective, database systems simply store data, and it is only when data is presented in a meaningful fashion that it is considered to be a record. However, when talking in terms of database concepts and attempting to correlate the commonly held notion of records with data stored in relations, it is challenging to derive a definition of a record that can be widely accepted.

As an example of the complexity involved, consider a typical university database which manages student enrollment and finances. For the university registrar, the term 'student record' will most likely refer to the entire history of courses in which the student has enrolled. However, it is unlikely that the finance department will have the same definition of a student record, as they would be more interested in the various fee payments made by the student. Similarly an instructor might want to know whether students enrolled in his class have all the relevant pre-requisites, consequently defining a record as a subset of what the registrar considers as a record. Going back to the issue of meaningful presentation of data, a single tuple representing an entity, such as a student, can be meaningful and considered to be a record. At the same time a single tuple in a many-to-many relation, such as one relating a student number to a course number indicating the enrollment of a student in a particular course, may not be meaningful in isolation (if not considered in the context of a join).

للاستشارات
www.manaraa.com

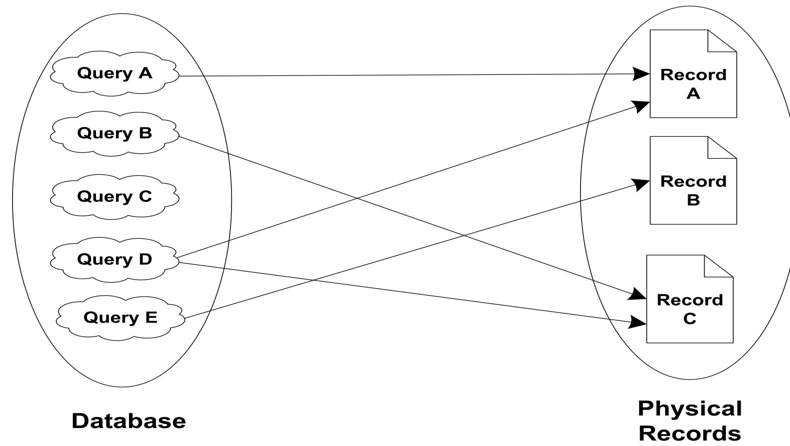

Figure 2.1: Queries and physical records are both infinite sets with possible overlapping information present between them.

There is no simple answer to what is a record in relational systems. Most of us would typically think of records as tuples or rows in a database table. Unfortunately from the examples above we can see that this perspective on a record is not only very limited but also not universally applicable. Any row in a table can be meaningless unless we associate a context with it. Similarly while one person may consider a row to be a meaningful record, another might only consider the contents of the whole table to be meaningful. To mediate such differences in interpretation, we define a record in the most simple and elegant terms, as data presented such that it holds meaning in a user's context. In essence it is the well formed and meaningful questions (queries) posed to a database system whose results are considered to be records. Each user may have different questions whose results they consider valuable records. Since the number of queries that can be posed to a database system is infinite, it follows that the number of potential records that can be generated using a database system is also infinite. Of course the result of all queries on a database system may not be meaningful from a business perspective, but there is no algorithm to determine which queries produce meaningful records. It is only the users of a database system who can decide what is a record generating query. Conversely, there can also be an infinite number of physical records that are not derived from a database system. Figure 2.1 summarizes this relationship between physical records and database queries.

The problem of identifying the particular queries and the results that are meaningful enough to be considered a business record leads to an interesting legal dilemma.

المنارة للاستشارات
www.manaraa.com

The assertion that the printed (or stored) result of every query can legally constitute a record is rather far-fetched. On the other hand, claiming that databases store data and not records, as a defense for not complying with legislated records retention policies, is also not a convincing argument. Our initial research has led to the conclusion that most organizations consider only documents that are visibly generated from database systems, such as invoices and sales reports as business records. On the other hand, recent court rulings related to legal and forensic analysis of computer equipment (as in the case of Enron) have considered every piece of available data as fair game for prosecutor examination. Interestingly, most OLAP tools and report creation wizards have the ability to create an overwhelmingly large number of reports and documents through complex manipulations of data. Whether businesses should be aware of (or liable for) the wide array of undiscovered potential records that can be created from their database systems is still unclear in the law. Unfortunately, our examination recent work related to privacy in database systems (presented in Section 6.1) has revealed that most experts think of databases as a collection of rows. However as soon as we go beyond this definition of record and include the fact that these rows can be combined and manipulated in hundreds of ways to obtain other interesting records, we run into very complex problems for data retention purposes.

2.1.1 Proposed Definition

Definition 2.1. *A relational record is a logical view specified by a relational expression over a fixed physical schema.*

Informally a relational record is the data presented as the result of an arbitrary SQL query. Note that unlike physical (static) records, the data presented in the record can change, but the record definition always stays the same. Also note that the above definition does not restrict the expressiveness of the language used to specify individual records in any way. However the advantages of doing so will duly be pointed out throughout this thesis. As an example consider record definitions to be restricted to conjunctive queries. Since conjunctive queries are well understood, we can leverage known results from database theory to help in statically analyzing different properties of records. The less restrictive the language used to specify relational records the more difficult all our management tasks will become.

Note that this definition is inherently flexible as it allows us to define a record as we wish. For example we can still take the basic approach to records being

المنارة للاستشارات
www.manaraa.com

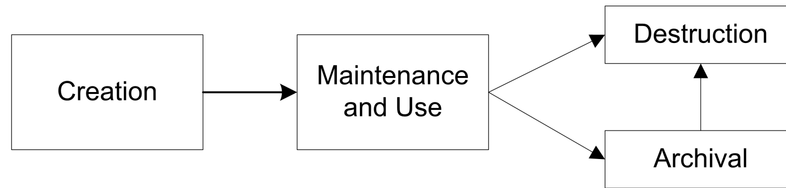

Figure 2.2: A simplified overview of Emmerson's Records Lifecycle [Emm89, UB04]

rows and attributes, and at the same time add as much complexity in operations on relational data as we wish for defining a record. Finally, it is important to emphasize the equivalence of relational records and logical views over the physical schema. Since it is only the presentation of data that makes a record, views are the ideal abstraction for data manipulated in a meaningful fashion (We discuss this point further in the light of some real world examples in Section 2.3). From this point onwards we will use the terms relational record, view and query interchangeably, as they are equivalent for our purposes.

2.2 Temporal Records

Records have a lifecycle consisting of at least three stages, namely creation, activity and destruction (or archival). For a broad ranging discussion of various proposed models of records lifecycle the reader is directed to [YC00]. This thesis asserts that, barring certain exceptions, such as temporary data which is created and deleted within the same transaction, records in relational databases have a similar lifecycle (see Figure 2.2). Unfortunately, such lifecycle models cannot explicitly specify the conditions under which a record moves between individual stages nor specify stages where a record may be protected (or subject to destruction). Depending on the sensitivity of the data, some records may achieve protective status from the moment they are created, whereas others may have long retention periods which may require them to be deleted after they have been archived into warehouses[1]. Our survey of a wide array of records retention requirements legally mandated on businesses (Sarbanes-Oxley, HIPAA etc) leads to the conclusion that a majority of these policies are defined with time-driven conditions.

Consequently, a temporal function needs to be introduced in the language used to express records, through which users can represent the current time in record

[1]This problem is discussed in greater detail in Chapter 6.

المنارة للاستشارات
www.manaraa.com

definitions and policies. This temporal function will be denoted as NOW. Using NOW, users can define records as a function of time, such as "invoices created more than six years ago." Although the use of NOW adds significant expressive power to the record definition language, there are certain complications and challenges with the use of this temporal construct. Temporally defined records are not easy to understand due to the fluid nature of time. A record that may be empty at one instance of time may contain information at the next. The opposite, of course, can also happen. Even though a database may not have changed state in any way, simply the passage of time may trigger changes in a record. Note that we are not proposing to extend the relational model to support temporal data. Instead NOW will only be used to define records and policies over traditional relational schemas and stored data such as timestamps/date-time attributes. If the existing data model (schema) cannot support the storage of temporal information and is unable to answer temporal queries then we will not be able to support retention on such records. This limitation is a direct consequence of the fact that only data which is stored physically can be retained and that the physical design must be able to accommodate data retention requirements.

2.3 Physical Documents as Sets of Views

A set of relational records can be considered a direct mapping of a traditional record onto a database schema. When considering traditional records such as invoices and sales reports in their physical form, it can be observed that many of these records have a structure, or a template, that can be directly translated onto the relational world. For example, invoices are typically structured as having uniquely identifiable information followed by the details of individual line items concluded with a summary of the invoice. Similarly, in the case of phone bills, customer information is typically followed by a list of entries signifying the calls placed/received. From a database programmer's perspective the significant difference between two phone bills is not the information contained in those records, but rather the parameters, such as customer ID and billing period, that are used to generate those records.

Consider a simplified schema for a phone company defined, where it is assumed that a customer may have more than one phone number with the company:

Customer (Customer_ID, Street_Address, City, ...)
Phones (Customer_ID, Phone_Number)
Calls (Call_ID, Origin, Destination, Duration, DateTime, ...)

المنارة للاستشارات
www.manaraa.com

In this schema the physical phone bill (or the data that goes therein) for the number 519-123-4567 for the month of July 2007, can be specified as a combination of two relational expressions:

Example 2.1

$Bill_{5191234567} = \{R_1, R_2\}$

where $R_1 =$
```
SELECT * FROM Customer, Phones
WHERE Phone_Number = '5191234567'
AND Customer.Customer_ID = Phones.Customer_ID
```
and $R_2 =$
```
SELECT * FROM Calls
WHERE Origin = '5191234567'
AND DateTime >= 'July 01, 2007' AND DateTime <= 'July 31, 2007'
```

or alternatively as a single expression:

Example 2.2

$Bill_{5191234567} = \{R_1\}$

where $R_1 =$
```
SELECT * FROM Customer, Phones, Calls
WHERE Phone_Number = '5191234567'
AND DateTime >= 'July 01, 2007' AND DateTime <= 'July 31, 2007'
AND Origin = Phone_Number
AND Customer.Customer_ID = Phones.Customer_ID
```

Note that both of the above definitions for a phone bill capture and identify the same data using conjunctive queries, but they are expressed differently. A document (physical record) can have the result of several queries embedded in it, therefore a document can be described as a collection of relational records. It can also be argued that since the first definition clearly separates entities (customer and calls placed) into different relational records, it mirrors a physical record more accurately than the second definition. There are several advantages of mapping physical documents into multiple relational records which are their visible equivalents in the physical world. These are discussed in Section 5.2.

Going further with our examples of relational records, we can also capture multiple physical records (or a whole category of physical records) together by eliminating parameters from record definitions. The following relational record which is very similar to the previous one, captures the information contained in *all* telephone bills issued for the month of July 2007:

Example 2.3

المنارة للاستشارات
www.manaraa.com

$Bills_{July2007} = \{R_1\}$
where $R_1 =$
```
SELECT * FROM Customer, Phones, Calls
WHERE Origin = Phone_Number
AND DateTime >= 'July 01, 2007' AND DateTime <= 'July 31, 2007'
AND Customer.Customer_ID = Phones.Customer_ID
```

Although it is unlikely that there will ever be a single physical document capturing the data contained in all telephone bills for the month of July 2007, records managers still should be able to declaratively define such records for the purposes of retention policy enforcement. Finally, the following is an example of a temporal record definition, which captures all the phones bills issued between three and six years ago. This record is of course non-traditional in nature (a continuously sliding window) and does not have a physical equivalent.

Example 2.4
$Bills_{VeryOld} = \{R_1\}$
where $R_1 =$
```
SELECT * FROM Customer, Phones, Calls
WHERE Origin = Phone_Number
AND Years(NOW - DateTime) <= 6
AND Years(NOW - DateTime) >= 3
AND Customer.Customer_ID = Phones.Customer_ID
```

An important observation is that record definitions are not limited in any way. The complexity of the record is only limited by the expressions used to specify the record. For example, if the marketing department of the phone company wants a report listing customers that have placed at least two calls that lasted for more than 60 minutes, from Toronto to London in a one month period, they are free to do so using their choice of declarative query language. Similarly, if government legislation requires phone companies to retain records of international calls placed to certain countries for at least 10 years, the particular records that they require can be defined using this mechanism.

Note that the examples of records given above are all in SQL syntax. The reason for choosing such presentation is only simplicity and because it makes the above examples easy to understand for the typical records manager. Any other language capable of querying a relational database, such as datalog or relational algebra, would also suffice for defining records as views. However, it is very likely that the eventual goal of any records retention system for relational databases would be to define and implement policies over records expressed using the full expressiveness

المنارة للاستشارات
www.manaraa.com

of SQL. Consequently discussing examples in the light of actual SQL queries is a suitable approach.

Defining a record gives us a handle on the data that we want to protect or destroy when the correct conditions are met. Consequently, we have chosen a very broad and flexible definition of a relational record, which can allow records managers to capture all possible records that can be generated through a database system. This definition is in stark contrast to the traditional row-based approach of looking at records and is far more expressive in the type of records it allows us to express for policy enforcement. Note that, even within the small schema used as an example, countless critical records that may have complex retention obligations on them can be easily generated. With more complex schemas such as those in large corporate databases, emergence of an unmanageable web of interrelated and derived records is a certainty. Simple questions of whether information contained in a particular record can be deleted without damaging other records or compromising the integrity of the database need to be addressed. More importantly, what does it mean to delete a record and how can users specify the conditions necessary for proper retention and deletion of arbitrary relational expressions? The next two chapters define the notion of obligations on records and how these obligations can be met throughout a record's lifecycle.

المنارة للاستشارات
www.manaraa.com

Chapter 3

Protecting Relational Records

3.1 Requirements

The foremost obligation in the lifecycle of records is that of guaranteeing minimum retention. In a typical record's lifecycle, after creation and subject to certain conditions being met, it may be necessary to preserve it until some other pre-defined conditions are met. In most business situations, determining whether a record should be protected involves checking simple temporal conditions defined on the record (see Figure 3.1 for an overview).

As in the case with physical records, there are several levels of protection that need to be offered to relational records (previously defined as logical views). For example, protecting a physical sales invoice on which tax has been collected for the relevant period may imply that the business has to ensure that:

- The invoice is not destroyed (deletion).
- The invoice is not modified (update).
- No new details are added to the invoice or no pre-dated invoices are created (insertion).

