Efficient Similarity-based Operations for Data Integration

Dissertation zur Erlangung des akademischen
Grades Doktoringenieur (Dr.-Ing.)

angenommen durch die Faultät Informatik der
Otto-von-Guericke-Universität Magdeburg

von Diplominformatiker Eike Schallehn,
geboren am 22.9.1971 in Schönebeck

Gutachter:
Prof. Dr. Gunter Saake
Prof. Dr. Kai-Uwe Sattler
Dr. Ralf-Detlef Kutsche

Promotionskolloqium in Magdeburg
am 26. März 2004

Zusammenfassung

Das Forschungsfeld der Datenintegration ist ein Gebiet mit wachsender praktischer Bedeutung, besonders unter Berücksichtigung der wachsenden Verfügbarkeit großer Datenmengen aus mehr und mehr Quellsystemen. Entsprechend beinhaltet gegenwärtige Forschung die Lösung von Problemen zur Beseitigung von Konflikten auf der Datenebene, welche in dieser Dissertation betrachtet werden.

Die Behandlung von Diskrepanzen in Daten ist immer noch eine große Herausforderung und zum Beispiel relevant zur Beseitigung von Duplikaten aus semantisch überlappenden Datenquellen als auch zur Verbindung komplementärer Daten aus verschiedenen Quellen. Entsprechende Operationen können meist nicht nur auf Wertegleichheit basieren, da nur in wenigen Fällen über Systemgrenzen hinweg gültige Identifikatoren existieren. Die Verwendung weiterer Attributwerte ist problematisch, da fehlerhafte Daten und unterschiedliche Darstellungsweisen ein häufiges Problem in diesem Kontext sind. Deshalb müssen solche Operation auf der Ähnlichkeit von Datenobjekten und -werten basieren.

Der Begriff der Ähnlichkeit ist selber problematisch bezüglich seiner Verwendung und der Grundlagen seiner Bedeutung. Erfolgreiche Anwendungen haben oft eine sehr spezifische Sichtweise auf Ähnlichkeitsmaße und -prädikate, welche einen eingeschränkten Fokus auf den Kontext der Ähnlichkeit im gegebenen Szenario widerspiegelt. Um ähnlichkeitsbasierte Operationen für die Datenintegration bereitzustellen, benötigen wir eine umfassendere Sichtweise, die auch geeignet ist, um zum Beispiel verschiedene generische und angepaßte Ähnlichkeitsmaße, die in einem gegebenen Datenintegrationssystem anwendbar sind, zu kombinieren.

Dieser Probleme wird sich in der vorliegenden Dissertation angenommen, indem ähnlichkeitsbasierte Operationen entsprechend einem leichtgewichtigen, generischen Rahmen bereitgestellt werden. Die ähnlichkeitsbasierte Selektion, der Verbund und die Gruppierung werden bezüglich ihrer allgemeinen Semantik und besonderer Aspekte der zugrundeliegenden Ähnlichkeitsrelationen diskutiert. Entsprechende Algorithmen für die Datenbearbeitung werden für materialisierte und virtuelle Datenintegrationsszenarien beschrieben. Implementierungen werden vorgestellt und bezüglich der Anwendbarkeit und Effizienz der vorgestellten Ansätze evaluiert.

Auf der Prädikatebene konzentriert sich die Dissertation auf die Ähnlichkeit von Zeichenketten, und zwar basierend auf der Levenshtein- oder Editierdistanz. Die effiziente Bearbeitung von ähnlichkeitsbasierten Operationen hängt in erster Linie von der effizienten Auswertung von Ähnlichkeitsprädikaten ab, was für Zeichenkettenähnlichkeit basierend auf Indexunterstützung in materialisierten und durch Preselektion in virtuellen Integrationsszenarien dargestellt wird.

Efficient Similarity-based Operations for Data Integration

Eike Schallehn

March 29, 2004

Abstract

The research field of data integration is an area of growing practical importance, especially considering the increasing availability of huge amounts of data from more and more source systems. According current research includes approaches for solving the problem of conflicts on the data level addressed in this thesis.

Dealing with discrepancies in data still is a big challenge, relevant for instance during eliminating duplicates from semantically overlapping sources as well as for combining complementary data from different sources. According operations most often cannot only be based on equality of values, because only in rare cases there are identifiers valid across system boundaries. Using other attribute values is problematic, because erroneous data and varying conventions for information representation are common problems in this field. Therefore, according operations have to be based on the similarity of data objects and values.

The concept of similarity itself is problematic regarding its usage and foundations of its semantics. Successful applications often have a very specific view of similarity measures and predicates that represent a narrow focus on the context of similarity for this given scenario. To provide similarity-based operations for data integration purposes requires a broader view on similarity, suitable to include for instance a number of generic and tailor-made similarity measures useful in a given data integration system.

These problems are addressed in this thesis by providing similarity-based operations according to a small, generic framework. Similarity-based selection, join, and grouping operations are discussed regarding their general semantics and special aspects of underlying similarity relations. According algorithms suitable for data processing are described for materialised and virtual integration scenarios. Implementations are given and evaluated to prove the applicability and efficiency of the proposed approaches.

On the predicate level the thesis is focused on string similarity, namely based on the Levenshtein or edit distance. The efficient processing of similarity-based operations mainly depends on an efficient evaluation of similarity predicates, which is illustrated for string similarity based on index support in materialised and pre-selection in virtual data integration scenarios.

Contents

List of Figures

Chapter 1

Introduction

The ever-growing importance of information in our life, described by the debatable notion of the Information Society and mainly driven by the tremendous success of new technologies like the Internet and the Web, has not only changed the way we live but also the direction of information and computer science. The Information Society was defined by the IBM Community Development Foundation in the 1997 report [Fou97] as follows:

> A society characterised by a high level of information intensity in the everyday life of most citizens, in most organisations and workplaces; by the use of common or compatible technology for a wide range of personal, social, educational and business activities, and by the ability to transmit, receive and exchange digital data rapidly between places irrespective of distance.

And yes, the availability of general and specialised information has increased by magnitudes. More information is stored in more local information systems in companies, institutions, and by private persons. Often these information are made globally available via the Internet. Heritage information like books, newspapers, music, cultural assets, card indexes, and more stored for centuries on other media are conditioned and transformed to digital media for better availability. Information has gained a higher reputation as a productivity factor, making the notion of information as commodity more common. Trading almost all kinds of goods using information technology became know as eCommerce and is nowadays common practice. In the private sector new information technologies lead to new levels in communication and entertainment. And though the turbulences of the early phases during the mid-1990s have calmed down a little bit, the trend of an ongoing deeper penetration of our life with information and according technologies will certainly persist.

But, while there is more and more information available from more and more sources, the actual information needs of users are and will remain constrained by human abilities of perception. Actually, we are only able to process a very limited amount of data required to cope with a certain task. So the best profit we can make from the wealth of information available is the improved likelihood of finding information that suit our needs. The downside of a high level of availability of information is the effort required to find or condense that small piece of information users are after.

Nevertheless, the above conflict to some degree marked a turning point for some research areas in information and computer science. The research focus shifted from *how to make information available* towards *how to make information useful*. For years, the main concern in research on data management and information systems was to efficiently store, access, and transfer data. After these questions for the most part were solved, enabling us to provide information with a quality and quantity that some years earlier only few would have dared to dream of, other questions became more important: Where can I find needed information? How can I access these information? How can I make use of the load of information provided? How can I decide if it is even useful? How can I present the information in a way that suits my needs? In other words, technologies to map the huge quantity of available information to the actually small information need were required.

One of the main problems with the created information jungle is the great number of different sources we may use or even have to use to fulfil our information requirements. This is especially true in global scenarios like the Web, where even experts may not be aware of all relevant sources and the kind and quality of information they provide. While this is considered less problematic in local scenarios, it is still a difficult problem to fully understand the meaning and relationships of information stored in different systems. And in addition, in both scenarios we have to deal with a multitude of technologies, interfaces, languages, etc. on various technical levels to access the systems of interest. Though standards exist, there are a number of them for each application and compliance is often limited.

These problems are addressed in the area of information system and data integration, where the main focus is on concepts for resolving conflicts on different levels of data access and data representation between heterogeneous information systems. The main objective of the integration task is to provide the user a single point of access to relevant information from several sources. To achieve this, data can either be gathered and physically stored in one place, or an integrated system may provide a virtual integration by providing a unified access, internally mapping queries to and results from the originate systems.

Though there are no generally agreed upon dimensions of integration, in this thesis physical versus virtual integration is used as an important distinction. For a coarse classification a further characteristic is the degree of stucturedness in the data to be integrated, ranging from highly structured data found for instance in business information systems based on relational databases, to Web sites representing weakly structured or unstructured documents. Furthermore, the local and organisational scope of the integration is used as a dimension, i.e. whether the information sources exist in a more local, e.g. a company, or global context, e.g. the Web. The scope dimension has a significant impact on a number of important aspects of an integrated systems like the autonomy of the source systems, the degree of data-level conflicts, performance considerations, etc.