Additionally, there need to be mechanisms through which users are able to specify the period of time in which a record is to be protected and the conditions under which the record should be deleted. This chapter deals with the problem of specifying protective retention policies and presents an outline of how such functionality can

للاستشارات المنارة
www.manaraa.com

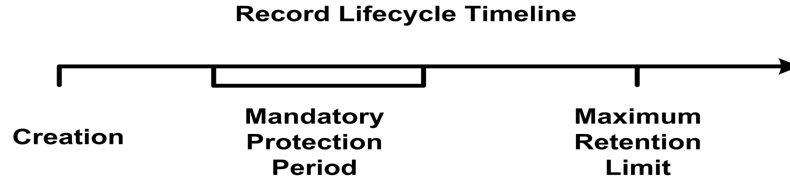

Figure 3.1: The retention timeline of a typical record. Note that not all data inserted in a database may lead to the creation of logical records as defined by the users. Furthermore not all records may have mandatory protection periods and retention limits.

be implemented in a typical off-the-shelf relational database management system. Our framework adopts an approach for protective policy specification that allows multiple retention policies to be specified for each record. Consequently users are first required to define meaningful records and then instantiate protective retention policies on them.

3.2 Protective Policy Specification

Before discussing how users are able to specify individual protective retention policies, the notions of retention conditions and protection levels need to be introduced. We remind the reader that individual records should now be considered to be user defined views over the physical schema.

Definition 3.1. *A retention condition is an optional boolean formula applied to every tuple of a record for the purposes of policy enforcement.*

In the context of a protective retention policy, if the retention condition holds for a tuple in the view specified by the record definition, then that tuple should be protected as per policy specifications. If a retention condition is not specified then it is considered to be always true. Each protective policy may only have one retention condition associated with it. The purpose of a retention condition is essentially to separate the critical tuples that warrant protection in the view specified by the record. A retention condition may be temporal through the use of the variable NOW, which was introduced earlier. Therefore temporal retention conditions can be used to specify minimum retention periods in the context of protective retention policies.

المنارة للاستشارات
www.manaraa.com

Protective requirements for relational records are captured using two distinct protection levels, namely *update protection* and *append protection*.

Definition 3.2. *A relational record under update protection is one in which any modification to the database is rejected if it leads to a modification (or a deletion) of a tuple in the record for which the retention condition holds true.*

Intuitively a view being update-protected contains some tuples that can not be updated or deleted. These tuples are exactly those for which the retention condition holds true. However for tuples where the retention condition is false, all modifications to the base data that lead to the modifications of such tuples are legal.

Definition 3.3. *A relational record under append protection is one in which any modification to the database is rejected if it leads to a new tuple becoming part of the record where the retention condition on that new tuple would be true.*

Intuitively a record which is append-protected does not allow the record to increase in size subject to the retention condition. To summarize, update protection disallows modifications (deletions and updates) of protected tuples in the record, whereas append protection disallows new tuples from becoming part of the record. When combined together, these two levels of protection capture all functional requirements of protecting records as discussed in Section 3.1.

3.3 Syntax and Examples

Let us now examine a simplified scenario that will illustrate the notion of temporal records and policies and demonstrate how both levels of record protection can be used to accomplish various retention requirements. The following two relations will be used to depict the schema of a typical invoicing system:

Invoice (<u>INV_ID</u>, Date, Approved, Paid, ...)
InvoiceLineItem (<u>ILI_ID</u>, <u>INV_ID</u>, Description, Amount, Tax, ...)

Let us also define two different records as follows:

Example 3.1
$OldUnpaidInvoices = \{R_1\}$
where $R_1 =$ `SELECT INV_ID, ILI_ID, Description, Amount, Tax`

www.manaraa.com

```
FROM Invoice, InvoiceLineItem
WHERE Years(NOW - DateTime) > 5
AND Paid = 'False'
AND Invoice.Inv_ID = InvoiceLineItem.Inv_ID
```

Example 3.2

$Invoices2006 = \{R_2\}$

where $R_2 =$

```
SELECT *
FROM Invoice, InvoiceLineItem
WHERE Date < 'Jan 01, 2007'
AND Date >= 'Jan 01, 2006'
AND Invoice.Inv_ID = InvoiceLineItem.Inv_ID
```

We begin by introducing a formal syntax for defining record protection policies. As stated earlier, a protective retention policy must be specified on pre-defined records. Note that the proposed syntax offers two forms of update protection, one where all columns of a record are protected, and the other where only selected columns are protected against updates.

Syntax 3.1

```
DEFINE  <Policy_Name> AS
PROTECT <Record_Name>
FROM    <UPDATE <column_list> | APPEND | ANYCHANGE>
WHILE   <retention_condition>
```

3.3.1 Protection Levels

To differentiate between UPDATE and APPEND protection consider the following two protective retention constraints declared on the record type *Invoices2006* defined above:

Example 3.3

```
DEFINE  Constraint_A AS
PROTECT Invoices_2006
FROM    UPDATE *
WHILE   Paid = 'True'
```

Example 3.4

```
DEFINE  Constraint_B AS
PROTECT Invoices_2006
```


www.manaraa.com

```
FROM    APPEND
```

Constraint A simply specifies that all invoices marked paid along with all their line items should not be updatable (or deletable). As far as the underlying data is concerned, Constraint A will effectively abort all transactions that lead to modifications of any tuple in the *Invoices2006* view if the tuple had the paid attribute set to true.

While Constraint A prevents updating of tuples that are protected, it does allow new tuples to become part of the record. Without Constraint B users are free to create invoices for the year 2006 and mark them as paid today, giving them immediate protective status. Consequently append protection as demonstrated in constraint B is used to manage the conditions under which a new tuple can be inserted into the record. The ANYCHANGE protection level is simply a simpler way of enforcing UPDATE * and APPEND at the same time. Using variations of update and append level protective constraints with a variety of temporal retention conditions, many flexible policies can be implemented on pre-defined records.

3.3.2 Temporal Records and Protection Levels

Although database systems have the ability to abort statements and transactions that attempt to modify protected views, it is impossible to abort the passage of time. Consequently, the definition of append level protection must be clarified to state that it will only protect records against user initiated modifications of the underlying data.

As an example consider the temporal record *OldUnpaidInvoices* (Example 3.1) defined above and the following append level protective constraint:

Example 3.5

```
DEFINE  Constraint_X AS
PROTECT OldUnpaidInvoices
FROM    APPEND
```

The retention condition in Constraint X (none present) is always true which may mislead us to believe that any increase in the size of the record should be rejected. However as invoices become older than five years and remain unpaid, they will automatically become part of the record and thus violate this constraint. Since we are unable to stop temporal violations of retention policies, we take this as valid behavior.

المنارة للاستشارات
www.manaraa.com

3.4 Framework for Implementation

3.4.1 Non-Temporal views

We note that protective retention policies on views simply disallow updates and insertions and resemble access control features on views. Implementation of such features is not a significant challenge and there are several approaches that can be taken including relying on access control (on base relations) or piggy backing protective retention policies over integrity constraints. However, the overall task of monitoring views and critical rows is equivalent to that of efficient view maintenance.

We now discuss the details of how the problem of implementing protective retention constraints can be reduced to that of view maintenance. In essence we use the previously established definitions of a retention condition and record to define a protected view. This view, which is formed by adding the retention condition into the record definition, will be denoted as a *critical view*.

Definition 3.4. *For a record specified by expression Q and a retention condition on Q specified by $C(Ret)$, the critical view for the associated policy is denoted as V_c and specified by $\sigma_{C(Ret)}(Q)$.*

In short, implementing update protection policies can be reduced to monitoring a critical view and rejecting all updates to the database that cause updates in the critical view. Similarly, append protection can be implemented by monitoring the relevant critical view and disallowing all updates to the database that add a new tuple to the critical view. The problem of detecting whether a particular update will impact a view has been well studied in the context of *relevant* and *irrelevant* updates to views.

Definition 3.5 (Blakely et al.[BCL89])**.** *For a given database D, a view definition V and an update U which takes D from instance d to instance d ′, U is considered to be irrelevant to V if V(d) = V(d ′) for all instances d.*

If an update cannot impact a view regardless of the database state then it is considered irrelevant. Blakeley et al. also define *irrelevant insertions* and *irrelevant deletions* with a similar meaning to that of update. Detecting irrelevant updates is fundamental for the efficient maintenance of a large collection of materialized views.

المنارة للاستشارات
www.manaraa.com

In the special case where the view definition language is restricted to conjunctive queries we can efficiently detect irrelevant updates [BCL89, BLT86]. Furthermore, it was shown that in the absence of the negation operator $O(n^3)$ runtime in the number of predicates is possible for detecting irrelevant updates of conjunctive views. Essentially for any arbitrary conjunctive view specified as $\sigma_{C(Y)}(R_1 \times R_2 \times ... \times R_n)$ and a tuple $t :< a_1, a_2, ...a_p >$ in R_i where $1 \leq i \leq n$, whether that t plays a role in the view can be determined by substituting t in $C(Y)$ for the relevant predicates and solving the satisfiability problem. If the expression is unsatisfiable, we are certain that t is irrelevant to the view.

This result is of extreme importance in our case. Firstly we argue that conjunctive queries are 'good enough' to describe a large number of business records. Most physical documents are designed to be straightforward, consequently the language used to describe them does not require significant expressiveness. We have examined several examples of physical documents such as sales reports, invoices, telephone bills, and all can be specified with relative ease using a combination of conjunctive queries. Furthermore, we point out that this result is not weak in any way. As noted by Cohen [Coh06], this result can be trivially extended to aggregates and a broad class of user defined functions. The observation is that for relevancy, a view V with an aggregated attribute is essentially equivalent to a view $V\prime$ with the same attribute without aggregation. If an update is irrelevant to $V\prime$ it is also irrelevant to V. New insertions in $V\prime$ that are 'neutral' in the context of aggregation (for example 0 has no affect for summation) are also irrelevant for V. Note that similar arguments could be made for many operations performed on data specified through the use of conjunctive queries.

While detecting irrelevant updates can significantly improve the performance of concurrently maintaining a large number of views, we gain nothing if we learn that a particular update is relevant. A relevant update has the *potential* to affect a critical view, but the actual database state needs to be examined to conclude whether the view will actually require changes. From an implementation perspective, protecting tuples that satisfy the criteria of being in a critical view does not require full materialization of the critical view. The cost of checking individual tuples which are being updated against all critical relevant view definitions is significant, but we will shortly demonstrate that in practical scenarios it will rarely be incurred.

To summarize, conditionally protecting non-temporal records can be done through direct monitoring of the critical views that they specify. This functionality is already present in most off-the-shelf systems and can also be modeled using mechanisms such as access control or integrity constraints. For conjunctive queries, which are suffi-

المنارة للاستشارات
www.manaraa.com

ciently expressive for our scenario, many worst case costs can be easily avoided by leveraging techniques presented in the literature for incremental view maintenance.

3.4.2 Temporal Views

In the context of data retention we do not require temporal views in their full generality since we do not model database histories. Instead we and are only interested in implementing and enforcing policies on the current state of the database. Consequently the class of temporal views that we need to support is very limited. In our model temporal views only result from the use of the temporal function NOW in the selection predicate of queries. Such views can also be considered to be sliding windows with respect to the system clock. The key issue with such views is that they may require maintenance at every clock tick. In other words they may need to be refreshed before every query can be processed, even if that query does not affect the state of the database. The notion of detecting relevant updates in this case needs to consider the passage of time as well as modifications to base data.

Bækgaard and Mark [BM95] performed a comprehensive analysis of such views. Their work relied on the fact that since NOW is a monotonically increasing function, we can pre-determine which tuples will eventually become part of a temporal view and similarly pinpoint which tuples will no longer be part of the view as time progresses. By keeping an ordered list of such tuples in a view denoted as a "superview", which itself is maintained incrementally, we can greatly reduce the need to recompute temporal views from scratch. This idea is a special case of deferred/scheduled maintenance of views where we are able to prepare in advance for the actual maintenance. Analysis shows that the efficiency of this technique directly depends on the additional space available for superviews [BM95]. If a superview cannot hold all tuples that will eventually become part of the primary view, then additional costs may arise to maintain the superview. For example, consider a view listing all flights that have departed from a particular airport in the last 24 hours. If the future superview is limited in space such that it can only accommodate flights departing from the airport in the next three days, we need to recompute the superview itself within the three day period so that it too can remain consistent to serve the primary view. However if a superview is "large enough", efficiency is not a major concern as the window to recompute it will be large, providing an adequate safety margin for the re-computation of the superview.

المنارة للاستشارات
www.manaraa.com

3.4.3 Expected Performance

Although we have not implemented and tested a protective retention framework for records, we envision that the performance penalty incurred by such a system will be minimal. There are several reasons for this claim, and we conclude this chapter by presenting a brief summary of our findings that support our hypothesis.

In the context of non-temporal views, it is generally accepted that as the complexity of the view (expressiveness of the associated language) increases, so does the cost of maintaining it. Over the past two decades significant progress has been made in reducing overheads associated with view maintenance. We believe that the majority of transactional business documents can be described as a collection of queries expressed in a limited query language and that such queries will be feasible to monitor as views. Our examination of physical business records on which legal retention obligations are specified has concluded that, it is very unlikely that features such as multiple levels of nesting and recursion will be required to describe such documents. Furthermore it is also anticipated that typical databases will already be maintaining materialized copies of views that are actively used to generate business records. Therefore piggy-backing on existing infrastructure for views should not significantly impact the monitoring of non-temporal views.

Although dealing with temporality may seem like a daunting challenge, in the realm of legal retention policies that is simply not the case. There are many issues with regard to time and physical business records that make management of temporal records much easier than the overall problem of temporal view maintenance. Firstly we note that retention policies are usually specified on the order of days and often months. For example, an invoice created precisely on July 22nd 2006 at 14:12:32.122 is typically treated to be created on July 22nd and the retention policies specified on the record use the granularity of days if not weeks and months to specify how long after an event it should be protected. Thus the window of time available to prepare for record protection and destruction is large and actions on the record do not require arbitrary temporal precision. In light of the previous discussion of temporal views (and superviews) this implies that maintenance of the relevant critical temporal views can be done lazily and at times when the database system is under low load conditions.

Secondly, in the context of views, we had noted that relevant updates had the potential to impact multiple views and that in the worst case we may need to check the contents of each view and the underlying relations to check for policy violations. However we can use temporal properties of business records to our benefit and

المنارة للاستشارات
www.manaraa.com

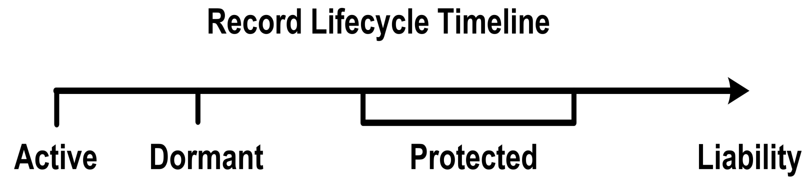

Figure 3.2: The number of modifications to a record typically declines with the passage of time. The majority of the updates happen on active and newly created information. It is also unlikely that there will be a large number of attempts to modify protected records.

almost completely eliminate this problem. In many business scenarios, as records get older they achieve temporal stability and are less likely to be modified. Intuitively we do not expect organizations to regularly modify old records such as invoices issued in prior years (Figure 3.2). Consequently it may be beneficial to identify individual tuples in relations that are current and have no retention obligations on them (complement of tuples participating in the critical view), and then check if the update operations impact any non-active tuple.

Finally we note that other heuristics for efficient monitoring of temporal views can also be developed using the above mentioned temporal stability property of records. One important property of physical business records is that they are typically referenced monotonically with respect to time. For example, if Invoice #500 was created on Jan 01, 2006 and Invoice #1355 was created on Dec 31, 2006 it is very likely that the entire range of invoices numbered between these two will also have been created in the year 2006. In such situations simple maintenance of pointers provides us with virtual indexes for very efficient checking of temporal conditions. However mechanisms for automatic inference of such correlations must be implemented or users should be able to specify them, such that the implementation framework is able to benefit from this knowledge about records.

المنارة للاستشارات
www.manaraa.com

Chapter 4

Timely Destruction of Relational Records

In this chapter we take a closer look at how individual business records modeled as sets of relational records can be removed from a database system when they have persisted beyond their specified maximum retention period. Our primary goal is to formalize the notion of deletion for arbitrary relational expressions and examine implications of our definition of deletion in the light of typical retention requirements.

The natural equivalent of destroying a physical record in the relational space is that of deleting data which led to the creation of the particular record. However because of several properties of relational databases such as functional dependencies and other integrity constraints, data in arbitrarily defined views cannot simply be deleted. We begin by proposing semantics of how users can specify record deletion policies and the features that should be available to them in any view based record retention system. We then examine restrictions that must be placed on record removal policies and demonstrate how they can be efficiently checked statically (and dynamically where necessary) to determine if they violate database integrity constraints. Finally we conclude this chapter by discussing how aspects of record and database design can significantly affect the scope of enforceable retention policies.

4.1 Requirements

The motivation for destruction policies comes from organizations wishing to absolve themselves of the liability associated with records derivable from its data. There are

المنارة للاستشارات
www.manaraa.com

two natural ways to "destroy records": users can either delete all data comprising the record or modify certain parts of it such that it becomes *safe* to preserve indefinitely. Selective modification of records is often done to anonymize private and personal records such as hospital records. Anonymity ensures that data retains its business value, in terms of statistics, and at the same time poses no storage liability. In a view based organization of data this implies that users should be able to specify when tuples need to be deleted or, when and how certain attributes need to be modified. To do this we reuse the notion of a retention condition to specify a logical condition which when satisfied by a tuple, implies that it should be removed from the view or appropriately modified. Note that we still rely on the notion of critical views but they are now treated in a new way. The new objective is to ensure that critical views for destructive policies are always empty. As tuples in a record become part of the critical view, they begin to satisfy the destruction condition and should be disposed immediately.

A significant limitation of our proposed approach to records removal is that for complex views involving multiple relations, the meaning of deletion or modification (update) must be well defined for policy enforcement purposes. But before we discuss updates on views in the light of business records, we must separate the objectives of monitoring and enforcement. It is important to note that in many situations organizations may only be interested in continuous monitoring of records for policy violations. Once a policy violation is detected, the system administrator may be notified to resolve the issue. The remedial actions such as deletion of critical data in such situations can be performed on a case by case basis, by the administrator. However, an ad-hoc management of retention violations is only feasible if the number of violations occurring is small and there exist good business work-flow processes. We also mentioned earlier that there is often significant flexibility in the window of time available to delete critical data. Organizations that adopt the ad-hoc approach to dealing with records retention policies must also have adequate time for administrator action, or they must design policies in a pre-emptive fashion. For example, if a legal retention limit for a record is one year, organizations going with such a strategy could be expected to enforce a retention policy that is stricter and informs administrators regarding imminent policy violations after 11 months. Ad-hoc enforcement of retention policies is trivially accomplished by coupling user alerting mechanisms, such as triggers, with the monitoring of critical views. The difficulty lies in ensuring that meaningful human intervention actually takes place and compliance is achieved. Consequently we will focus on automated enforcement and the complications associated with balancing flexibility in actions that can be

المنارة للاستشارات
www.manaraa.com

performed to destroy records and the compliance guarantees that can be offered.

The most flexibility in automated enforcement can be offered if on every policy violation a user specified program (such as a stored procedure) is executed. This user programmed procedure could be arbitrarily complex, may examine any part of the database, and then perform any appropriate actions (updates) to deal with the violation. A more restrictive approach is that of only allowing modifications to the view through the record definition itself. Such modifications may include setting individual attributes in the critical view to null or deleting tuples in specific base relations. Although it may seem illogical to reduce flexibility in actions that can be performed on records on policy violations, we will shortly demonstrate that doing so can have significant advantages. Restricting modifications through views leads to several questions such as what kind of views can be supported and how updates can be specified or inferred. There has been significant work done on detecting ambiguity in updates for views [BS81], automatically inferring correct updates [Kel85], and helping users visually map updates on views to base data [Kel86]. We argue that the framework used for updating views is orthogonal to our problem of enforcement. As long as the specified enforcement actions are not ambiguous for the record definition, we can fully enforce all actions that the records manager specifies. We believe that any records management system on views must be flexible and be able to provide conclusive guarantees regarding enforcement. Consequently we support both these paradigms in our framework. Users should be able to write special procedures for handling non-empty critical views or be able to directly make modifications to the view (if possible). Before we discuss the benefits of both approaches we present what we envision as a simple syntax for defining destruction policies. Note that in the syntax given below, records managers have the ability to specify actions such as deletion from specific base relations and simple overwriting of data in the view, in addition to the execution of their own custom built procedure:

Syntax 4.1

```
DEFINE <Policy_Name> ON <Record_Name>
DO     DELETE FROM <relation> |
       SET <assignment_list> |
       EXEC PROCEDURE <proc_name>
WHEN   <retention_condition>
```


المنارة للاستشارات
www.manaraa.com

4.2 Flexibility and Proof of Compliance

One general benefit of records retention policies is that businesses can use them as defense in civil litigation scenarios [MS02]. However to satisfy courts and government agencies that the relevant data has been destroyed, two independent conditions need to be met. Firstly, outdated data should physically not exist within the organizational database, for if it is found in an audit, we have failed to abide by given requirements. More importantly with the introduction of legislation such as the Sarbanes-Oxley Act, businesses need to prove beyond a reasonable doubt that there exist policy mechanisms, manual or automated, within the organization by which the relevant data must have been destroyed. The combination of compliance and providing proof of compliance has been one of the strong selling points of modern ECM systems. They allow businesses not only to demonstrate transparently the non-existence of outdated data but also to offer a proof of it being destroyed according to a policy.

Such 'proofs' of compliance given by businesses typically rely on simple and on best efforts arguments. For example, consider a scenario where an organization's internal emails are part of a subpoena and it only offers the past two years' emails as evidence. A proof of compliance in this case could simply be the use of suitable records management suite such as Microsoft Exchange Server or OpenText's Email Management System, with the relevant policy rule of deleting emails older than two years. Strictly speaking a formal "proof of compliance" can never be given because of uncertainties such as emails persisting on paper and in disconnected nodes (more on these issues in Section 6.2.3). However, the audited use of ECM systems offers a certain level of confidence and a potential guarantee to external observers. The value of such proofs can be witnessed from the fact that many ECM systems, especially for email management, are now being branded as capable of offering "Department of Defense (DoD) compliance" and "Environmental Protection Agency (EPA) compliance." Government agencies such as the United States DoD naturally do not want their records to be floating around in their sub-contractors' computer systems. In these cases, an organization (or a contractor) communicating with the Department of Defense through email, must demonstrate compliance through the use of retention-aware email management systems. Thus it becomes obvious why ECM vendors have jumped on the opportunity to capitalize on this requirement of a meta proof of compliance.

We believe that any data retention framework for relational databases systems must also be able to provide a proof of compliance. As awareness of data retention

المنارة للاستشارات
www.manaraa.com

encompasses relational systems, providing proofs of deletion (or similar actions such as overwriting with nulls) of data will become a necessity. However, these proofs must be better than best-effort (reasons discussed in Section 5.2), and if offered must formally hold in all instances of the database. We argue that if a proof of compliance is required, retention policies and actions must be treated as first class citizens similar to integrity constraints. Analogous to the fact that no user (not even the administrator) is able to violate fundamental rules such as primary key and foreign key constraints, no user should be able to bypass and violate retention policies. Assuming that retention policies are correctly formed, required actions are timely executed without any possible chance of exception, and the closed world assumption holds for the database, then a proof of compliance is easily derivable.

Note that we can still entertain the notion of flexibility in policy enforcement, but it cannot be interchanged with the correctness of a proof of compliance. For example, if a user programmed stored procedure is proven (through formal verification) to delete all outdated records in all possible instances of a database without exception, and this programmed procedure is executed weekly, then we certainly have enough evidence to prove that our database is retention compliant up to a weekly basis. We will refer back to this all-encompassing user programmed procedure to examine what are its fundamental requirements if it is to provide a formal proof of compliance. This procedure/oracle is the only benchmark for existing solutions and is the closest mechanism that can be compared to our proposed framework

Sadly, verification of user programmed procedure containing arbitrarily defined policy rules is non-trivial and is essentially equivalent to the problem of verification of arbitrary C programs. Depending on the data structures used by this procedure and its complexity, the solution may simply be infeasible. Furthermore creating and maintaining such procedure(s) in itself is a daunting task. Fortunately, by reducing the flexibility in enforcement actions we can make the problem of creating and reasoning about policy actions much more manageable.

4.3 Correct Enforcement

Let us assume that we are given a procedure (oracle) to delete outdated records when a policy violation is detected. The first critical requirement for proving correctness of this procedure is to prove that for all database instances and for all policy violations, the actions taken as a response by the oracle have a 'remedial effect.' More specifically, whenever a critical view becomes non-empty, the actions taken by the

المنارة للاستشارات
www.manaraa.com

oracle must eliminate the newly added tuples from the view. Yet another way to look at it is that for every policy violation the oracle must be notified only once and it should always resolve the violation successfully. We formalize these requirements using two correctness criteria as defined below:

Definition 4.1 (Weak Correctness)**.** *A destruction policy P is weakly correct if on any tuple becoming part of the critical view V_c the actions specified by the policy denoted by $\alpha(R)$ will ensure that V_c will be empty.*

Definition 4.2 (Non-Invasive Policies)**.** *A destruction policy P is non-invasive with respect to another policy P' if the actions specified by P denoted as $\alpha(R)$ never affect the critical view of P'. P is called invasive with respect to P' if $\alpha(R)$ has the potential to change the contents of the critical view of P'.*

Definition 4.3 (Strong Correctness)**.** *A destruction policy P is strongly correct if it is weakly correct and*
i) P is non-invasive with respect to all other destruction policies in the database or
ii) P can only delete tuples from all other destructive critical views

Weak correctness implies that actions specified by the policy must be meaningful for the record itself and that the critical tuples will be promptly removed from the view by applying $\alpha(R)$. Weak correctness by itself does not provide any guarantees for termination of policy actions. This is because actions of a particular destruction policy may introduce tuples in critical views of other destruction policies. Consequently the notion of non-invasive policy actions is presented so that we can reason about policy triggering patterns. Note that the scope of invasive policy actions is not restricted in any way by the definition. Actions could themselves directly impact other views or indirectly do so, for example as a side affect of several integrity constraints such as cascading-deletes of foreign key references.

Lemma 4.1. *A set of destruction policies $P = \{P_1, P_2, ..., P_n\}$ is guaranteed to be terminating, if the directed graph constructed with edges from P_i to P_j when P_i is invasive with respect to P_j, has no cycles.*

The above lemma specifies a strong condition that needs to be met to avoid circular enforcement of retention destructive actions. Note that this result is very

المنارة للاستشارات
www.manaraa.com

similar to the solutions presented to the classical problem of determining trigger termination and cyclic execution of rules [vdVS93, AWH92]. This result is also known to be strictly stronger than what is minimally required for guaranteed termination [LL99], since cyclic execution can also be shown to terminate if the execution depth is bounded. However in our case, the likelihood of having bounded recursive dependencies between business records is unlikely, therefore we adopt the simpler graph theoretic approach to verify termination properties of destructive policy actions.

4.4 Weak Correctness

For arbitrarily defined views and enforcement actions statically determining if they satisfy the above stated correctness criteria is undecideable. However if we consider the class of updatable views specified by conjunctive queries, with actions limited to deletion from specific base relations or overwriting with statically determined values, we can efficiently show whether a policy is correct.

Lemma 4.2. *For a record R specified as an updatable conjunctive view, a destructive retention policy P leading to critical view V_c with conditions $Critical(R_p)$, and a simple remedial action specified by $\alpha(R)$, P is weakly correct if and only if $Critical(R_p) \wedge \alpha_p(R)$ is unsatisfiable, where $\alpha_p(R)$ is the post-condition of $\alpha(R)$.*

When dealing with conjunctive views and simple actions as defined above, this lemma states that we can easily check statically whether our policy actions are correct. In short we need to solve the satisfiability problem for the formula $Critical(R_p) \wedge \alpha_p(R)$ and can then be certain that enforcement is guaranteed if and only if this formula is unsatisfiable. As an example of this, consider a simple record derived to track credit card transactions over the following schema:

Transaction (<u>TxnID</u>, *CstID*, CardNumber, TxnDate, ...)
Customer (<u>CstID</u>, Name, ...)

Using the above hypothetical database we define the following record and destruction policy.