From a very broad perspective, materialised integration systems range from data warehouses (highly structured, data, local source systems) that store condensed business data mainly for usage by the management, to Web search engines (unstructured data, global source systems), which gather textual information to allow a search over web documents providing only links to the sources as a result. Virtual integration systems include meta search engines (semi-structured data, merely global source systems) offering integrated access to databases available over the Web serving a special purpose. Federated databases (highly structured data, merely local source systems) are less common, mostly due to the often high complexity of this kind of integration and the very few commercial tools and solutions available, e.g. the IBM Information Integrator.

To somewhat narrow the focus of this thesis, the research presented here deals mostly with structured data as found in databases, as well as mediated and federated systems. Semi-structured data is considered, as far as a schema for querying and presenting the data can be derived, though the process of finding and using this schema is not a key issue in this thesis. Distinctions along the dimensions of physical/virtual and global/local integration have a profound impact on approaches presented within this thesis, even causing different approaches suitable in respective scenarios.

The integration approaches mentioned so far, and others discussed later on, solve some of the pressing problems mentioned above. A user no longer has to worry, where to find the information and how to access them. Relying on the designers, implementors, and administrators of an integrated system, the information is now available in one place – physically or virtually – and using one uniform representation.

But considering only the access aspects of integration does not fully solve the problem of providing information according to user needs. The amount of information available is still huge, even bigger when data sets from various sources are combined. The content of various sources may overlap, presentations of data values may vary due to different conventions, and finally, erroneous data is a common problem in local databases making their integration even harder. Furthermore, relationships within data sets, locally expressed using unique identifiers, cannot easily be resolved, because these identifiers very often do not carry any meaning in a global scope.

Hence, making it possible to work with an integrated set of data itself is a difficult part of the integration process, requiring new approaches in data processing, and thus, inspired research in various directions. The motivation of the research presented in this thesis derives from the high degree of vagueness in integrated data sets existing due to the previously mentioned reasons. One way to deal with this problem is to introduce new operations that explicitly address imprecise, erroneous, and only weakly related data to present a more consistent or condensed view to the user.

To provide such operations, the concept of similarity gained interest in data integration and data management in general. Instead of relying on the equality of data values, which forms the basis of data models and according operations in current data management, data can be queried, linked, condensed, or cleaned based on similarity to or between values or more complex objects. Actually, here the notion of similarity is not without problems, because we may talk about real similarity between different objects, as well as similarity of different representations of the same real-world object. The former can apply where data has to be condensed, e.g. for analytical processing, or linked according to some loose relationship. The latter is a very common problem in data integration, known as entity identification, record linkage, or the same-object problem, where different sources often provide overlapping information on objects in a common scope of interest.

The term similarity itself raised some interesting questions, because its usage and definition in sciences like psychology, mathematics, philosophy, etc. varies widely, sometimes even within one area. Furthermore, while equality is a relationship independent of the application domain, a similarity relationship or a similarity measure in most cases is specific to the application, to the objects, or actual attributes to be compared. The current usage of the term in computer science, having its roots in fields like information retrieval and knowledge-based systems, most often has a very narrow focus. This may be suitable in some applications, but to efficiently support data integration scenarios requires a closer look at various aspects of similarity.

So, to provide new operations based on similarity we have to consider the characteristics of the relationship, and, based on this, integrate the operations in a framework for similarity-based query processing. Building on this general view, the special requirements in certain integration scenarios, e.g. as part of a middleware in a virtual integration or as an extension to database management systems for a physical integration, have to be considered. A further key aspect of offering similarity-based operations is their efficiency. At first, this is due to the nature of similarity, which in general requires more effort to decide about its existence between objects than equality. Secondly, the target integration scenarios, as previously mentioned, often have to deal with huge amounts of data gathered from numerous sources, this way exceeding the requirements of conventional query processing.

1.1 Motivation

In the past few years, there has been a great amount of work on data integration. This includes the integration of information from diverse sources in the Internet, the integration of enterprise data in support of decision-making using data warehouses, and preparing data from various sources for data mining. As a motivating example in this section requirements of a mediator-based integration scenario are outlined. The system is intended to provide integrated information on cultural assets drawn from various systems available over the Internet.

Some of the major problems in this context – besides overcoming structural conflicts – are related to overcoming conflicts and inconsistencies on the data level. This includes the elimination of duplicate data objects caused by semantic overlapping of some sources, as well as establishing a relationship between complementary data from these sources. Both aspects are illustrated in the following.

The implementation of operations dealing with conflicts on the data level has a significant difference to usual data management operations: only in some rare cases we can rely on equality of attributes. Instead we have to deal with discrepancies in data objects representing the same or related real-world objects which may exist due to input errors or simply due to the autonomy of the sources. Furthermore, the amount of data to be processed in integration scenarios can be equal to or even greater than from a single source, so, efficiency of the implementation becomes a critical issue. While for this motivating example only the principal requirements are outlined, the focus of this thesis will be on providing the required operations in a way that suits the probably large data volumes and user expectations regarding the performance of an integrated system.

As an example for a similarity join consider an information system on art objects providing information for instance on paintings and their creators. One source may provide a plain collection of these items, but we intend to present additional biographical information on the artists given by a standard catalogue integrated from another source. The example shown in Figure 1.1 demonstrates three problems common in this application domain.

First, due to language issues a number of different spellings or transcriptions of names may exist, like in the case of ‘Albrecht Dürer’ or ‘Ilya Repin’. Secondly,

Paintings

Artist	Title
Ilja Repin	Barge Haulers on the Volga
Vincent vanGogh	Drawbridge with Carriage
Albrecht Duerer	A Young Hare
El Greco	View of Toledo
…	

Artists

Name	Birth	Death	…
Albrecht Dürer	1471	1528	
Vincent van Gogh	1853	1890	
Ilya Repin	1844	1930	
Dominico Theotocopuli	1541	1614	
…			

Title	Artist	Birth	Death	…
Barge Haulers on the Volga	Ilya Repin	1844	1930	
Drawbridge with Carriage	Vincent van Gogh	1853	1890	
A Young Hare	Albrecht Dürer	1471	1528	
View of Toledo	El Greco	1541	1614	
…				

Figure 1.1: Example data and result for a similarity join

a common problem in many application domains are inconsistencies due to typing errors, like in this case the incorrect writing of ‘Vincent van Gogh’. Whereas both these problems could be handled by string similarity, the problem of pseudonyms – or more generally synonyms – as demonstrated by the example artist name ‘Dominico Theotocopuli’, better known to the world as ‘El Greco’, can be solved by applying thesauri during the join on the artist name.

Efficiently performing such a similarity join in a locally materialised database itself is a challenging task and topic of current research. This includes for instance results presented in Chapter 5 of this thesis. Yet, we assumed a virtually integrated scenario where the data resides in different Web databases accessible only through possibly very limited query interfaces. In this case, finding the correct entry for an artist for each painting based on possibly conflicting representations of their names is an even harder problem. This issue is addressed in Chapter 6 of this thesis.

Figure 1.2 demonstrates another problem during data integration, namely the identification and reconciliation of tuples representing the same real-world entity. Assume the input relation of the according operation represents the combined information on paintings from a number of source systems, which may overlap semantically and provide incomplete or imprecise information. In addition to the problems mentioned above, the example illustrates that a complex similarity description involving a number of attributes is required. Conflicts between data values may appear in some fields like the artist name, the title of the painting, or the year of creation, and maybe there are conflicts in all of them.

Furthermore, we have to deal with the fact that more than two tuples may represent the same object, and among these representations may exist varying degrees of similarity. Yet, all of them have to be identified to relate to the same real-world

Title	Artist	Year
Resurrection	El Greco	1579
Resurrection	Dieric Bouts	1460
…		
The Holy Trinity	El Greco	1577
The Holy Trinity	El Greco	16th cen.
…		
Self-Portrait at 28	Albrecht Dürer	1500
Self-Portrait at 28	Albrecht Duerer	
Self Portrait at 28	Albrecht Dürer	1500
…		
Fifteen Sunflowers	Vincent van Gogh	1888
Fifteen Sunflowers	Vincent van Gogh	1889
…		

Title	Artist	Year
Resurrection	El Greco	1579
Resurrection	Dieric Bouts	1460
…		
The Holy Trinity	El Greco	1577
…		
Self-Portrait at 28	Albrecht Dürer	1500
…		
Fifteen Sunflowers	Vincent van Gogh	1888
Fifteen Sunflowers	Vincent van Gogh	1889
…		

Figure 1.2: Example data and result for similarity-based duplicate elimination

entity, so, we have to provide means to establish a single representation of identified objects, which for instance can be done based on additional information on data quality of the integrated sources.

Finally, the example shows that decisions based on complex similarity conditions are not trivial. Though the data on the paintings by Vincent van Gogh may look alike, the according tuples actually represent two different paintings. Hence, the design of similarity predicates and complex similarity conditions as part of the design of the integrated system is a complex task involving the analysis of falsely identified and falsely unidentified objects.