Example 4.1
$OldCreditCardTransactions = \{R_1\}$
where $R_1 =$ `SELECT TxnID, CstID, CardNumber, TxnDate, ...`

المنارة للاستشارات
www.manaraa.com

```
FROM Transaction, Customer
WHERE Transaction.CstID = Customer.CstID
AND CardNumber IS NOT NULL
```

Example 4.2

```
DEFINE Obscure_CardNumbers ON OldCreditCardTransactions
SET    CardNumber = null
WHEN   Days (NOW - TxnDate) > 365
```

In this scenario an organization is attempting to destroy credit card numbers for transactions that took place more than a year ago. To accomplish this, on detection of a non-empty critical view the specified action will set the CardNumber attribute of the violating tuple to *null*. Note that in this case the logical condition for a tuple to be in the critical view, $Critical(R_p)$, is $(Transaction.CstID = Customer.CstID) \wedge (CardNumber \neq null) \wedge (Days(NOW - TxnDate) > 365)$. Thus the proof of compliance can simply be generated by solving the satisfiability problem which combines the constraints of the critical view and the remedial action, that is $(Transaction.CstID = Customer.CstID) \wedge (CardNumber \neq null) \wedge (Days(NOW - TxnDate) > 365) \wedge (CardNumber = null)$. Since we have a contradiction on the attribute CardNumber, this formula is unsatisfiable implying that there can not exist a tuple which can satisfy both $Critical(R_p)$ and $\alpha_p(R)$ at the same time. Therefore we have effectively proven that the action $\alpha(R)$ when applied to *any* tuple in the critical view will cause it to no longer persist in the critical view. The reverse implication for this lemma is also trivially provable, for if $Critical(R_p) \wedge \alpha_p(R)$ is satisfiable, there can exist at least one tuple on which performing the remedial action will not remove it from the relevant critical view, and thus our policy action may fail to have any effect in that case.

This simple check allows us to be certain that the record lifecycle will always terminate in destruction (defined as removal from the critical view) and more importantly we can formally prove this by solving a problem which is at worst NP complete. The fundamental observation to make is that, while we can not reason about arbitrarily complex views and user programmed procedures, we can be certain about policy execution on a restricted set of views and actions. For updatable conjunctive views and simple actions, we can determine statically at policy-creation time whether there can exist a situation where our enforcement actions can fail to provide the remedial effect. Consequently for a restricted class of relational records we can now work objectively towards offering a formal proof of compliance.

المنارة للاستشارات
www.manaraa.com

4.5 Integrity Preservation

It was mentioned earlier that to reason formally about retention policies and their effect, we have to treat them as first class citizens in a database system. However, actions performed by retention policies to rectify a policy violation should never compromise the integrity of the database. In this section we will further refine our notion of offering a proof of compliance to include integrity preservation.

Definition 4.4 (Integrity Preservation)**.** *A destructive policy P is integrity preserving if the suggested actions $\alpha(R)$ when applied to any valid instances of the database d will also lead to a valid instance of the database with respect to all specified integrity constraints.*

Recall that one of the requirements for provable automated enforcement is that the user programmed actions for enforcement should always (without exception) terminate successfully. Note that this requirement is independent of the previously discussed requirement of correctness and the actions having a remedial effect. For example the actions taken in response to a policy violation may simply do nothing and always terminate in the same valid database state as the one which has a retention policy violation. We need to consider the word "exception" in a sense similar to that used in programming environments, where it is quite often used interchangeably with the notion of an error. One aspect of proving that enforcement actions are always error-free is that of proving that they will never violate integrity constraints. For example, let us assume that a user programmed procedure in response to a policy violation attempts to delete a tuple in a relation that contains a foreign key reference. There could be many reasons why this could happen, including badly formed record definitions and poorly programmed actions. But unless we can be certain that we will never encounter such a state, where user specified actions cannot proceed without administrator intervention, we can not fully prove that our policies are automatable.

Once again, offering such guarantees for arbitrarily defined views and actions on them is impossible. It should also come as no surprise that the previously discussed restricted class of records specified by updatable conjunctive views and simple actions on them can be restricted further to be made integrity preserving. We examine several types of integrity constraint and specify the restrictions that need to be placed on simple actions for guaranteed integrity preservation. We will assume that the database catalog is available to us so that we can make decisions regarding

المنارة للاستشارات
www.manaraa.com

the legality of the user specified actions. Our aim is to pinpoint actions that can cause integrity violations and detect them statically.

4.5.1 Primary Key and Uniqueness

One of the most significant problems (in terms of physical design related issues) in data retention, is of a primary key constraint. In short all data contained in a tuple cannot exist without the primary key uniquely identifying that data. Consequently potential modifications such as setting primary key values of attributes to *null* or other static values have to be rejected straightaway. To get rid of a primary key value, the only option users have, is to delete the tuple from the base relation. It follows that a uniqueness constraint on a tuple can also not be checked statically. We can never be certain that a modification to an attribute that is specified in the schema to be unique can always be legally done. Checks always need to be performed on the particular instance of the database to ensure that the updated value is unique and no universal guarantees that the actions will always complete can be given for all instances of the database. Consequently users requiring a formal proof of compliance must choose deletion of data in such situations.

4.5.2 Foreign Keys

Modifying foreign key attributes can often serve as a simple mechanism for data destruction and simultaneous preservation of vital database statistics. To motivate the application of modifying foreign key attributes and the associated intricacies, consider the following database schema storing data about traffic violations:

Drivers:(<u>LicenseNumber</u>, FirstName, LastName)
Violations:(<u>OffenseNumber</u>, OffenseDescription)
Committed(*LicenseNumber*, *OffenseNumber*, OffenseDate, ...)

The interesting relation in the above example with foreign key references is one which relates drivers with the traffic violations they committed. Let us further assume that we have a legal retention requirement such that traffic violations are removed from an individual's driving record after 12 years from the OffenseDate. However the statistics related to offenses should be maintained forever. When obscuring or overwriting a foreign key value, there are two possible courses of action.

المنارة للاستشارات
www.manaraa.com

If the foreign key attribute is nullable then it can always be set to *null*. However there can be many cases where null is not suitable. The other option is that of overwriting the foreign key value with a different but valid one. In the above scenario we could introduce a privacy preserving entity in the drivers relation to accomplish this. A *privacy preserving entity* can be informally described as an entity that does not exist in the real world and its associated data is there only for obfuscation and compliance with the existing schematic/integrity constraints. For example, a tuple in the Drivers relation such as ("A1234-567890","Unknown", "Person") could be used to refer to a driver that does not exist in the real world, but is there to support anonymous storage (in the given design) of all traffic violations committed more than 12 years ago. As soon as 12 years have past after a particular offense being committed by a real driver, the LicenseNumber of that driver related to the offense will be simply overwritten with "A1234-567890". The following record definition and constraint serve the scenario well:

Example 4.3

$CommittedViolations = \{R_1\}$
where $R_1 =$ `SELECT * FROM Committed`

Example 4.4

```
DEFINE ObscureOldViolations ON CommittedViolations
SET    LicenseNumber = A1234-567890
WHEN   Years (NOW - OffenseDate) > 12
       AND LicenseNumber ≠ A1234-567890
```

Note that "A1234-567890" is a statically determined value embedded in the policy, and such static assignments of foreign key attributes have two major consequences. Firstly we must ensure that "A1234-567890" should exist in the primary relation at the time of policy instantiation, and secondly the system must mandate that the statically used primary key of the privacy preserving entity (in our case "A1234-567890") will itself never be modified or deleted, thus ensuring that the policy remains valid throughout its lifetime. In short, static assignment of foreign key values in policy must introduce an implicit protective retention policy for the primary key, in our case the entity identified by "A1234-567890" in the Drivers relation. This implicit constraint, which itself can be modeled as a protective retention policy (See Chapter 3), ensures that the privacy preserving entities persist as long as the referencing destruction policies are active.

المنارة للاستشارات
www.manaraa.com

4.5.3 Other Issues in Integrity Preservation

Although in theory a set of delete/update actions can be shown to be error-free and correct, it is significantly harder to make general claims in a typical database system. One interesting complication in modern day database systems is the use of mechanisms such as triggers and check-constraints to enforce complex user programmed integrity constraints. With these additional procedures of arbitrary complexity in place, it becomes harder to determine statically the consequence of actions taken in response to policy violations. For example, if our attempts to remove outdated records are rejected by a user programmed trigger, we will be unable to offer compliance guarantees unless the trigger is removed (or suspended) and the remedial actions repeated.

There is no easy way to resolve this issue. A tedious approach would be to verify each user programmed trigger and check-constraint for sources of possible interference with the data removal actions specified by retention policies. If the number of user defined triggers is small and they are each of reasonable complexity, this task can be done manually. However if we have to deal with a complicated web of interrelated trigger-oriented events that can have side effects on the underlying data, the task of proving that our retention actions will always work will be extremely difficult. Trigger analysis and the verification of their properties such as termination has been long studied in the database literature [Wid96, DKM86]. However, even with the use of automated verification tools, proving strong properties for a large number of complex triggers, with all policy actions can be a significant challenge.

Another approach is to circumvent the problem by arguing that not only are retention policies first class citizens but they also have special privileges that allow them to bypass the invocation of triggers. What this means is that the monitoring of triggers and user programmed integrity constraints will be temporarily suspended when actions specified by a retention policy are being executed. Thus user programmed integrity constraints could be demoted to second class citizens in such scenarios. However the burden of being kind to user programmed integrity constraints will then be placed on retention policy actions. We also note that due to the complexities associated with triggers such as non-standard semantics and behaviour among database vendors, execution ordering, recursive execution, lack of a guarantee of termination, and resource consumption issues, they are often avoided by database programmers [SD95, CCW00]. Consequently this approach may be favorable in situations where the number of triggers present in a database system is limited.

للاستشارات المنارة
www.manaraa.com

While on the one hand disregarding triggers can instantly solve our problem of provably enforcing retention, on the other hand it may lead to unexpected results if the retention actions are badly designed. For example, if retention policies violate user programmed integrity constraints, user applications may be misled to make incorrect assumptions about the data. The core conflict here is between database administrators and records managers to determine which comes first, non-essential integrity or proof of compliance. Of course, if a formal proof and immediate deletion of outdated records are not critical requirements, conflicts between integrity and retention policy actions can always be resolved in favor of maintaining integrity. If the window of compliance is sufficiently large, we can always revert to the ad-hoc approach of conflict resolution where the violation is reported to a human arbitrator for review, who can then decide how to proceed further.

4.6 Schema and Policy Set Evolution

It is important to remind the reader that implementations of retention policies using views as abstraction for records are only interpretations of the legal data retention requirements for a given storage schema. If a change in the schema invalidates the previous interpretations (view definitions), it does not absolve the business from its records management obligations. Simply consulting the corporate privacy enforcement officer to reinterpret the affected records retention policies for the new schema is the recommended approach. However some transformative actions or policies may not allow the flexibility of a change in the storage of records. Similarly when new policies are being introduced, privacy administrators may not want to enforce new policies retroactively.

In such cases there may arise the need to maintain two parallel versions of the storage schema in order to fulfill obligations on the older schema as time progresses. The older snapshot of the database would not be used for actively inserting new business records, but rather only for fulfilling records management obligations on the records it contained at the time of the schema change. The deprecated schema and the data therein can only be safely discarded when all obligations as specified by relevant policy actions have been met. In situations where these requirements cannot be ported to the new schema, there is no choice but to maintain records until all of them have completed their lifecycle and no more policy actions will be required to fulfill retention obligations.

المنارة للاستشارات
www.manaraa.com

المنارة للاستشارات
www.manaraa.com

Chapter 5

Making it Work

In this chapter we further develop our notion of proof of compliance to include detection of inter-policy conflicts. We examine a procedure to detect overlapping and conflicting policies statically for simple records defined as conjunctive queries. A discussion of the various sources of conflicts is also presented which highlights how they can be avoided through the use of good physical design combined with precise and meaningful record definitions. We also present the results of our tests conducted that demonstrate that the overhead of our proposed records retention framework can easily be minimized in a typical business situation. We conclude this chapter by comparing our presented solution to data retention with existing approaches in the realm of ECM and privacy aware information systems.

5.1 Inter-Policy Conflicts

5.1.1 Detecting Conflicts

The final requirement for proving correctness for a set of policies that we examine is to show that they are consistent and conflict free. Inter-policy conflicts are basically caused by expired data that is to be removed or modified according to a destruction policy and is also at the same time protected under a protection policy. We call these *delete-protect conflicts* between policies and define them using the previously developed notion of invasive policies.

Definition 5.1 (Conflict Free Policies)**.** *A destruction policy D is conflict free with respect to a protection policy P if D is non-invasive with respect to P.*

المنارة للاستشارات
www.manaraa.com

In other words if the actions specified by a destructive policy can not impact the critical view of a protective policy then the two policies are guaranteed not to conflict with each other. In the context of database theory, the problem is reduced to that of determining whether a given update (destruction policy action) is irrelevant to a given relational expression (protected critical view). Note that our definition of invasive policies, in the context of cyclic execution of destruction events and conflicting policy actions, is essentially a specialization of Blakely's definition of irrelevant updates [BCL89]. If an update is irrelevant to a critical view defined by a policy, then that update can not impact the contents of the critical view regardless of the database state.

We can thus safely claim that detection of delete-protect conflicts is equivalent to determining whether modifications caused by destruction policies are irrelevant to critical views of protective policies. Furthermore detecting conflicts among simple select-project-join view based records is only NP-hard whereas in the general case detecting conflicts is undecideable. To prove total correctness we have to show that all pairs $\{P,D\}$, where P is a protection policy and D is a destruction policy, are conflict free. We point out that testing for policy conflicts is a one-time and offline cost incurred at policy instantiating time.

5.1.2 Source of Conflicts

There are two major reasons why conflicting policies can be introduced in any data retention framework. Firstly, retention requirements as stated may inherently be conflicting among themselves. To distinguish these conflicts from other types of conflicts, we call them *direct policy conflicts*. Trivially we can see that direct policy conflicts will always lead to delete-protect conflicts. Such conflicts are more likely to occur if policies written independently by various governing bodies are to be implemented on the same database. However for most small-medium businesses that are subject to laws from one country/authority direct policy conflicts will be relatively rare.

The primary source of delete-protect conflicts is the translation from policies stated in natural language to the implementation mechanism that we proposed in this thesis. Recall that in our discussion of ECM systems, the definition of a record was very simple and already given to us. In ECM systems a record is basically a self-contained file residing on a disk. Whether the file represents an email, a document or a spreadsheet is irrelevant because the notion of record in that scenario is very well

www.manaraa.com

defined. Most importantly a record (file) is atomic and its data can not overlap with another record in the system. Consequently conflicts arising through the shared use of data between multiple records do not exist in ECM systems. The only source of conflicts in ECM systems is cross-classification at the level of complete objects, for example a spreadsheet could be shared between two different departments of an organization that have two different retention policies for their spreadsheets. Similarly the only destruction action that ECM systems support is simple and self-contained deletion of the entire record and these deletions do not have any side effects such as cascading deletes of other records.

In relational database systems neither pre-defined mapping between records and data nor pre-defined destruction policy actions exist. There is a wide variety of options available through which protection/destruction of sensitive data can be accomplished. Consequently the task of choosing the best possible record definitions and destructive actions is significantly more challenging than in ECM systems.

5.2 Conflict Avoidance

A database administrator must first identify all records on which policies are to be enforced. More specifically, given a physical schema the administrator has to enumerate all queries that generate records that need to be protected/destroyed according to a retention policy. In a way this task is similar to that of organizational work-flow management, and all aspects of good record design used for business documents can also be used by the database administrator. For example, it would be inefficient to design a spreadsheet such that two different departments of a business work with totally disjoint parts of the spreadsheet. Just as it would make sense to split the single sheet into two separate and independently lock-able parts, it would also make sense that the database administrator enumerate queries defining records such that they have minimal overlapping data between them.

Based on our discussion in previous chapters, we argue that the single most important factor in determining the number of policy conflicts is the precision of record definitions. Direct translation of policies from an ECM system to our relational framework will lead to very broad record definitions and thus should be avoided. For example, if a telecommunications company is required to keep track of destinations of the calls placed by its subscribers, it would be unwise to take the ECM approach and enforce retention policies on queries that specify entire telephone bills. For such physical records the administrator has to identify very precisely parts

المنارة للاستشارات
www.manaraa.com

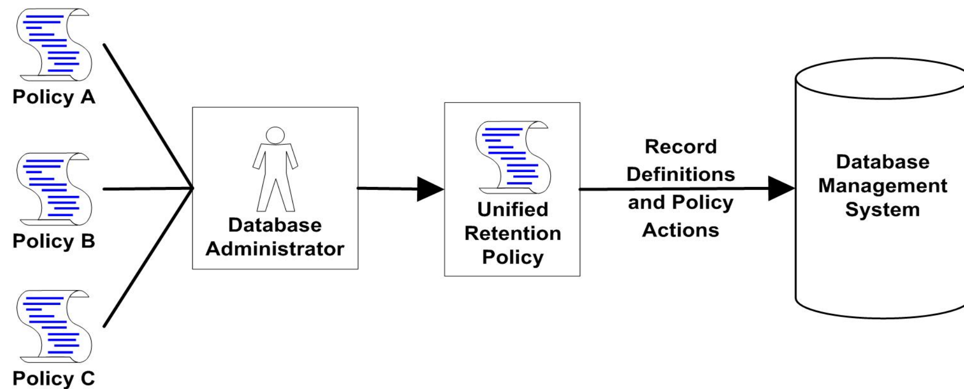

Figure 5.1: An ideal scenario where a set of policies is translated into record definitions and policy actions programmed by a single administrator. Even if conflicting policies are present, the same administrator can always re-interpret the set of given policies to resolve conflicts.

of each document that can contain sensitive data and the queries that minimally capture that data over all similar records. This means accessing only the required relations in record definitions and having as selective predicates on policy actions as possible. Typically the ability to accomplish this will directly depend upon the quality of the physical design of the database. For example the ability to reduce the number of joins in record definitions to a minimum directly depends on the given normal form of a schema and a well normalized schema can substantially reduce the number of overlapping records definitions.

In an ideal scenario (depicted in Figure 5.1) a single policy maker will consolidate various policies into one database. However for large multinational businesses encompassing various federated databases, it is unlikely that there will be a single expert that will translate all retention requirements into record definitions and protection/destruction actions. In such situations (depicted in Figure 5.2 on Page 47) the likelihood of conflicts increases significantly and all conflicts have to be ultimately resolved/mediated between the respective policy administrators.

Conflicts in policies need not arise directly because of data shared among records. In our examples we have largely discussed the use of select-project-join based views to define records. The elegance of these simple records can be misleading and it may give the illusion of making records management very easy. For more complex records, such as those that use operators such as intersection and union, understanding the data which they encapsulate becomes significantly harder. As an example consider

المنارة للاستشارات
www.manaraa.com

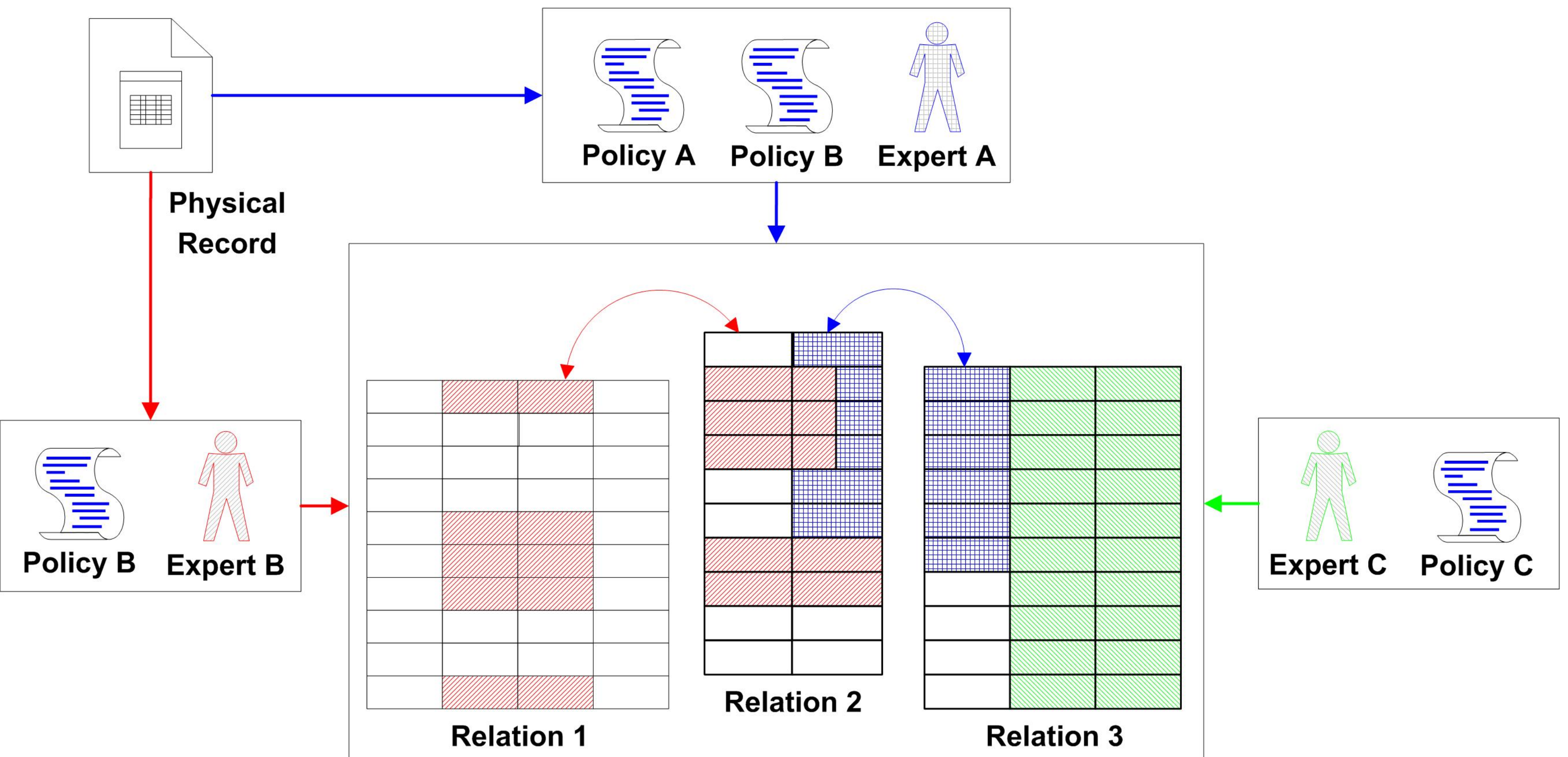

Figure 5.2: The figure shows a scenario where different interpretations of policies can lead to the same underlying data being subject to several and possibly conflicting retention policies. Note that the record definitions created by experts A and B share the same data which could become a source of conflict. Although records defined by experts A and C do not share any data, there could be functional dependencies in Relation 3 that limit the scope of what destruction/protection actions can be taken on records defined by A and C in order to avoid conflicts.

المنارة للاستشارات
www.manaraa.com

the use of the negation operator between two relations X_1 and X_2, each with a single attribute called *id* and two record definitions, R_1 which selects all *id's* in X_1 that are not in X_2, and R_2 which selects all *id's* in X_2 that are not in X_1. Note that by definition R_1 and R_2 are always disjoint ($R_1 \cap R_2 = \emptyset$) but the data/definitions of R_1 and R_2 are inseparable. Performing actions (insertions and deletions) on any of these records will most certainly have a direct impact on the other. We have repeatedly stressed the need to keep record definition "simple"; unfortunately we claim that it is impossible to formally define the notion of simplicity for arbitrary queries. This is because any definition of a simple query language may disallow certain policies to be enforceable. Simplicity of record definitions and policies is largely an abstract concept and a function of a database administrator's confidence in the meaning of each relational expression used to identify records for policy enforcement. Adopting a conservative approach and keeping to conjunctive queries only may not be a feasible approach for all situations. The only critical requirement that needs to be met for good record/policy design, is that administrators need to fully understand record definitions and the actions of their policies.

5.3 Implementation

Our framework for continuous records monitoring can best be summarized as a proposal for active integrity constraints and actions on views. We assert that the task of periodically monitoring destructive critical views and purging records will not be a significant source of performance degradation in our model. This is because of the temporal flexibility available in destructive data retention requirements and the fact that stricter versions of destructive policies can usually be enforced. For example, this flexibility allows us to execute daily or weekly batch jobs that enforce destructive retention policies without interfering with the online operation of the database system.

However if a record is accidentally deleted when it is supposed to be protected by an organization, the consequences are more serious. Naturally, there is no flexibility in protecting records and every modification to a database has to be checked against the relevant policies to ensure compliance. Consequently the cost of computing the effect of updates on protected records will be the most significant source of overhead. In light of this fact, the aim of our experiments is twofold: first, to measure the overhead of continuous record protection precisely for a broad mix of protective policies in a high-update and heavily regulated business scenario; second,

المنارة للاستشارات
www.manaraa.com

Policy #	Critical Row Coverage	Type
1	.08%	C
2	1.8%	C
3	6%	S
4	2.8%	C
5	6.6%	S
6	2.6%	S
7	5%	C
8	5.3%	C
9	.6%	C
10	13.3%	A
11	2%	A
12	100%	A

Figure 5.3: A summary of the policies. The critical row coverage represents the proportion of order and lineitem tuples that were relevant to a specific policy. The type denotes the complexity of the critical view being denoted as simple/on a single relation (S), conjunctive/involving multiple base relations (C) or involving an aggregate value (A).

to determine and recommend means of minimizing this overhead using features already present in existing database systems.

There are two widely used mechanisms to detect events specified by arbitrary relational expressions (such as a tuple becoming critical) in database systems: incremental computation or total re-computation. More specifically, to detect changes in the contents of a critical view, we can either materialize the view completely and implement triggers on the materialized view (a feature present in some commercial systems such as Oracle) or implement triggers on base relations that detect (possibly after execution of an additional query) whether the relevant critical view will be impacted by a triggering statement [CW91]. A detailed examination of implementing triggers on views, including an algorithm for mapping triggers on views to an equivalent set of triggers on base relations, has been presented in the literature [SNS06].

We note that for monitoring views and the effect of every update on views, there are two well known optimizations that can be exploited to reduce the associated overhead. First, we can benefit from the observation that certain policies and event

www.manaraa.com

detection on the associated views naturally favor a particular mechanism. For example using triggers to monitor a view with an aggregate value is ill-advised, given that this value will not be preserved after the trigger invocation and will require total re-computation at every relevant update. Therefore it would be prudent to selectively decide on materialization or re-computation for policies such that the relative overhead for each policy is minimized. Generally, the decision to materialize or re-compute depends on the average cost incurred per relevant update, and most database optimizers can quite easily assess the cost of a typical update and re-computation query, to give a reasonable estimate of which technique will be better than the other. Second, we can use to our benefit the fact that several policies can quite often be clustered around a small number of related tables. Instead of instantiating a large number triggers on base relations, the approach of trigger grouping [HCH+99, SNS06] can be used to reduce the number of triggers per table and to exploit the fact that multiple policies on similar predicates can be checked in a single trigger invocation. The result of these optimizations can lead to a significant reduction in the cost of view monitoring. Instead of incurring worst case costs of each monitoring mechanism (views and triggers), we can combine these into a hybrid critical view monitoring technique which attempts to take advantages of the positive aspects of both approaches.

5.4 Experimental Evaluation

For our tests we relied on the TPC-H (Transaction Processing Council- Benchmark H) schema depicting a business scenario involving the sale of parts to customers worldwide. We developed 12 different protective data retention policies and translated them into relevant views that would need to be monitored. These policies were directly derived from real-world records retention requirements imposed on TPC-H like businesses by various security, export and taxation agencies. Examples of such policies include protecting purchase orders involving the sale of special parts like uranium fuel rods, protecting order details with suspiciously large monetary sums, and protecting sales tax totals for specific countries. A detailed list of these policies, the synthetic parameters and the relevant views is given in Appendix A. Our policy set consisted of 3 simple policies on single relations, 6 policies leading to the monitoring of conjunctive views and 3 policies which involved protection of an aggregate amount. Although the majority of these policies individually had a low tuple coverage, defined as the number of tuples subject to protection/monitoring

المنارة للاستشارات
www.manaraa.com

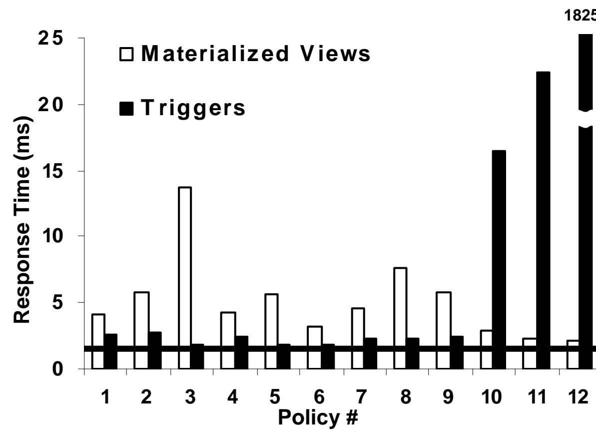

Figure 5.4: Average completion (commit) time for an update to a single tuple in a base relation protected under a single policy implemented using triggers over base relations and using the incremental maintenance approach. The horizontal line represents the cost of an update without any monitoring mechanisms in place.

in base relations, all tuples were protected against modifications from at least one policy (Figure 5.3).