Duplicate elimination is not only required during the integration of query results in virtual integration, but it is also a sub-task of data cleaning in materialised scenarios that comprises further tasks for improving data quality like transformation, outlier detection etc. Assuming SQL-based integration systems, the natural choice for duplicate elimination is the **`group by`** operator using the key attributes of the tuples in combination with aggregate functions for reconciling divergent non-key attribute values. However, this approach is limited to equality of the key attributes – if no unique key exists or the keys contain differences, tuples representing the same real-world object will be assigned to different groups and cannot be identified as equivalent tuples. To base the grouping on similarity implies an at least atransitive similarity relation, which has to be dealt with during query processing. These problems are addressed in Chapter 5 of this thesis.

1.2 Structure of the Thesis

The work presented in this thesis is structured with the general intention to provide a reader having a solid comprehension of database and information systems all the necessary information to fully understand the scope and contents of the described research results. Literature references are used to refer to sources of given descriptions or to satisfy further interest in mentioned topics beyond the scope necessary to understand the content of this thesis.

Chapters 2 and 3 will give overviews of the two underlying areas of data integration and research on similarity. This includes positioning this work according to related research and introducing the vocabulary of concepts used throughout the thesis. Chapter 4 has a bridging function between the foundations presented in the two previous chapters and the own contributions which are described in the later Chapters by providing an own view on similarity-based operations. Then, the main research results are described in Chapters 5 and 6. Accordingly, the structure of the thesis in more detail is as follows.

After this short introduction to the motivation, structure, and contributions of the thesis, Chapter 2 gives an overview of data integration based on the current state of the art. Typical aspects of data integration and resulting problems are introduced based on the commonly considered characteristics of heterogeneity, distribution, and autonomy. Important aspects of according research fields like schema integration and query processing in distributed, heterogeneous environments are shortly described. Then, major approaches like *Federated database management systems*, *Mediators*, and *Data Warehouses* are positioned according to the previously introduced characteristics and related to the contributions of this thesis.

In Chapter 3 an overview of concepts of similarity is given, by first describing the importance, the research background, and some fundamental problems with the comprehension of similarity. Then, terms such as similarity measures, relations, and predicates are defined. The common understanding of similarity measured using distances in metric space as well as the according properties, problems, and implications are discussed. Because the work presented in the later chapters is mainly focused on string similarity predicates, these and related aspects are introduced separately.

As previously mentioned, Chapter 4 can be seen as an introduction to the own contributions of the thesis by providing the framework for the operations described in the latter chapters and introducing the problems at hand during their implementation. For this purpose, the possible levels of similarity support in data management solutions are discussed and the focus of the thesis is fixed accordingly. Then, the semantics of similarity predicates as well as operations, namely selection, join, and grouping, are specified.

Chapter 5 and 6 are the chapters describing the main contributions of the thesis. Both chapters can be distinguished by the kind of integration scenario they target. Results presented in Chapter 5 described operations where the data to be processed is materialised, either because a materialised integration approach – like for instance a Data warehouse – was used, or because the operations work on temporarily materialised, intermediate results. Chapter 6 describes an approach generally applicable in virtual integration scenarios, where global similarity predicates can be transformed for evaluation during distributed query processing.

The approach presented in Chapter 5 is based on string similarity predicates and their efficient processing applying tries. Accordingly, the implementation of the join and grouping operations are described for such predicates or more complex similarity conditions. Furthermore, the implementation was evaluated and further aspects and applications were considered.

Chapter 6 follows a different approach by considering the evaluation of string similarity predicates during distributed query processing with source systems providing limited query capabilities. For this purpose, query predicates are transformed as part of query re-writing based on gathered substring selectivity statistics to grant efficiency. Furthermore, implementations for the selection and join operations are outlined and their efficiency is evaluated.

In Chapter 7 the thesis is concluded by a summary, and an outlook on directions of possible further work is given.

1.3 Contributions of the Thesis

As mentioned in the previous section, the main contributions of this thesis are described in the Chapters 4, 5, and 6. The novel aspects outlined in these chapters are described here in more detail. Furthermore, some research results were previously published and are listed here with the respective references. Results of joint work with Ingolf Geist are included for reasons of completeness of the description of the approaches. This is marked accordingly within the chapters and furthermore pointed out in the following short description.

Chapter 4 – Similarity-based Operations: while the most part of this chapter describes foundations of similarity-based operations that were used accordingly in previous approaches, the description of the semantics of a similarity-based grouping operation is new. It is therefore described in more detail and includes a thorough discussion of dealing with atransitivity which may occur due to the usage of similarity predicates. Furthermore, complex similarity conditions and special aspects of similarity relations are most often neglected in related research. The according research results were previously published for instance in [SSS01] and in [SSS02].

Chapter 5 – Similarity-based Operations for Materialised Data: the chapter describes novel algorithms for implementing similarity-based join and grouping operations in materialised data integration scenarios. The operations were implemented accordingly as part of a mediator query engine and, alternatively, using the extensibility interfaces of the database management system Oracle8i. Because efficient support for string similarity is often required in data integration, an approach for index-based approximate string matching described in [SM96] by Shang and Merret was used, and is the first application of such an index in the context of providing similarity-based operations. To prove the applicability and investigate aspects of using the index-supported string similarity predicate, the results of the evaluation regarding the implementation based on Oracle8i are described. Furthermore, the application of extended aggregation functions for data reconciliation is outlined. Finally, aspects of designing and adjusting similarity measures based on similarity distributions within data sets are discussed. These results were previously published for instance in [SSS04] and in [SS03].

Chapter 6 - Re-writing Similarity-based Queries for Virtual Integration: in this chapter a novel approach for processing operations based on string similarity predicates during distributed query processing in virtual data integration is introduced. To the best of the authors knowledge, there is no other approach targeting the same problem for this application scenario. The processing of the operations is based on pre-selection strategies mapping string similarity predicates to disjunctive substring predicates which are suitable for source systems with limited query capabilities. To grant efficiency of this approach, the mapping must be based on substring selectivity information. This mapping is described in detail. An according management of statistic information on substrings and the estimation of selectivities described is based on joined work with Ingolf Geist, as well as the evaluation of corresponding aspects. Furthermore, based on predicate mappings the implementations of similarity-based selection and join operations are described. Finally, the approach is evaluated regarding applicability and performance, the latter measured in terms of the quality of the created pre-selections. The approaches presented in this chapter represent the newest research results of the thesis and were therefore not previously published.

To summarise, the work presented in this thesis targets the inclusion of similarity-based concepts into data processing in integration scenarios. This problem is addressed on a predicate and operational level. Operations based on possibly complex similarity conditions are introduced, suitable for a wide range of applications. Aspects of the implementation of such operations are described for materialised and virtual integration scenarios. For evaluation purposes the focus was on string similarity predicates, because there is a general lack of support for these in current data management as well as only partial solutions provided by current research.

Chapter 2

Data Integration Approaches

The integration of data sources and information systems has gained a growing interest in research and practice over the last twenty years. As outlined before, this development was driven by the growing number of systems in a local and global scope like enterprises and the World Wide Web, respectively, and the growing data volume within these systems. While the problem of integration is addressed on various levels, in this thesis we will focus on data integration. Application integration, which is a current topic most of all in enterprise scenarios, is not addressed here. This chapter will give a short overview of problems and approaches in data integration necessary to position the presented contributions.

2.1 Introduction

While database systems were intended to have an integrative character themselves by providing a storage solution for many local applications, the integration aspect never played the designated role. The main reasons for this were the diversity of different systems satisfying special requirements, organisational and strategic decisions during system installation or implementation, as well as aspects of distribution and connectivity of the embedding infrastructures. The result were insular solutions tailor-made for specific tasks, limited user groups, and content of a constricted scope.

When the potential need of new applications for more general tasks, more and different user groups, and broader content scopes became obvious shortly after database systems were adopted successfully, the idea of data integration was born in the late 1970s. Terms like *federated* and *multi-database systems* were first mentioned by Hammer and McLeod in [HM79] and Adiba and Delobel in [AD77], respectively. Obviously, the task at hand was – and still is – extremely complex, while the actual need to do data integration only grew slowly in the 1980s. At this time, it inspired research, mainly intended to fully subsume the aspects of data integration. Furthermore, the *schema integration* necessary to provide a unified access to heterogeneous data became a first research focus. The work by Sheth and Larson described in [SL90] not only introduced influential system and schema architectures for federated database systems, but also summarised early research and the terminology used in data integration.

In the 1990s data integration finally became a major research topic, driven mainly by the previously mentioned requirements resulting from a better availability of data. As data integration became more relevant in practice, the focus shifted toward architectural issues and query processing. Furthermore, the Web and XML required new ways to deal with unstructured and semistructured data. Distributed, multi-tier, and heterogeneous architectures became more easily manageable with technologies like CORBA, DCOM, and Java. Data Warehouses arose as a very successful data integration application, urgently requiring practical solutions for some problems that were until then only discussed theoretically. Apart from specific solutions in the latter area, IBM's DB2 DataJoiner ([GL94]) – nowadays integrated with research results from the Garlic project ([TAH$^+$96]) and known as the DB2 Information Integrator – was the first successful product for database integration.