Due to the nature of the TPC-H schema, most of the policies were clustered around the two largest tables in the schema (Orders and Lineitem). We believe that this will be a standard observation in databases used by highly regulated businesses, as these regulations can be expected to be uni-faceted and will certainly be based on the primary function of the business. For example, compared to a database maintained by a large stock-broker, a medical database used by a hospital will likely be subject to more retention obligations for patient and treatment records than for the financial records of the hospital. Therefore it is natural to expect that a large number of policies will be clustered around relatively few tables and, as we describe shortly, several avenues of optimization arise because of this observation.

Our tests were conducted using a 1GB dataset on ab Intel Core 2 Duo (1.8Ghz) machine with 1.5GB of RAM. All tests were performed on a warm database using DB2 v9.5 and involved issuing several thousand (typically 5000 or more) individual update statements on base relations that impact critical views defined by protective retention policies. The standard error in measured wall-clock response times in our tests was usually less than 5% of the average time to commit for an update.

Figure 5.4 summarizes the results of the overhead incurred by each of the 12 protection policies when they are individually implemented as a set of triggers and

المنارة للاستشارات
www.manaraa.com

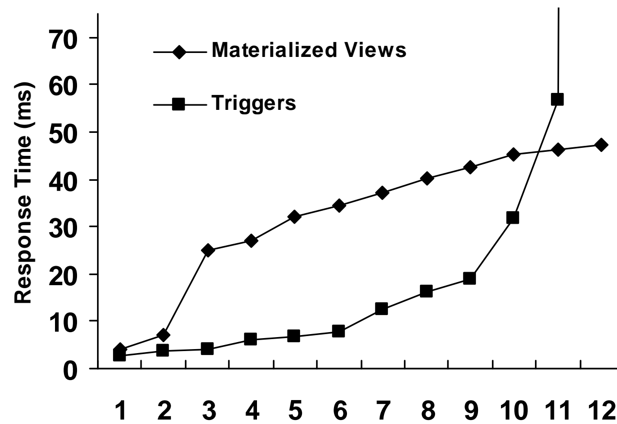

Figure 5.5: The scalability of incremental and total re-computation in detecting changes in the contents of critical views. The values on the x-axis represent the total number of protective policies being enforced on the database. For example at 7, Policies 1 through 7 are all being enforced at the same time.

as a materialized view. All updates were performed randomly over the dataset (each tuple was equally likely to be modified). Policies involving aggregated information (for example, policies 10, 11 and 12) clearly favor incremental computation, whereas for other policies the maintenance overhead caused by materialization is far greater than simple checking of updates for relevance.

The reason for triggers individually performing better than materialized views for event detection is largely due to the TPC-H schema and specifications. For example a purchase order in the TPC-H schema is related to only one customer, nation and region. Furthermore, the data contained in any purchase order includes at most seven line items. Consequently the majority of the simple policy decisions on updates pertaining to individual purchase orders can be made by examining a small number of tuples. It is only when repetitive re-computation of aggregates takes place that triggers pay a heavy price.

Figure 5.5 demonstrates how both the incremental and total re-computation approaches scale independently as more and more policies are implemented. Materialization suffers from the overhead of view maintenance whereas triggers have scalability issues arising from one policy on a view being translated into multiple triggers on base tables. For maintaining efficiency in transaction processing most commercial database systems limit the number of triggers that can be instantiated on a relation (typically fewer than 64). Consequently it is very unlikely that, when

المنارة للاستشارات
www.manaraa.com

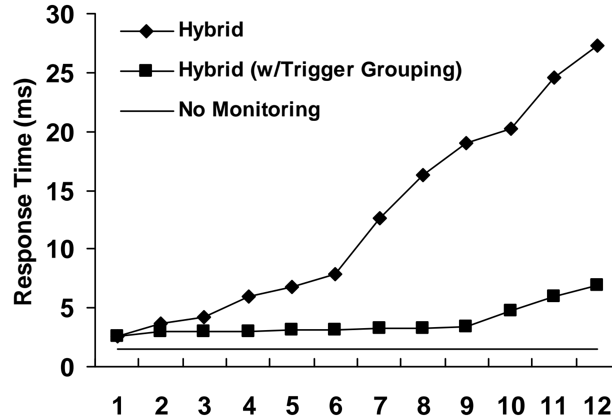

Figure 5.6: The scalability of the hybrid approaches to event detection on views. The values on the x-axis represent the total number of protective policies being enforced on the database. For example at 7, Policies 1 through 7 are all being enforced at the same time.

dealing with a large number of policies, triggers would be able to accommodate the monitoring of all protective views. Note that the policies where triggers incur a high overhead are deliberately introduced as the last three policies in the mix. Thus the figure seems to favor the use of triggers for record monitoring. However the total cost associated with monitoring all policies using triggers is far greater than that of using materialized views. This is simply because of the high cost of repetitive re-computation of aggregates over a large amount of data. The addition of a single policy (Policy 12), which is intended to protect the aggregated total of all orders as long as the total remains under a certain maximum, makes the use of triggers for view monitoring infeasible. We deliberately delayed introducing this policy in the mix to ensure meaningful presentation of results. Had this policy been introduced as the first policy in the mix, the results would have been very different. The more important observation is that, neither triggers nor incremental view maintenance, can in general work well for monitoring a broad variety of views. The overhead incurred for monitoring all policies at the same time was a factor of 30 using incremental computation approach and roughly a factor of 1350 for triggers when compared against no monitoring of updates.

The results in Figure 5.6 demonstrate that simply choosing the correct and more suited event monitoring mechanism for each policy can substantially reduce the cost associated with monitoring critical views. The plain hybrid strategy, which simply chooses between triggers or materialization, performs better than either technique

المنارة للاستشارات
www.manaraa.com

alone. However it still suffers from having to instantiate a large of number triggers which greatly reduces its scalability. Since the TPC-H schema is relatively compact, and the fact that a large number of our policies revolve around the order and lineitem tables, trigger grouping provides a much more significant improvement in response times by minimizing the number of trigger invocations per update on a base table.

We note that there are bound to be differences between our tests and real world scenarios. However we claim that these differences will not degrade performance in a large number of cases. The foremost difference will be that of a non-random model for updates where all protected tuples are not equally likely to be modified. As noted earlier many businesses do not actively modify temporally stable records (for example very old purchase orders). Consequently the use of temporal/range based indexing and partitioning may substantially reduce monitoring costs. Second we note that our tests were done using unrealistically high record coverage and while certain database applications such as those keeping track of medical histories will be highly regulated, typical coverage can be expected to be lower than what we tested against.

In practice we recommend that the decision to use a particular strategy for monitoring views be made by the query optimizer after considering the expected number of policy violations, expected number of relevant updates, level of temporal stability exhibited by records and types of views to be monitored. Given a particular workload, modern database systems are able to recommend materializations of views that can improve query performance. Much of the infrastructure to measure the cost and benefit of incremental computation of views versus re-computation costs of queries already exists. Therefore we believe that by using these existing features a "retention policy advisor" can be built such that given a set of record definitions, protective policies on records and an expected workload profile, the best monitoring mechanism can be easily determined. We acknowledge that as the expressiveness of critical views and the complexity of actions specified (perhaps as stored procedures) increases, so will the overhead of having a view/tuple protective data retention framework. The work done by Blakely et al. in the area of views attempted to detect irrelevant updates statically by only examining the view definition and a given update. However more than two decades' research into views has made modern database systems capable of exploiting a large range of available information, such as functional dependencies and materialized views, for detecting irrelevant updates. Although there is the remote possibility of encountering the worst case cost of re-computation of a critical view at every modification to the database, we argue that no other existing mechanism in relational databases (such as triggers or access

www.manaraa.com

control features) will in general be able to avoid such costs.

Finally there is the cost of static analysis of policies. We believe that tasks such as detection of conflicts, and proving termination properties for destructive actions can largely be mechanized. Even though the run time of most algorithms to accomplish this will be exponential, and human intervention may still be required, these costs will be one-time only. Laws pertaining to retention do not change very often, thus when a stable set of record definitions and policies are created, they will rarely require re-verification.

المنارة للاستشارات
www.manaraa.com

للاستشارات المنارة
www.manaraa.com

Chapter 6

The Broader Picture in Records Retention

We conclude this thesis by comparing our proposed framework to existing solutions and then examining several broad-ranging issues related to data retention in and beyond the scope of relational databases. We examine various problems pertaining to retention of distributed and offline data. Possible remedial steps that can be taken to mitigate these problems are then discussed.

6.1 Existing Solutions

There are several software based ECM management solutions that manage a wide variety of records such as emails and documents. Unfortunately, we have not seen a similar solution for records management in relational database management systems. There are several storage management solutions such as Tivoli (IBM) and StorageWorks (HP) that can make regular backups of entire databases and subsequently delete them at specified periods of time. However to the best of our knowledge there exists no published solution nor a comprehensive examination of issues pertaining to records management in relational database systems.

It is evident that industry and academia are well aware of the problems in the store-everything approach with databases. However databases are very much still treated as coarse structures similar to files. Several papers published in technical conferences have stressed the need for proper frameworks for data retention, but none have gone beyond the surface to explain how their vision will be implemented.

المنارة للاستشارات
www.manaraa.com

Privacy policy specification languages such as P3P [AHKS02] and EPAL [And06] also have support for retention tags for data, but they do not address fundamental issues such as how these tags will be used in various situations.

More closely related, but equally vague, descriptions of how retention features in database systems can be implemented are briefly mentioned in the work of Agarwal et al. [AKSX02, ABG+05]. Unfortunately not much is described in these papers other than IBM's long term vision for Hippocratic/Privacy-aware databases. HP's vision for management of privacy and data retention related obligations is described in the academic literature [MT06b, MT05, MT06a]. However the solution that they outline is an attempt to monitor privacy obligations enterprise-wide using an elaborate central obligation monitoring system. It can best be described as a systematic way of scheduling events throughout all corporate data repositories such that the execution of these events will ensure compliance with all privacy obligations. Such a technique is rather naïve and more importantly it is very inefficient. It fails to recognize that checking of policy violations can be best done at the source of the data, and at the same time as updates are applied to the data. Consequently we believe that an external one-size-fits-all solution to enterprise wide privacy obligation monitoring along with storage of the different what-if scenarios to deal with situations, will not scale up for high performance database systems. Although we propose a database oriented record monitoring system, our work is also somewhat orthogonal in the sense that it does not assume existence of pre-defined mechanisms/procedures in a database that are guaranteed to be correct and lead to a privacy friendly state after their execution.

It seems that prior work has generally failed to address the question of what is a record in relational database systems, acknowledge that retention requirements can be both protective and destructive, and observe there are several issues specific to database systems, such as dependencies among data and their retention requirements, that need to be considered. From our review of the literature we believe that there is a common misconception that attaching a timestamp to every piece of data in a system denoting its expiry will solve all problems. Stated differently, this is very much the ECM approach to classifying data into different categories based on the retention obligations. Unfortunately such a solution is clearly infeasible for large-scale relational databases.

Firstly we note that the granularity of data at which a timestamp oriented technique can be applied can not be universally specified. Records retention policies are typically defined by authorities without considering any specific database schema. Consequently the foremost shortcoming of this metadata approach is that it does

المنارة للاستشارات
www.manaraa.com

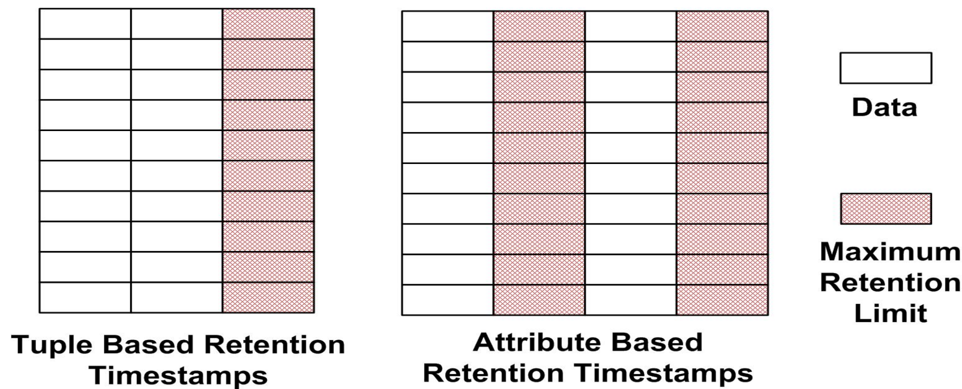

Figure 6.1: Simply associating an expiry timestamp with every piece of data can increase the storage requirements by a large factor.

not answer the question of where these timestamps will come from and how they will be specified. Our framework on the other hand addresses these question directly. Secondly we note that attaching a retention timestamp to every tuple, or even worse to every attribute of every tuple, is a very space inefficient method of keeping track of retention obligations (See Figure 6.1). Such an approach is only workable for simple and well defined objects, such as files, and in situations where storing a timestamp has a relatively insignificant cost compared to the storage of the record. Our proposed solution on the other hand implicitly stores these timestamps as a policy expressed over the structure of records.

Not only does the timestamp approach ignore functional dependencies among individual pieces of data, but it is also blind to the fact that retention obligations may themselves be related to each other. If the time at which a record is to be deleted is dependent on factors such as the last time the record itself was updated, then we run into the additional problems of maintaining and keeping up-to-date the particular retention timestamp in question. Furthermore timestamps themselves do not provide adequate expressiveness for typical temporal retention conditions. Lastly, the motivation for most published work in this area comes from providing enhanced privacy guarantees to customers and external observers. Most research into privacy aims to improve the state of the art in privacy enhancing technologies for the greater good of the public and not the corporate records management team. Therefore the end goal is not compliance with minimum retention periods but rather to ensure that private information is protected and not retained indefinitely in organizational databases. Consequently the problem of managing and enforc-

المنارة للاستشارات
www.manaraa.com

	Email Management	Our Framework
Records	Emails with Attachments	Tuples in the view "R" defined by the expression $\sigma(email \bowtie attachment)$
Protected Records	Emails that have to be retained until some condition is met	Tuples in a critical view of a protection policy on R
Expired Records	Emails that have delete as soon as possible	Tuples in a critical view of a destruction policy on R
Policy Enforcement	Monitoring all emails that are created or destroyed against all policies	Maintenance of critical views and detecting relevant updates

Figure 6.2: A comparison of how records management is accomplished in a typical central email storage/monitoring system against how it would be done in our framework.

ing minimum retention periods for internal operational business records is largely ignored in the literature.

Our proposed framework suffers no such shortcomings. We recognize that unlike ECM systems, which manage files and data objects like email messages, there are no boundaries for records in databases that can be used in all situations. Consequently we have introduced a novel way to tackle the problem such that users can themselves define records and enforce policy actions that they deem fit. Although our proposed framework shares the same fundamental objectives with modern ECM systems (see Figure 6.2 for a comparison) it is much more flexible than any existing mechanism for managed records retention.