In the late 1990s up to current day, new research is moving towards a so-called *semantic integration*, which is incorporating domain knowledge and advanced meta data into the integration process and integrated systems. From an architectural point of view, XML and *Web Services* provide a reasonable infrastructure for integration in a global scope. Furthermore, a number of applications, for instance from the fields of eCommerce, Life sciences, Digital Libraries, etc., drive the implementation of previously theoretic or experimental integration approaches.

Though a great amount of research went into data integration and many aspects were covered over the last decades, the complexity resulting from combining approaches according to the requirements of real-world applications is still huge. Therefore, applications of data integration are often very limited in their functionality and can only partially be supported by tools and standard components.

2.2 Characteristics of Data Integration

There are several attempts at giving classifications for data integration approaches according to certain characteristics, criteria, or dimensions. In [SL90] Sheth and Larson introduced the often cited dimensions of distribution, heterogeneity, and autonomy. According to these dimensions they distinguished integrated database systems such as *Multi-database systems* and *Federated database systems* from classical approaches like centralised or distributed databases and further refine the classification of integrated systems.

While on the one hand it can be argued that these dimensions are not orthogonal, such as heterogeneity most often is the result of autonomy, especially design autonomy, there are also a number of more recent integration approaches such as *Data Warehouses* introduced by Inmon and described in [Inm96], *Mediators* introduced by Wiederhold in [Wie92], and others that can hardly be classified considering only these three aspects.

Instead of providing new "dimensions" or a new classification, we will discuss characteristics necessary to position the work presented here and relate it to the relevant integration approaches.

2.2.1 Heterogeneity

The heterogeneity of integrated data sources is by far the most important aspect of data integration, because it causes most of the problems that have to be overcome to provide an integrated access to sources. As such, heterogeneity includes differences of source systems ranging from a hardware level to the semantics of the data to be integrated. Therefore, a number of slightly different classifications exists, sometimes only covering a certain scope of interest, described for instance in [SL90, SP91, Wie93, BKLW99]. Based on these we present a rather rough classification, where one class of heterogeneity may imply heterogeneity on another level, such as the usage of different systems may imply different data models, which provide diverging modelling constructs resulting in diverging schemas.

System or technical heterogeneity: this comprises differences in source systems resulting from the hardware and software infrastructures such as:

- hardware
- networks and their infrastructures
- protocols and middle ware
- database systems and other storage solutions
- data models
- languages and interfaces

These problems can be addressed to some degree based on standard protocols and interfaces. The mapping to different languages and interfaces can on the other hand become very problematic, if for instance interfaces are very limited, such as for databases accessible only through Web interfaces. This problem is relevant for the thesis by mapping similarity queries to standard interfaces as described in Chapter 6.

Schematic heterogeneity: this mainly results from the *Design autonomy* across different systems or their differing data models. Depending on the requirements leading to a certain design and the structural primitives offered by a data model, the same real-world aspect may be represented in various ways. This leads to conflicts outlined for instance by Spaccapietra and Parent in [SPD92] and Kim and Seo in [KS91]. Overcoming these heterogeneities was a major focus of research in schema integration discussed later on. For this thesis for the most part schemas are assumed to be integrated before a further resolution of heterogeneities on the data level takes place, whereas this latter resolution is the main concern here.

Semantic heterogeneity: while the previous two classes of heterogeneity result from decisions regarding the environment local systems run in and regarding the design of a system itself, semantic heterogeneity results from the usage of the system. It concerns different interpretations and meanings of data values, objects, schema elements, and the overall scope of the data in several systems as well as the relations between different interpretations. While semantic heterogeneity on the schema level is addressed during schema integration, this thesis deals mainly with semantic heterogeneities on the data level, where an interpretation is necessary for identifying representations of real-world objects or their relationships. Due to differing representations this interpretation often has to be based on similarity, which is the main topic of this work.

Schematic heterogeneities are addressed through schema integration techniques and according processes to design an integrated schema, which were widely covered by research. Schema integration techniques can for instance be based on assertions between schemas (Spaccapietra et al. in [SPD92] and others) and according view definitions, or advanced modelling concepts in object-oriented data models (for instance *upward inheritance* as described by Schrefl and Neuhold in [SN90] and others). Because schema integration like schema design in general is characterised by several degrees of freedom, quality criteria for schema integration are completeness, correctness, minimality, and understandability.

A further classification of view-based approaches is given by *local as view* versus *global as view* approaches, depending on whether the global schema is defined as a view on the local schemas or vice versa as described by Garcia Molina et al. in [GPQ$^+$94] and Levy et al. in [LRO96], respectively. Both approaches are quite different regarding necessary efforts for query processing and the maintenance of the integrated system. Anyway, all schema integration approaches yield some kind of mapping between local source schemata and a global schema, which is used for query re-writing and result transformation in a virtual integration scenario, or only for transforming extracted data sets in a materialised integration.

A rough classification for schema integration processes is given by the distinction between *top down* versus *bottom up* schema integration, as for instance described by Sheth and Larson in [SL90]. The distinction made in this classification is based on whether the target schema is designed to suite the needs of certain global applications (top down), or if it should fully represent the set of integrated schemas. Another field of interest is the necessary evolution of integrated schemas. As we assume schema integration problems to be resolved in a design phase and before the operations proposed here are applied, a further discussion of this wide field is beyond the scope of this short introduction. For far more detailed overviews we refer to descriptions by Ram and Ramesh in [RR99] in the collection edited by Elmagarmid et al. [MRJ99], Conrad in [Con97] (in German), Rahm and Bernstein in [RB01], Özsu and Valduriez in [ÖV99], Batini et al. in [BLN86], and Sheth and Larson in [SL90].

Closely related to heterogeneity is the demand for transparency in the resulting integrated system. Related to data integration transparency refers to the characteristic that the integrated access should be provided through a unified interface hiding all the previously mentioned heterogeneities from the user. In a very strict sense, when accessing the integrated data set the user should not be aware of

- the origin of the data,
- any aspect related to retrieving the data,
- necessary transformations or condensations, and
- the reconciliation of possible conflicts.

Contrary to the need of transparency in some integration applications there is a demand for the traceability of data, i.e. the user may have an interest in some of the aspects mentioned above. Typical questions can be: where does the data come from? Are there any costs involved or how long does it take to retrieve the data? How were the data changed on the way or are there more details available when accessing the separate systems? A typical example where traceability plays an important role are Data warehouses. In an ideal solution, information regarding the origin of data and performed transformations is stored as metadata within the warehouse with the intention to draw plausible conclusions about the quality of derived information.

Whether transparency or traceability is more important in a specific real-world scenario heavily depends on the given requirements for this application. This thesis deals mostly with the resolution of semantic heterogeneities by applying operations for the identification of conflicts and their reconciliation. Because these operations are based on similarity, there is a probabilistic aspect about it, e.g. a decision whether objects in several databases represent the same real-world entity can only be made with a derivable degree of certainty, resulting in a number of false matches and false non-matches. Therefore, traceability may be a strong requirement when such operations are applied. For instance, the user can be informed about the quality of the presented data or even be given details of the original data and drawn conclusions.

2.2.2 Distribution

Another typical characteristic of data integration is the physical distribution of data across various hosts running their own data management systems. This aspect of distribution is shared with more common solutions like distributed databases, where heterogeneity and autonomy do not play such an important role, and the distribution is merely a result of a distribution design targeting improved performance, scalability, and reliability. Contrary to this approach, the physical distribution of data in data integration is the initial situation, where the source systems were primarily designed to serve some local application. Nevertheless, both approaches may share further characteristics like

- transparency regarding several aspects like the physical location of data,
- some aspects of autonomy discussed later on,
- distributed query processing, though with quite different requirements, and
- independence of hardware, software, and network aspects

as described for instance by Date in [Dat90].

Dealing with distributed data sets in data integration spawned two basic approaches characterised by the physical location of the integrated data set.

Materialised integration copies data from the source to an integrated data set managed by one system. Typical representatives of such an approach would be Data warehouses or some archives of Digital libraries. The advantages of this approach mainly result from the possibility to do local query processing on the integrated data set. This is strictly necessary for data intensive operations, as for instance in *Online analytical processing* (OLAP). The major disadvantage comes with the autonomy of the data sources, where even in local scenarios like an enterprise Data warehouse the extraction, transformation, and loading (ETL) of data causes huge organisational efforts in addition to the necessary technical efforts, which include schema integration and are comparable to virtual data integration. Furthermore, the maintenance of integrated materialised data sets is a complex task, involving non-standard operations and application-specific tasks, for instance related to data cleaning. In global scenarios this approach is therefore often prohibitive, unless it relies on standards like for instance *Dublin core* and the *MARC format* for bibliographic metadata and the protocol of the *Open archive initiative* (OAI) for their exchange in the field of Digital libraries.

Virtual integration leaves the data in the component systems and provides an integrated access in terms of a complex distributed query processing, consisting of re-writing every query for the source systems and retrieving, transforming, merging, and reconciling the returned query results. This distributed query processing in heterogeneous environments is a very complex task, especially when complex schemas and large data volumes are involved. Its characteristics are quite different from local query processing, and therefore it is hardly supported by standard data management solutions. Though there are prototypes and commercial products like the previously mentioned IBM Information Integrator, the high degree of heterogeneity in any given application scenario often makes tailor-made solutions a requirement. Contrary to physical data integration, virtual data integration avoids redundancy, provides up-to-date results, and is applicable in scenarios with highly autonomous sources.

As this is not a black and white world, there are mixed approaches. In virtual integration a temporary or persistent materialisation of data is often considered, for instance for caching and archiving query results to improve the efficiency of query processing. On the other hand, in materialised integration it may become necessary to access the the original data to get a more detailed or more recent view of the data, such as provided by *Drill through* operations in Data Warehouses.

Materialised data integration is addressed in Chapter 5 where semantic heterogeneities can be resolved after data is extracted from the source systems as part of the transformation and cleaning steps. As an example, in a Data warehouse environment this is typically done in a *staging area*, which is a physically materialised database itself and is usually managed by one database management system. Therefore, various optimisations are conceivable, such as using index structures, statistics on data, and special algorithms. The approach presented in this thesis applies trie index structures for the evaluation string similarity predicates in a more general framework for similarity-based operations. Furthermore, extensibility interfaces of database management systems – though not standardised – are used to integrate the proposed operations more tightly with the conventional aspects of data processing. All this is possible because

- the full integrated set of data and all its characteristics are known to the system and

- all query processing takes place within one system.

Unfortunately, these two advantages do not hold if the integration is virtual and every query has to be processed by several systems.

The research on distributed query processing in virtually integrated heterogeneous systems has gained some interest since the early 1990s, but there are still a number of issues unresolved. Based on discussions by Sheth and Larson in [SL90], Kossmann in [Kos00], and by Özsu and Valduriez in [ÖV99] the following reasons for the complexity of this task must be considered:

1. Query re-writing as well as the transformation of query results has to be based on schema mappings resulting from schema integration as a design step. These transformations can be complex and may not be supported by standard operations as for instance provided by SQL.

2. Due to association autonomy discussed later on and heterogeneous data models, languages, and interfaces the query capabilities of integrated sources may differ.

3. The costs for evaluating queries and transferring query results may differ between systems. Furthermore, it is hard to estimate these costs, so a global query optimisation is more problematic than in local or homogeneous, distributed scenarios.

4. The transfer of data via networks represents an even more narrow bottleneck than IO operations on secondary storage. Therefore, the minimisation of network traffic becomes a main goal during query optimisation, and special algorithms for pipelined and parallel data processing are required.

5. The lack of statistical data necessary for query optimisation causes further problems for query optimisation.

6. Due to communication autonomy discussed below component system may or may not be available for query processing, or may even disconnect during processing a certain query.

7. Further problems are caused by execution autonomy, where for complex queries a transactional context may be lost when a global query translates to a number of queries to one component system.

The work presented in Chapter 6 of this thesis deals with specialised operations that have to be executed as distributed queries in such an heterogeneous environment. According to the previously listed challenges in distributed query processing in virtual data integration, the two main problems of providing similarity-based operations are query capabilities and efficiency.

Query capabilities: apart from the fact that query interfaces available for data integration may be restricted to a very limited functionality, for instance if only provided through a Web form as an interface to a Web database, additionally, the support for similarity-based functionality in most data management solutions is per se either marginal, intended only for for certain applications like multimedia retrieval, or simply not existent.

Efficiency: similarity-based predicates by default involve a larger search space depending on the “looseness” of the predicate, because not only exact but also approximate matches have to be considered. If the similarity predicate cannot be evaluated locally, the complete search space or a best possible approximation of this space has to be transferred for global query processing.

The approaches presented in this thesis address these problems for string similarity predicates by mapping similarity predicates to standard predicates and minimising the required data transfer by means of an efficient pre-selection based on statistics on substring selectivity.

2.2.3 Autonomy

The third important characteristic of data integration, and in most classifications considered as one of the three dimensions along with distribution and heterogeneity, is the autonomy of data sources. The autonomy is concerned with the control aspect and reflects the independence between source systems as well as between these systems and an integration layer regarding several aspects concerning the design, implementation, and operation of the systems. Veijalainen and Popescu-Zeletin in [VPZ88] considered the following kinds of autonomy.

Design autonomy reflects the independence during the design process of a system and implies independence of

- the data management system and the data model,
- the schema and according constraints,
- provided operations, and
- the universe of discourse and semantics of the data.

The majority of previously discussed heterogeneities exist due to this design autonomy.

Communication autonomy refers to the ability of a source system to decide about communication with other systems, i.e. if, when, and how it responds to requests. This can also be interpreted as the ability of source systems to leave or join an integrated systems at any time.

Execution autonomy is given, if integrated systems are able to independently decide about the way, time, and order of execution of operations. Execution autonomy is for instance problematic for a global consistency management in integrated scenarios.

In [AB89] Alonso and Barbara discussed an additional aspect of autonomy, that is of great importance for the work presented here, and was added to the three previously mentioned by Sheth and Larson in the most often referred classification in [SL90].

Association autonomy represents the ability of a system to independently decide about the degree of sharing its functionality with other systems. This comprises the data managed by the system, where only part of the schema or a subset the actual contents may be available to other systems, as well as the operations to work on it.

As such, the autonomy in general can be considered the cause for all the challenges that make data integration such a complex task. Design autonomy leads to system and schematic heterogeneities. Communication, execution, and association autonomy make the processing of global operations a very complex task as described before. Execution autonomy is critical, when a global transactional context is required and especially if update operations must be supported. Such access characteristics are beyond the scope of this work, and we refer to Chrysanthis and Ramamritham who give overviews of the related problems in [MRJ99] and [RC96].

Furthermore, autonomy is a characteristic and requirement of integrated sources, that must not be violated if no further agreement on control sharing exists. If such an agreement exist, we talk about co-operative sources, where the co-operation may be based on a common interest to take part in an integrated solution, for instance to make the local data available to a broader audience. Such a situation obviously exists for Data warehouses, where the provision of global data for analytical purposes is in the interest of the enterprise and its economic objectives. Co-operative approaches ease some of the problems in data integration, because for instance heterogeneities are often more easily resolvable on the source side. Agreements on a weaker autonomy of component systems may include

- the provision of suitable interfaces and protocols for existing data access operations,
- local support for only globally required operations,
- interfaces and protocols for coordinating distributed operations,
- active communication mechanisms from a source to an integrated system, and
- the exchange of metadata and statistical information about the managed data.

For the work presented in this thesis, and here especially Chapter 6, the main interest is in provided interfaces and operations as well as statistical information about data. The former refers to query capabilities as discussed before in the context of distributed query processing. Statistical information are used in the proposed approach for estimating the selectivity of queries and can be created from source data sets. If sources are not co-operative this information has to be gathered from query samples, which slightly decreases the accuracy and efficiency.

2.3 Data Integration Approaches

Based on the characteristics outlined in the previous section this section will give a short overview of relevant approaches and the aspects that apply for these systems. The approaches draw mainly from research on database integration, but also include research from a more general perspective of information system integration.

2.3.1 Virtual Data Integration

The earliest work on data integration focused on the virtual integration of data, because of the soundness of the approach avoiding redundancy while providing access to an up-to-date view of the data from the source systems. Inspired by the conceptual clarity which lead to the success of database management systems – and here especially the relational systems – research first dealt with conceptual questions regarding schema integration, the integration process, and query languages suitable to address the specific requirements in distributed heterogeneous query processing. This research was mostly done related to the concept of federated databases and multi-databases. While often argued to be a concurrent approach, mediators introduced by Wiederhold in [Wie92] rather present an architectural point of view on data integration, which is applicable to a number of

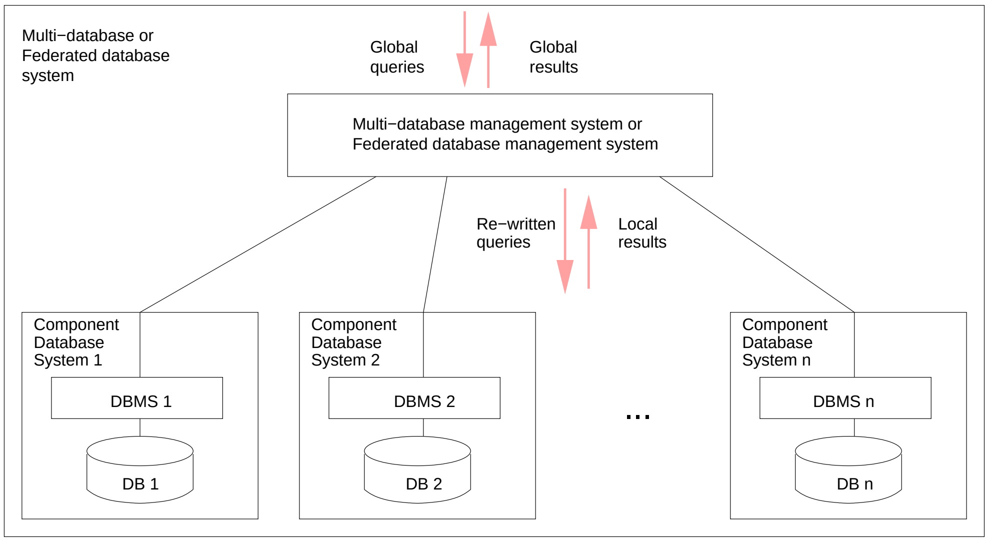


Figure 2.1: General architecture of a multi-database system

approaches and proved to be very successful. So instead of providing a classification, the following two sections present different points of view on data integration.