6.2 Guaranteed Destruction of Records

6.2.1 Backups and Offline Databases

The problem of indefinite retention of backed up data was first discussed up by Boneh and Lipton [BL96]. They observed that even with the use of a privacy aware systems that delete all expired data, offline backups (for example on tapes) can lead to sensitive information persisting indefinitely. In order to avoid the cost of physically mounting backups and deleting expired records, they proposed that

المنارة للاستشارات
www.manaraa.com

all sensitive information be encrypted before being backed up. When the data expires, simply deleting the encryption key (which itself is not stored alongside the backup) will effectively make the data irrecoverable for all intents and purposes. An examination of the problem of hierarchical key management was presented in their work, and they concluded that this technique can be repeatedly applied, even to backup keys themselves. However the most recent encryption key must be preserved with care. Their solution suggested that this master key must either be physically written down or protected through escrow, and then subsequently destroyed when another key replaces it. Although this technique is very practical, it is unlikely that the problem associated with making backups of the current master key can be resolved. If in a disaster situation the master key is lost, we effectively lose all backups of our data. On the other hand keeping copies of the master key induces risk of accidental retention of that key itself.

6.2.2 Distributed Systems

Data retention issues in distributed environments are far more complex than those for stand-alone systems. We define the retention problem in distributed systems as the problem of *provably* deleting a particular identifiable piece of data throughout a distributed system. The aim is to offer a reasonable probabilistic guarantee that records that may be replicated and/or partitioned over an arbitrary network are deleted when the necessary conditions are met. There are numerous models under which this problem can be studied. For example, in networks where nodes join and leave intermittently, the ability to guarantee with reasonable certainty that a particular delete request will propagate to all nodes has been examined [BCK+06]. Changing various parameters of the network and properties of nodes, such as assigning owners of data, introducing trust models for data exchange, and even including adversaries with limited storage in the network, can all lead to interesting retention scenarios. Unfortunately much of this work is spread across various aspects of computing research. Different research communities have published their results in separate contexts, such as in peer-to-peer data management systems [DHA03], distributed file systems [BCK+06], and of course in distributed database management systems [DGH+87]. As much as we would like to direct the reader of this report to a survey of such papers, there exists no comprehensive review of the literature that unifies work done in distributed data management for enforcing mechanisms for limited data retention.

www.manaraa.com

6.2.3 Unwarranted Data Retention

Several issues in data retention arise simply because the concept of deleting data has a different meaning in different contexts. For example environments such as file systems that support "undeletion" provide a very vague explanation to users of what deletion means. It is a well known fact that deletion in file systems implies nothing more than simple removal of pointers to give the illusion of deletion and available free space. In order to avoid the overhead of writing zeroes to disk, most file systems leave files recoverable, through physical examination of sectors on disks. We believe that similar forensic examination of deletion in various storage systems will reveal equally interesting results.

Most of these cases of unwarranted data retention arise because of the mismatch between user perception and system actions. Furthermore, it is interesting that the problem of unwarranted data retention goes beyond the realm of software and even storage systems. For example, modern day fax machines have convenience features that allow users the ability to re-send faxes that were sent earlier, by retaining them in large storage buffers. Whether users are aware of such features (their presence or their being activated) is rarely a significant issue in the design of user interfaces. While designing user oriented systems, whether they are web browsers or operating systems, convenience and performance have traditionally trumped privacy. The fact that traditional mechanisms for improving performance such as buffering, caching and replication were all designed without considering privacy related issues has very disturbing implications. Consequently we anticipate that a top down examination of various data oriented systems will reveal that unwarranted retention of data is a much bigger problem than it seems. In the context of forensic analysis of database systems, a comprehensive examination of issues was presented very recently by Stahlberg et al. [SML07]. They observed that database systems are notoriously misleading when it comes to deletion of data. The use of deletion bits instead of overwriting with zeroes and persistence of data in transaction logs and indexes are a few of the many problems associated with unexpected data retention in relational database systems.

6.3 Future Work

Apart from examining various problems in distributed systems and sources of inadvertent data retention, there is a wide array of topics in the field of records management that can be explored in the future. Instead of briefly touching on divergent

المنارة للاستشارات
www.manaraa.com

issues such as XML databases and support for retention in other areas, we suggest one critical problem in computer science that desperately needs more attention from database researchers.

A problem which has generally been ignored in relational database systems is the issue of ownership of data and rights over it. Several interesting problems emerge when data from multiple sources/owners is integrated into one database system, especially if these owners want to ensure different (retention) policies on their data. Database systems assume that all data contained within them is owned by one single authority. Typically a system administrator represents that authority and he/she can grant various users rights over data. In situations involving Electronic Data Interchange (EDI) or data import/export between organizations, a database system has no functionality of respecting the policies and rules on it that the original owner wished to enforce.

In the context of privacy preserving systems, this inseparability of data and policies on it is widely known as the *sticky policy paradigm*. The problem is similar to that of Digital Rights Management in the realm of anti music/file sharing technologies. No storage system that we are aware of has adequate support for enforcing privacy policies and supporting interchange of data and policy at the same time. However as people become more aware of the privacy implications of data interchange between large scale database systems, providing a mechanism for supporting ownership and portability of policies in databases will become of extreme importance. Organizations such as the United States Department of Defense have already started taking data exchange issues very seriously and perform regular audits of their sub-contractors' computer systems. Therefore research into this area from a database perspective certainly has value. Once again it is questionable whether attaching an ownership/policy tag to every piece of data will be a workable solution for high performance database systems. Consequently we believe that a better approach would be to develop a framework similar in nature to ours, where records and ownership are defined using a flexible granularity, as the basis for a database system that truly supports the sticky policy paradigm. However there are numerous other issues that arise in EDI, such as ensuring that the set of policies among data exchanging parties are non-conflicting and determining whether policies enforced by one party are stronger or weaker than the other. Even more complications can occur when records are built on top of data jointly owned by different parties.

المنارة للاستشارات
www.manaraa.com

6.4 Summary of Contributions

In this thesis we examined the problem of managed records retention in relational database systems. We recognized that no definition of a record will be universally applicable over relational data, and naïve solutions of expiry-tagging will be far too inefficient to scale up for high performance database systems. Consequently we proposed a novel way of looking at records as relational expressions, which subsequently allowed us to reduce challenges associated with policy enforcement to several well studied problems in database theory. We believe that our methodology of looking at policies beyond the level of rows and attributes makes the task of specifying and controlling policies over complex records much simpler for the average records manager. A framework for monitoring and enforcing data retention policies on view based records was presented, along with a comprehensive discussion of formal properties such as conflicts and termination can be verified. Our framework has the benefit of being not only space efficient but it is also capable of leveraging well known results in database theory to improve performance of records management tasks. From our analysis of a wide range of features in relational systems such as triggers, access control and view maintenance, we believe that our view based framework of policy enforcement is indeed the most practical solution for managed records retention.

www.manaraa.com

Appendix A

Policy Descriptions and View Definitions (DB2)

A.1 Policy 1

Parts 500-525, 999 and 1001 are parts used for nuclear weapons and fuel enrichment. All orders with line items which include the sale of the above parts must never be updated. Other line items in these orders must also be protected.

A.1.1 View Definition

```
CREATE TABLE P1 AS
(
SELECT L2.L_ORDERKEY AS L2OKEY,
L2.L_LINENUMBER L2LNUM,
L1.L_ORDERKEY L1OKEY,
L1.L_LINENUMBER L1LNUM,
O_ORDERKEY,
L2.L_COMMENT,
O_COMMENT,
O_TOTALPRICE
FROM
LINEITEM L1,
LINEITEM L2,
```


المنارة للاستشارات
www.manaraa.com

```
ORDERS
WHERE L2.L_ORDERKEY = O_ORDERKEY
AND O_ORDERKEY = L1.L_ORDERKEY
AND
(
L1.L_PARTKEY = 999
OR L1.L_PARTKEY = 1001
OR (L1.L_PARTKEY >= 500 AND L1.L_PARTKEY <= 525)
)
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P1;
```

A.1.2 Trigger on Orders

```
CREATE TRIGGER NUCLEAR_PARTS
BEFOREUPDATE OFO_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROWNEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM LINEITEM
WHERE L_ORDERKEY = OLD_ROW.O_ORDERKEY
AND (L_PARTKEY = 999
OR L_PARTKEY = 1001
OR (L_PARTKEY >= 500 AND L_PARTKEY <= 525))
)
THEN
ELSE
END IF;
END
```

A.1.3 Trigger on Lineitem

```
CREATE TRIGGER NUCLEAR_PARTS_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
```


المنارة للاستشارات
www.manaraa.com

```
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN
(OLD_ROW.L_PARTKEY = 999
OR OLD_ROW.L_PARTKEY = 1001
OR (OLD_ROW.L_PARTKEY >= 500 AND OLD_ROW.L_PARTKEY <=
525))
BEGIN ATOMIC
END
```

A.2 Policy 2

Protect all orders from Kenya with total price more than 150000.

A.2.1 View Definition

```
CREATE TABLE P2 AS
(
SELECT
O_COMMENT,
O_ORDERKEY,
O_TOTALPRICE,
C_CUSTKEY,
N_NATIONKEY
FROM
ORDERS, CUSTOMER, NATION
WHERE
C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 14
AND O_TOTALPRICE > 150000
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P2;
```


المنارة للاستشارات
www.manaraa.com

A.2.2 Trigger on Orders

```
CREATE TRIGGER KENYAN_ORDERS
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM CUSTOMER, NATION
WHERE C_CUSTKEY = OLD_ROW.O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 14
AND OLD_ROW.O_TOTALPRICE > 150000
)
THEN
ELSE
END IF;
END
```

A.2.3 Trigger on Lineitem

```
CREATETRIGGER KENYAN_ORDERS_LI
BEFORE UPDATE OF L_COMMENT ,L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM CUSTOMER, NATION,ORDERS
WHERE C_CUSTKEY = O_CUSTKEY AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 14
AND O_TOTALPRICE > 150000
AND OLD_ROW.L_ORDERKEY = O_ORDERKEY
)
THEN
```


المنارة للاستشارات
www.manaraa.com

```
ELSE
END IF;
END
```

A.3 Policy 3

Protect orders beyond a large total price of (300,000). Note that this policy implies that we are doubly protecting the Kenyan orders and all large orders

A.3.1 View Definition

```
CREATE TABLE P3 AS
(
SELECT
O_ORDERKEY,
O_COMMENT,
L_COMMENT,
L_ORDERKEY,
L_LINENUMBER
FROM
ORDERS, LINEITEM
WHERE
O_TOTALPRICE >= 300000 AND O_ORDERKEY = L_ORDERKEY
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P3;
```

A.3.2 Trigger on Orders

```
CREATE TRIGGER LARGE_ORDERS
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
```


www.manaraa.com

```
WHEN (OLD_ROW.O_TOTALPRICE > 30000)
BEGIN ATOMIC
END
```

A.3.3 Trigger on Lineitem

```
CREATE TRIGGER LARGE_ORDERS_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM ORDERS
WHERE OLD_ROW.L_ORDERKEY = O_ORDERKEY
AND O_TOTALPRICE > 300000
)
THEN
ELSE
END IF;
END
```

A.4 Policy 4

Parts 3000-10000 are fertilizers that can be used to make explosives. Any order which contains a line item with quantity greater than 10 units of these part should be protected along with the lineitems.

A.4.1 View Definition

```
CREATE TABLE P4 AS
(
SELECT
```


www.manaraa.com

```
O_COMMENT,
O_ORDERKEY,
L_ORDERKEY,
O_TOTALPRICE,
L_LINENUMBER
FROM
ORDERS, LINEITEM
WHERE
O_ORDERKEY = L_ORDERKEY
AND L_PARTKEY >= 3000
AND L_PARTKEY <= 10000
AND L_QUANTITY > 10
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P4;
```

A.4.2 Trigger on Orders

```
CREATE TRIGGER TNT_ORDERS
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM LINEITEM
WHERE L_ORDERKEY = OLD_ROW.O_ORDERKEY
AND L_PARTKEY >= 3000 AND L_PARTKEY <= 10000)
THEN
ELSE
END IF;
END
```


للاستشارات المنارة
www.manaraa.com

A.4.3 Trigger on Lineitem

```
CREATE TRIGGER TNT_ORDERS_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN (
OLD_ROW.L_PARTKEY >= 3000
AND OLD_ROW.L_PARTKEY <= 10000
AND OLD_ROW.L_QUANTITY > 10 )
BEGIN ATOMIC
END
```

A.5 Policy 5

Protect a new category of orders called "ULTRA URGENT". Execute the following query to set a fair number of orders to this priority:

```
UPDATE ORDERS
SET O_ORDERPRIORITY = "-1-ULTRA URGENT"
WHERE O_ORDERPRIORITY = "1-URGENT"
AND MOD(O_ORDERKEY, 5) = 0
```

A.5.1 View Definition

```
CREATE TABLE P5 AS
(
SELECT
O_ORDERKEY,
O_COMMENT,
L_COMMENT,
L_ORDERKEY,
O_TOTALPRICE,
L_LINENUMBER
FROM
ORDERS, LINEITEM
```


المنارة للاستشارات
www.manaraa.com

```
WHERE
O_ORDERPRIORITY = "-1-ULTRA URGENT"
AND O_ORDERKEY = L_ORDERKEY
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P5;
```

A.5.2 Trigger on Orders

```
CREATE TRIGGER URGENT_ORDERS
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN (OLD_ROW.O_ORDERPRIORITY = "-1-ULTRA URGENT")
BEGIN ATOMIC
END
```

A.5.3 Trigger on Lineitem

```
CREATE TRIGGER URGENT_ORDERS_LI
BEFORE UPDATE OF L_COMMENT ,L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM ORDERS
WHERE OLD_ROW.L_ORDERKEY = O_ORDERKEY
AND O_ORDERPRIORITY = "-1-ULTRA URGENT"
)
THEN
ELSE
END IF;
END
```


المنارة للاستشارات
www.manaraa.com

A.6 Policy 6

Protect orders with orderstatus “P”.