Multi-databases and Federated Databases

Earliest research on data integration was done in the field of multi-database and federated database systems. Both approaches provide an integration layer on top of existing database management systems differing only in the degree of autonomy of the integrated systems. The overall goal is to provide the users of the multi-database system a uniform, transparent, and full access to the integrated data sets from various source databases, using standard database interfaces and languages.

According to Sheth and Larson in [SL90], *multi-database systems* (MDBMS) integrate a number of component database management systems, which may be of the same (homogeneous MDBMS) or different kind (heterogeneous MDBMS). This, according to Sheth and Larson, explicitly includes components systems which may be distributed or multi-database systems themselves. A very abstract depiction of the according architecture is given in Figure 2.3.1.

A multi-database system is a *federated database system*, if the component system are highly autonomous according to the description in the previous section, i.e. they serve local applications and users and independently decide about processing of operations, their system communication, etc. In contrast to data sharing in federated database systems, the component systems in a *non-federated*

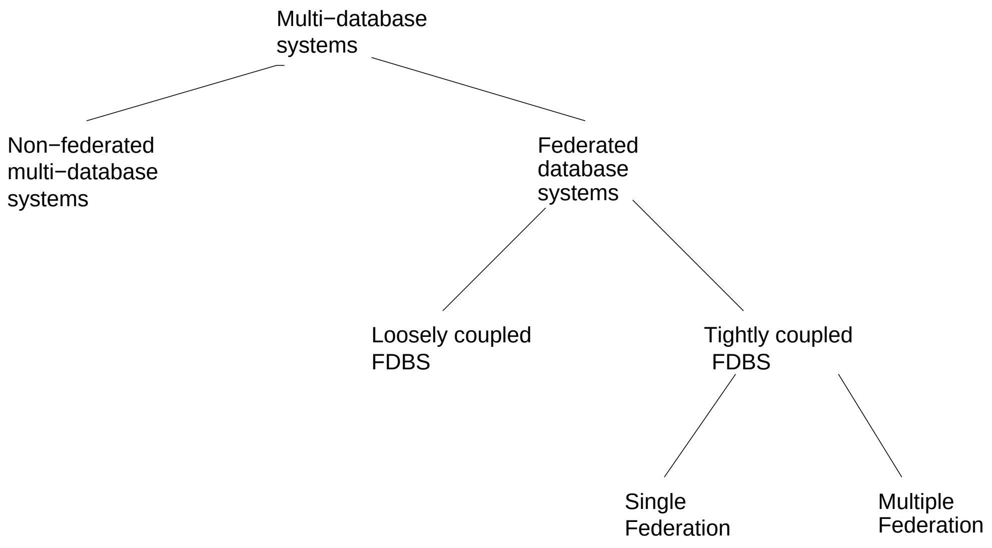


Figure 2.2: Classification of MDBMS according to Sheth and Larson

MDBMS also share the control for their operations with the integration layer. A further classification of federated database systems is given by the responsibility for creating and maintaining the component systems, i.e. an FDBMS is *tightly coupled* if administrators control the access to the component systems or *loosely coupled* if this is the responsibility of the users of the federation. Depending on whether the federation is capable of providing only one or several integrated schemas, a further classification of tightly coupled systems to *single* and *multiple federations* is considered.

This classification by Sheth and Larson is based solely on the autonomy aspect and shown in Figure 2.3.1. There are other classifications based on other aspects or including more recent approaches, but for the remainder of this discussion we will stay with this classification.

From a conceptual point of view, the next addressed issue was providing means to address schematic heterogeneities in the component schemas. Also in [SL90] Sheth and Larson introduced the well accepted 5-level schema architecture for data integration as the counterpart to the 3-level schema architecture in centralised DBMS. In addition to the latter, the three lower levels of the former address the extraction and transformation aspects of schemas of the component systems. Furthermore, Sheth and Larson gave abstract descriptions of architectural aspects regarding query and result transformation, thus providing the framework for the design of these mappings during schema integration.

Federated and multi-database systems inspired research in different directions which were shortly discussed before or are not discussed here in detail, but include for instance

- schema integration,
- multi-database query languages,
- query processing and optimisation in multi-database systems,
- transaction processing across component systems,
- the evolution of integrated systems, and
- the integration of non-database systems.

Of these directions, this thesis will deal will aspects of query processing and optimisation regarding necessary extensions to support similarity-based operations. Furthermore, the proposed operations will be described as extensions to SQL, to provide a possible integration with a multi-database query language.

Mediators

Mediators as introduced by Wiederhold in [Wie92] and later on refined in [Wie94] and [WG97] are often seen as a concurrent approach to federated databases, because the concept of a mediator in deed differs regarding some aspects and was subsequently assigned some differing properties to distinguish it from other approaches. Yet, the concept is rather complementary because it addresses issues of data integration from the broader perspective of information system integration, this way adding some relevant aspects discussed below. Furthermore, it is developed around a very abstract architectural approach of mediators and source wrappers, the latter covering the lower levels of heterogeneities and this way realizing a separation of concerns.

According to [Wie92] Wiederhold described the general intention behind mediator architectures as follows:

> In order to provide intelligent and active mediation, we envisage a class of software modules which mediate between the workstation applications and the databases. These mediators will form a distinct, middle layer, making the user applications independent of the data resources.

These mediator components were complemented by *wrappers* for source systems, covering interface and language heterogeneities. Furthermore, more complex, hierarchical mediator architectures were envisioned, to support different levels of integration. The principal architecture is shown in Figure 2.3.1.

Though it was this architecture that proved most influential and later on was adopted to several integration approaches including federated databases, where

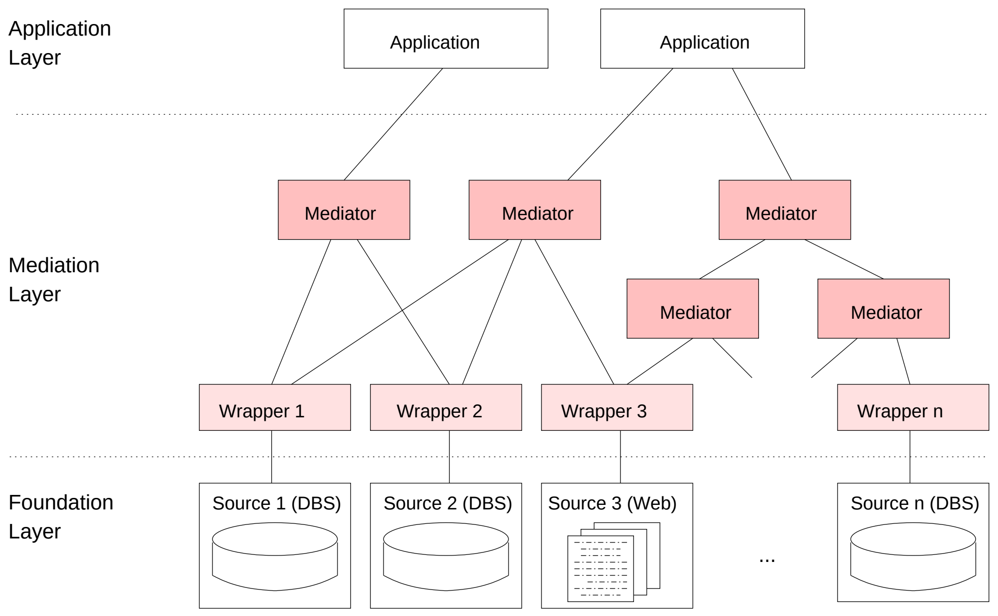


Figure 2.3: Principal mediator architecture

the integration layer now consisted of components providing functionality conforming to mediators and wrappers, the research on mediators raised a number of other interesting questions

Semantic integration: the importance of knowledge about integrated data sets and the way they are created and used was always pointed out by Wiederhold and subsequent research. In deed, managing this knowledge as part of an integration system is a key advantage in dealing with the complexity of data integration and later on branched numerous research activities.

Query capabilities: because mediators are seen in the broader context of information system integration, the aspects of association autonomy as well as interface and language heterogeneities had to be dealt with more explicitly.

Data quality: from an information system point of view the quality of data in integrated data sets is very problematic and therefore plays an important role. Research on mediators right from the beginning included related aspects of semantic heterogeneities like data quality, uncertainty, and differing levels of abstraction in integrated data sets.

Like multi-database systems, mediator-based systems require a uniform schema resulting from schema integration, and based on that, distributed query processing can take place. As previously mentioned, there are a number of properties assigned to mediators, that were not necessarily intended in the original concept, such as

- mediators only provide read access to integrated data sets,
- schema integration must take place in a top-down fashion,
- mediator based integration is best suitable for the integration of non-database systems,
- mediators and especially wrappers are “hard-coded” and not application-independent,
- etc.

In deed, this were characteristics of some successful applications of the mediator concept, but rather than continuing the discussion about general properties we will focus on the architectural and further aspects mentioned before when referring to mediator-based systems.

For the work presented in this thesis aspects of query capabilities and data quality addressed in the context of mediators play the most important role. Wrappers are intended to provide a description of the query capabilities to the mediator, which then can consider these descriptions during query re-writing. This approach is applied in Chapter 6 for mapping similarity-based predicates on string attributes to standard functionality provided by most databases and information systems. The main intention of using similarity-based operations lies in the improvement of data quality by automatically finding related or matching data across various sources.

2.3.2 Materialised Data Integration

Despite the soundness of virtual data integration, in the 1990s an alternative concept of data integration became most popular in practice. Especially in local scenarios like enterprises and other organisations, some aspects of autonomy did not play such an important role anymore, and the complete materialisation of integrated data sets were established as a pragmatic solution to integration problems. This way, not only the complexities of distributed query processing in heterogeneous environments was avoided, but the materialisation also granted the necessary efficient query processing on huge integrated data sets required in many integrative applications.

On the other hand, physical data integration shares many other aspects with virtual integration approaches, such as

- the resolution of system heterogeneities during the import of data from integrated component systems,
- the resolution of schematic heterogeneities through schema integration and the application of resulting mappings during a transformation step,
- the resolution of semantic heterogeneities during a data merging or data cleaning step, and
- concepts to deal with the communication and association autonomy of sources during the import of data.

Furthermore, physical data integration is only applicable where the resulting redundancy and the limited up-to-dateness is acceptable in a given application scenario. Though the general approach of physical data integration can be and was applied in numerous applications, the field drawing most attention was the research on and practical experience made with Data warehouses as introduced by Inmon and described for instance in [Inm96], which is later on described in more detail. Another interesting application exists in the area of integrating semi-structured data from Web sources by extracting their contents as for instance described by May et al. in [MHLL99].

Data Warehouses

The general idea behind *Data warehouses* is to provide access aimed at analytical operations for management decision support on an integrated data set covering data from multiple operative systems within enterprises or organisations. Because these integrated data sets can be huge and analytical operations often have to work on large subsets of the integrated data, a virtual integration does not seem appropriate due to reasons of efficiency.

For the import in a first step data is extracted from each source system to a so-called staging area. The extract either comprises the full set of data or the delta since the last extraction took place. Database systems nowadays provide mechanisms for the efficient export, import, or replication of data, which can be applied during this step. The staging area is a database, where all necessary transformations take place, including

- structural integration according to a schema resulting from schema integration according to virtual integration,
- data cleaning to grant data quality of the integrated data set by removing or repairing implausible or obviously erroneous data, and

- data matching, merging, and grouping to reconcile overlapping or related data sets or achieve a common level of abstraction.

The latter aspects of data matching, merging, and grouping are most often considered part of data cleaning, but are described here separately because they are of special interest for this thesis as discussed later on.

Schema integration for Data warehouses usually is driven by requirements of the analytical applications and, hence, is carried out in a top-down fashion. In a next step, the transformed, cleaned, and merged data set is brought into a separate database forming the basis for the analytical data processing such as Online analytical processing or data mining techniques.

Techniques proposed in this thesis, and here especially Chapter 6, can be applied for the previously mentioned tasks of data matching, merging, and grouping.

Data matching is used to find objects representing the same or related real-world objects. Most often there are no common identifiers usable for this task and imprecise or erroneous data values make an equality based matching impossible. For same objects data merging can take place, and for related objects references or foreign keys can be adjusted.

Data merging is necessary, if objects from different sources representing one real-world object are identified during the matching phase, a common, reconciled representation has to be derived based on the possibly conflicting data values.

Data grouping or householding is used to condense data from source systems or relate it to some more abstract concept in the integrated schema to reach a common level of granularity in the integrated data set.

These problems, also referred to as record matching, entity identification, the merge/purge problem etc., were addressed in research related to data cleaning and Data warehouses as for instance by Calvanese et al. in [CdGL$^+$99], Galhardas et al. [GFSS00], Monge and Elkan [ME96], Hernandez and Stolfo in [HS95], etc. Furthermore, commercial tools provide specialised solutions for very specific tasks like for instance customer address matching.

2.4 Conclusions

This chapter gave a short overview of problems and approaches in data integration, as far as they are related to the research presented in this thesis. This includes the origins of problems resulting from characteristics like heterogeneity, distribution, and autonomy on the most general level of abstraction. In this context, aspects of typical techniques to address these problems like architectural considerations, schema integration, and distributed query processing in heterogeneous environments were described. Furthermore, an overview of existing data integration approaches was given based on the general distinction between virtual and materialised integration.

As this thesis deals with similarity-based operations in data integration related to various problems and applicable in a number of approaches, the research results presented in the latter chapters were positioned accordingly. Here, the main focus was on aspects of distributed query processing in heterogeneous environments and specific aspects resulting from either virtual or materialised integration approaches.

The current situation regarding data integration is characterised by research results covering most critical aspects of the problem on all different levels. Often there are a number of concurrent solutions, as for instance regarding the well studied problem of schema integration. Unfortunately, the complexity of the overall problem spawned many solutions addressing partial problems, which are not necessarily orthogonal and cannot easily be combined. Successful systems such as research prototypes or the very few commercial solutions therefore are often limited in their functionality and tailor-made for certain applications or suitable only for constricted classes of applications.

Accordingly, there are no standards for data integration, so, the work presented in this thesis cannot be described based on such standards and rather relates to common knowledge or certain approaches in the field. Furthermore, like previous research results the techniques proposed here provide solutions for another partial problem, leaving other aspects aside.

Summarised, the area of data integration remains an active research field, where the current focus is on filling the gaps regarding certain open problems, like in this thesis, or applying new technologies which are better suitable to provide stable solutions. For the future, the many parts of the puzzle resulting from research must be put together to form a more coherent picture of the overall task of data integration driven by real-world applications.

Chapter 3

Concepts of Similarity

To provide data processing operations based on similarity, one first has to gain a certain understanding of the characteristics of similarity. While there is a general acceptance of the importance of similarity in various sciences, there also is an obvious lack of common foundations and definitions of the term. Wherever similarity as a concept is used successfully, it is seen from a very specialised point of view. Its formalisations and properties used in one context often are debatable or not useful in another. This chapter discusses different views on similarity, explicitly not intending to provide a new generalised view, but instead discussing implications of certain aspects and adjusting the focus for the usage of similarity in data integration.

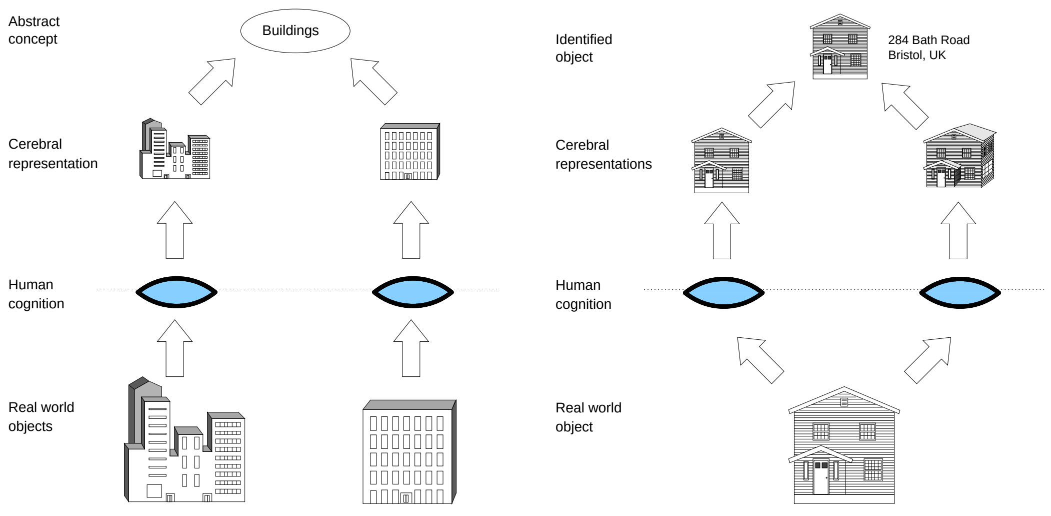
3.1 Introduction

The importance of similarity in our daily life is often underestimated, but it is clearly pointed out in the field of cognitive sciences, comprising psychological and philosophical aspects. Not only that a main inspiration for similarity in computer science is the research done in the field of psychology, but there are also parallels of the way information has to be processed based on similarity by computers and humans. To achieve the capabilities humans have in processing information from the real world and to bridge communication gaps between men and computer similarity will have to play a key role.

The most important application of similarity is taking place in the human brain every millisecond when incoming sensual information is processed. In 1890 William James stated the following ([Jam90]):

> This sense of sameness is the very keel and backbone of our thinking.