A.6.1 View Definition

```
CREATE TABLE P6 AS
(
SELECT
O_ORDERKEY,
O_COMMENT,
L_COMMENT,
L_ORDERKEY,
O_TOTALPRICE,
L_LINENUMBER
FROM
ORDERS, LINEITEM
WHERE
O_ORDERSTATUS = “P”
AND O_ORDERKEY = L_ORDERKEY
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P6;
```

A.6.2 Trigger on Orders

```
CREATE TRIGGER P_ORDERS
AFTER UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN (OLD_ROW.O_ORDERSTATUS = “P”)
BEGIN ATOMIC
END
```


المنارة للاستشارات
www.manaraa.com

A.6.3 Trigger on Lineitem

```
CREATE TRIGGER P_ORDERS_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM ORDERS
WHERE OLD_ROW.L_ORDERKEY = O_ORDERKEY
AND O_ORDERSTATUS = "P"
)
THEN
ELSE
END IF;
END
```

A.7 Policy 7

Create a new Lineitem status "P". Assign this status to every 20th lineitem, then protect all orders which contain such a lineitem: *UPDATE LINEITEM*
SET L_LINESTATUS = "P"
WHERE MOD(L_ORDERKEY, 20) = 0

A.7.1 View Definition

```
CREATE TABLE P7 AS
(
SELECT
O_ORDERKEY,
O_COMMENT,
L_COMMENT,
L_ORDERKEY,
O_TOTALPRICE,
```


المنارة للاستشارات
www.manaraa.com

```
L_LINENUMBER
FROM
ORDERS, LINEITEM
WHERE
L_LINESTATUS = "P" AND O_ORDERKEY = L_ORDERKEY
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P7;
```

A.7.2 Trigger on Orders

```
CREATE TRIGGER P_LINEITEMS
AFTER UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM LINEITEM
WHERE L_ORDERKEY = OLD_ROW.O_ORDERKEY
AND L_LINESTATUS = "P"
)
THEN
ELSE
END IF;
END
```

A.7.3 Trigger on Lineitem

```
CREATE TRIGGER LINEITEM_STATUS_P_LI
BEFORE UPDATE OF L_COMMENT ,L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN (OLD_ROW.L_LINESTATUS = "P")
BEGIN ATOMIC
```


www.manaraa.com

```
END
```

A.8 Policy 8

Protect lineitems with tax amount greater than 4500.

A.8.1 View Definition

```
CREATE TABLE P8 AS
(
SELECT
L_COMMENT,
L_ORDERKEY,
L_LINENUMBER,
O_ORDERKEY,
O_TOTALPRICE
FROM
LINEITEM,ORDERS
WHERE L_ORDERKEY = O_ORDERKEY
AND L_TAX * L_EXTENDEDPRICE > 4500
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P8;
```

A.8.2 Trigger on Orders

```
CREATE TRIGGER HIGH_TAX
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
```


المنارة للاستشارات
www.manaraa.com

```
SELECT * FROM LINEITEM
WHERE L_ORDERKEY = OLD_ROW.O_ORDERKEY
AND L_TAX * L_EXTENDEDPRICE > 4500
)
THEN
ELSE
END IF;
END
```

A.8.3 Trigger on Lineitem

```
CREATE TRIGGER HIGH_TAX_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
WHEN (OLD_ROW.L_TAX * OLD_ROW.L_EXTENDEDPRICE > 4500)
BEGIN ATOMIC
END
```

A.9 Policy 9

All orders from customers from nation 10 (Iran) which total more than 250000 have to be protected for at least 2 years. This policy overlaps with policy #3.

A.9.1 View Definition

```
CREATE TABLE P9 AS
(
SELECT
O_COMMENT,
O_ORDERKEY,
O_TOTALPRICE,
C_CUSTKEY,
```


المنارة للاستشارات
www.manaraa.com

```
N_NATIONKEY
FROM
ORDERS, CUSTOMER, NATION
WHERE
C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 10
AND O_TOTALPRICE > 250000
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
REFRESH TABLE P9;
```

A.9.2 Trigger on Orders

```
CREATE TRIGGER IRANIAN_ORDERS
BEFORE UPDATE OF O_COMMENT ,O_TOTALPRICE ON ORDERS
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM CUSTOMER, NATION
WHERE C_CUSTKEY = OLD_ROW.O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 10
AND OLD_ROW.O_TOTALPRICE > 250000
)
THEN
ELSE
END IF;
END
```

A.9.3 Trigger on Lineitem

```
CREATE TRIGGER IRANIAN_ORDERS_LI
BEFORE UPDATE OF L_COMMENT , L_EXTENDEDPRICE ON LINEITEM
```


المنارة للاستشارات
www.manaraa.com

```
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
IF EXISTS
(
SELECT * FROM CUSTOMER, NATION,ORDERS
WHERE C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND N_NATIONKEY = 10
AND O_TOTALPRICE > 250000
AND OLD_ROW.L_ORDERKEY = O_ORDERKEY
)
THEN
ELSE
END IF;
END
```

A.10 Policy 10

Policies 10-12 involve views with aggregated values. Triggers must incur the cost of executing the specific query at every relevant update. Policy 10 describes a view for an abstract policy where a manager may want to protect the average value of urgent orders.

A.10.1 View Definition

```
CREATE TABLE P10 AS
(
SELECT
COUNT(*) AS CNT,
SUM(O_TOTALPRICE) AS S,
O_ORDERPRIORITY
FROM ORDERS
WHERE O_ORDERPRIORITY = "1 - URGENT"
GROUP BY O_ORDERPRIORITY
```


المنارة للاستشارات
www.manaraa.com

```
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P10;
```

A.11 Policy 11

Another policy involving an aggregate that makes the use of triggers infeasible. The policy designer may want to ensure that the sum of total prices that are below a certain limit from certain customers is protected. A trigger would have to re-compute this query at every relevant update. Active recomputation would only require a lookup on SUM(totalprice) in this materialized view. The special customers in this case are those whose ID is divisible by 50.

A.11.1 View Definition

```
CREATE TABLE P11 AS
(
SELECT COUNT(*) AS CNT,
SUM(O_TOTALPRICE) AS S,
O_CUSTKEY
FROM ORDERS
WHERE MOD(O_CUSTKEY, 50) = 0
GROUP BY O_CUSTKEY
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P11;
```

A.12 Policy 12

An aggregate over all possible orders.

المنارة للاستشارات
www.manaraa.com

A.12.1 View Definition

```
CREATE TABLE P12 AS
(
SELECT COUNT(*) AS CNT,
SUM(O_TOTALPRICE) AS S,
O_ORDERSTATUS
FROM ORDERS
GROUP BY O_ORDERSTATUS
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;
REFRESH TABLE P12;
```


المنارة للاستشارات
www.manaraa.com

References

[ABG+05] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan, and W. Rjaibi. Extending relational database systems to automatically enforce privacy policies. In *ICDE 2005: Proceedings. 21st International Conference on Data Engineering*, pages 1013–1022, Los Alamitos, CA, USA, 2005. IEEE Computer Society Press. 58

[AHKS02] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-P3P privacy policies and privacy authorization. In *WPES '02: Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, pages 103–109, New York, NY, USA, 2002. ACM Press. 58

[AKSX02] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *VLDB*, pages 143–154, 2002. 58

[And06] A. Anderson. A comparison of two privacy policy languages: EPAL and XACML. In *SWS '06: Proceedings of the 3rd ACM Workshop on Secure Web Services*, pages 53–60, New York, NY, USA, 2006. ACM Press. 58

[AWH92] Alexander Aiken, Jennifer Widom, and Joseph M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 59–68, New York, NY, USA, 1992. ACM. 35

[BCK+06] Gal Badishi, Germano Caronni, Idit Keidar, Raphael Rom, and Glenn Scott. Deleting files in the Celeste peer-to-peer storage system. In *SRDS '06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 29–38, Washington, DC, USA, 2006. IEEE Computer Society. 61

المنارة للاستشارات
www.manaraa.com

[BCL89] José A. Blakeley, Neil Coburn, and Per-Åke Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst.*, 14(3):369–400, 1989. 24, 25, 44

[Bet02] C. Bettini. Obligation monitoring in policy management. In *Third International Workshop on Policies for Distributed Systems and Networks*, pages 2–12, 2002.

[BJ98] Jean-Franois Blanchette and Deborah Johnson. Data retention and the panoptic society: The social benefits of forgetfulness. *ACM Policy Conference 1998*, 1998. 6

[BL96] D. Boneh and R. Lipton. A revocable backup system. *Proceedings of the 6th USENIX Security Conference*, pages 91–96, 1996. 60

[BLT86] José A. Blakeley, Per-Åke Larson, and Frank Wm. Tompa. Efficiently updating materialized views. *SIGMOD Record*, 15(2):61–71, 1986. 25

[BM95] Lars Bækgaard and Leo Mark. Incremental computation of time-varying query expressions. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):583–590, 1995. 26

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981. 31

[CCW00] S. Ceri, R. Cochrane, and J. Widom. Practical applications of triggers and constraints: Success and lingering issues (10-year award). In *VLDB 2000: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 254–262, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 40

[Coh06] Sara Cohen. User-defined aggregate functions: Bridging theory and practice. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 49–60, New York, NY, USA, 2006. ACM Press. 25

[CW91] Stefano Ceri and Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB*, 1991. 49

[DFK06] H. Drinan, N. Fontaine, and B. Kesler. News briefs. *IEEE Security and Privacy Magazine*, 4(1):14–16, January-February 2006. 3

المنارة للاستشارات
www.manaraa.com

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM. 61

[DHA03] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 76, Washington, DC, USA, 2003. IEEE Computer Society. 61

[DKM86] Klaus R. Dittrich, Angelika M. Kotz, and Jutta A. Mülle. An event/trigger mechanism to enforce complex consistency constraints in design databases. *SIGMOD Record*, 15(3):22–36, 1986. 40

[Eid06] Thomas Eid. Records retention requirements. *Market Share: Enterprise Content Management Software, Worldwide, 2003-2005*, 2006. 7

[Elk89] C. Elkan. A decision procedure for conjunctive query disjointness. In *PODS '89: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 134–139, New York, NY, USA, 1989. ACM Press.

[Emm89] Peter Emmerson. *How to Manage Your Records: A Guide to Effective Practice)*. Hemel Hempstead, UK: ICSA Publishing Limted, 1989. 14

[Etz93] O. Etzion. PARDES: a data-driven oriented active database model. *SIGMOD Record*, 22(1):7–14, 1993.

[GF05] P. Gama and P. Ferreira. Obligation policies: An enforcement platform. *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks POLICY'05*, 00:203–212, 2005.

[GW76] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.

[HCH+99] Eric N. Hanson, Chris Carnes, Lan Huang, Mohan Konyala, Lloyd Noronha, Sashi Parthasarathy, J. B. Park, and Albert Vernon. Scalable Trigger Processing. In *ICDE*, 1999. 50

المنارة للاستشارات
www.manaraa.com

[HN99] Eric N. Hanson and Lloyd X. Noronha. Timer-driven database triggers and alerters: semantics and a challenge. *SIGMOD Record*, 28(4):11–16, 1999.

[Kel85] Arthur M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *PODS '85: Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 154–163, New York, NY, USA, 1985. ACM. 31

[Kel86] Arthur M. Keller. Choosing a view update translator by dialog at view definition time. In *VLDB '86: Proceedings of the 12th International Conference on Very Large Data Bases*, pages 467–474, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc. 31

[KHS94] G. Knolmayer, H. Herbst, and M. Schlesinger. Enforcing business rules by the application of trigger concepts. In *Proceedings of the Priority Programme Informatics Confrence*, pages 24–30. Swiss National Foundation, 1994.

[KRS04] Orly Kalfus, Boaz Ronen, and Israel Spiegler. A selective data retention approach in massive databases. *International Journal of Management Science*, 32(32):87–95, 2004.

[KS02] G. Karjoth and M. Schunter. A privacy policy model for enterprises. In *15th Computer Security Foundations Workshop*, pages 271–281, 2002.

[KSW03] G. Karjoth, M. Schunter, and M. Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. *Lecture Notes in Computer Science*, 2482(5):69–84, August 2003.

[LL99] S. Y. Lee and T. W. Ling. Unrolling cycle to decide trigger termination. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 483–493, 1999. 35

[LS99] C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[ML85] N. Minsky and A. Lockman. Ensuring integrity by adding obligations to privileges. In *ICSE '85: Proceedings of the 8th international conference*

المنارة للاستشارات
www.manaraa.com

on Software engineering, pages 92–102, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[ML03] Stephen Moore and Lovelace. Records retention requirements. 2003. 5

[MS02] R. Middleton and H. Smith. Data retention policies after enron - damned if you do, damned if you don't? a look at data retention policies in the aftermath of enron. *Computer Law and Security Report*, 18(5):333–337, October 2002. 3, 32

[MT85] Claudia Bauzer Medeiros and Frank Wm. Tompa. Understanding the implications of view update policies. In *VLDB '1985: Proceedings of the 11th International Conference on Very Large Data Bases*, pages 316–323. VLDB Endowment, 1985.

[MT05] M. Mont and R. Thyne. A system to handle privacy obligations in enterprises. In *Hewlett-Packard Internal Technical Report (HPL-2005-180)*, 2005. 58

[MT06a] M. Mont and R. Thyne. Privacy policy enforcement in enterprises with identity management solutions. In *Hewlett-Packard Internal Technical Report (HPL-2006-72)*, 2006. 58

[MT06b] M. Mont and R. Thyne. A systemic approach to automate privacy policy enforcementin enterprises. In *PET 2006: 6th Workshop on Privacy Enhancing Technologies*, 2006. 58

[SD95] Eric Simon and Angelika Kotz Dittrich. Promises and realities of active database systems. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 642–653, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 40

[SML07] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine. Threats to privacy in the forensic analysis of database systems. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 91–102, New York, NY, USA, 2007. ACM. 62

[SNS06] Feng Shao, Antal Novak, and Jayavel Shanmugasundaram. Triggers over nested views of relational data. *ACM Transactions on Database Systems*, 2006. 49, 50

المنارة للاستشارات
www.manaraa.com

[Swe02] Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, 10(5):557–570, 2002. 5

[UB04] Ekweozor Ugonwa and Theodoulidis Babis. Review of retention management software systems. *Records Management Journal*, 14(2):65–77, 2004. 14

[vdVS93] Leonie van der Voort and Arno Siebes. Termination and confluence of rule execution. In *CIKM '93: Proceedings of the Second International Conference on Information and Knowledge Management*, pages 245–255, New York, NY, USA, 1993. ACM. 35

[Wid96] J. Widom. The starburst active database rule system. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):583–595, 1996. 40

[YC00] Zawiyah M. Yusof and Robert W. Chell. The records life cycle: An inadequate concept for technology-generated records. *Journal of Information Development*, 16(3):135–141, 2000. 14

[YW98] Jun Yang and Jennifer Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *EDBT '98: Proceedings of the 6th International Conference on Extending Database Technology*, pages 389–403, London, UK, 1998. Springer-Verlag.

المنارة للاستشارات
www.manaraa.com