As Robert Goldstone pointed out in [Gol99] intellectual and cognitive processes have to be based on similarity, because we only can store and perceive varying



(a) Abstraction (b) Identification

Figure 3.1: Usage of similarity for identification and abstraction of real world objects

or incomplete representations of aspects of the world. Of course humans are able to recognise a person they have met before, but for every new meeting this other person and the new perception of her or him has changed more or less. So the human brain has to be able to map the perceived to the stored representation, or as Sir W. Hamilton put it in [Ham]:

> Identity is a relation between our cognitions of a thing, not between things themselves.

Besides the identification, where two representations refer to the same object in the real world, similarity is also applied in other intellectual processes like association, classification, generalisation, etc., where representations refer to an abstract relationship or concept based on context-specific commonalities. These two aspects of similarity are illustrated in Figure 3.1. In the following sections we will see that similarity is used in computer science in corresponding ways.

Before we have a closer look at certain characteristics of similarity and similarity models, we have to consider the common human comprehension of the word *similar*. Deriving from the Latin word *similis* meaning *like* or *resembling*, the word *similar* is most often used intuitively to compare or relate objects regarding certain common aspects. Yet, these aspects are often left unspecified or are given based on a very loose terms. Hence, in dictionaries one will find descriptions of *similar* and *similarity* like the following from the Oxford English

Dictionary ([Tho95]):

> similar:
>
> 1. of the same kind in appearance, character, or quantity, without being identical.
> 2. ...

This very loose description of the usage of the term already raises two interesting points. At first, similarity between things, persons, concepts, etc. is based on equality of certain aspects or their abstraction. Secondly, the proposition that identical objects cannot be similar, challenges many of the theories introduced later on. Actually, one can find a contradiction within the very same source, when looking up *identity*:

> identity:
>
> ...
>
> 3. a close similarity or affinity.

The relation between similarity and identity – whether it is independence, complementation, or implication in any direction – is discussed later on in more detail, and one will see that all propositions may make sense under more specific conditions.

Another description of the term *similar* is given in The American Heritage Dictionary of the English Language ([Mor96])

> similar:
>
> 1. related in appearance or nature; alike though not identical.
> 2. ...

Here, something is related to something else by being similar, hence, similarity is explicitly considered as a – probably binary – relation. Two non-identical objects are similar if some unspecified condition on common aspects holds. While this certainly reflects a common usage of the term, another understanding that is widely used in research on similarity issues considers similarity as a *measure of likeness* between objects. For example, a simple similarity measure is the number of features two objects have in common – the greater this value is, the more similar the two objects are. In this case not only the fact that objects are similar is of interest, but also the quantifiable degree of similarity. Both points of view are useful, and similarity measures are actually a common way to specify the above mentioned conditions of a similarity relation.

One major problem of similarity is that it heavily depends on the context of usage. From a quite pessimistic point of view the philosopher and linguist John R. Searle wrote in [Sea79]:

> Similarity is a vacuous predicate: and any two things are similar in some respect or another. Saying that the metaphorical "S is P" implies the literal "S is like P" does not solve our problem. It only pushes it back a step. The problem of understanding literal similes with the respect of similarity left unspecified is only a part of the problem of understanding metaphor. How are we supposed to know, for example, that the utterance "Juliet is the sun" does not mean "Juliet is for the most part gaseous", or "Juliet is 90 million miles from the Earth", both of which properties are salient and well-known features of the Sun.

Given Shakespeare's play "Romeo and Juliet" plus some knowledge of human social interaction as the "respect of similarity" the cited metaphor becomes understandable. Contrary to equality relationships for similarity relationships one has to be more specific about the conditions under which the relationship holds. Furthermore, these conditions can be specific not only to certain classes of compared objects but also to single instances, which makes the usage of similarity even more difficult. This problem is outlined from a psychological point of view by Medin et al. in [MGG93] and from a philosophical perspective by Goodman in [Goo72].

Unfortunately, there is no such thing as a general theory of similarity in mathematics. The most common usage is related to geometrical shapes, where a binary relationship between two shapes exists if a limited set of transformations, e.g. dilation, rotation, expansion, reflection, etc. depending of the kind of similarity, can be applied to transform one object to the other. Sometimes these transformations are referred to as *similarities* [Wei99]. Another occurrence of the term is related to *self-similarity* and fractals [Wei99, Hut81]. But, just like the former this is a rather specialised application of similarity instead of a general view on the concept. Nevertheless, mathematics provide the foundations of popular similarity measures like the distance in metric spaces described in the following section.

For a first summary, while it is hard to overestimate the importance of similarity, there are major problems with the very foundations of similarity as a concept. Its relationship to other important concepts like identity varies depending on the usage. The term is ambiguously used for relationships as well as for measures of similarity. Similarity depends heavily on the context of its usage, and very often it is not easy to specify this context. These problems and more specific ones described in the following sections constrained a wider usage of this generally very useful concept in computer science. Therefore, when introducing various models of similarity in the following sections we have to be very careful with the term itself and related terms.

3.2 Models of Similarity

Just as the common understanding of the term *similarity* varies, there is a number of contradicting formalisations of similarity and its characteristics. Therefore, throughout this paper we will use rather loose definitions of terms separated from certain characteristics that may or may not apply in certain scenarios.

For this purpose, basic concepts and terms are discussed and formalised. Metrics as a general concept for measuring similarity and their shortcomings regarding some aspects of similarity are described. Related to this discussion, various specific measures of similarity are described. A rough classification of models for measuring similarity was given by Goldstone in [Gol99] as

- geometrical models,
- featural models,
- alignment-based models, and
- transformational models.

The focus in this section will be on geometrical, featural, and especially on transformational models. Geometrical models are the currently most often used approach in computer science, while featural models as used in psychology are closer to the human understanding of similarity. Transformational models are applied for instance in approximate string matching which is most relevant for the research presented in this thesis. Therefore, string similarity is discussed in more detail in the next section.

3.2.1 Similarity Measures and Predicates

At first, we have to draw a clear distinction between similarity measures and a similarity relation. Given two objects a and b from not yet further defined domains A and B, respectively, we define a similarity measure as follows:

Definition 3.1 *A similarity measure is a non negative function* $sim : A \times B \rightarrow \mathbb{R}$ *expressing the similarity between two objects* $a \in A$ *and* $b \in B$*. If* $A = B$ *and the similarity measure is* $sim : A \times A \rightarrow \mathbb{R}^+$ *we call it a similarity measure on A.*

This definition explicitly includes the *dissimilarity* or *distance* between two objects, because an interpretation of the result of the function is not yet specified. Typical interpretations of similarity measures are:

A normalised similarity measure: $sim : A \times B \to [0,1]$ where $sim(a,b) = 0$ if the objects are least similar and $sim(a,b) = 1$ if the objects are most similar or identical, depending on the context of usage.

A distance or dissimilarity measure: $dist : A \times B \to [0,\infty]$ or $dist : A \times B \to [0, maxDistance]$ where $dist(a,b) = 0$ if the objects are most similar, and $dist(a,b) = maxDistance$ if the objects are most dissimilar.

We will see that the concept of a distance measure is used very often – for instance in metric dimensional scaling – because the distance of two objects can be measured more easily than their commonalities. Nevertheless, transformations between different interpretations are possible, like for instance for the two previously mentioned $sim(a,b) = 1 - \frac{dist(a,b)}{maxDistance}$ or $sim(a,b) = \frac{1}{1+dist(a,b)}$ if a maximum distance is not known.

An often used normalised similarity measure from the area of Information retrieval is the cosine similarity defined for two vectors $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ as

$$sim(x,y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Because document vectors in Information retrieval are representations of the frequency of terms in the document compared to the frequency in the overall collection, the vectors are considered to be in the positive sector. Accordingly, the function returns 1 whenever the angle between two vector representations is 0 and returns 0 if the vectors are orthogonal, i.e. the vector representations of texts share no common terms ([BYRN99]).

Apart from the interpretation of the result there is much more left unspecified in this definition. Usually definitions include a set of characteristics of the function like symmetry or the triangular inequality that are only applicable for certain measures described later on. Furthermore, we did not yet specify the context of similarity and how the similarity is measured.

To provide similarity based operations we will require similarity relations between objects and according similarity predicates.

Definition 3.2 *A similarity relation is a binary relation $SIM \subseteq A \times B$, where $(a,b) \in SIM$ if two objects a and b are similar.*

Definition 3.3 *A similarity predicate $SIM(a,b)$ or $a \sim b$ is a two-valued logic operator deduced by a similarity relation, i.e. $a \sim b \Leftrightarrow (a,b) \in SIM$.*

Instead of measuring the similarity or dissimilarity of objects, given a similarity relation and the deduced predicate we can explicitly say that two objects actually are similar, again, not yet considering he context and the criteria the relation is based on. Nevertheless, for future usage we will mostly rely on similarity predicates which are defined based on similarity measures. A common way to specify similarity predicates this way is to introduce a threshold for the similarity measure. For the previously mentioned interpretations of similarity measures, the according predicates are:

Normalised similarity predicate: $a \sim b \Leftrightarrow sim(a,b) > t$,
where $sim(a,b)$ is a normalised similarity measure and $t \in \mathbb{R}^+$ is a similarity threshold $0 \leq t \leq 1$ meaning the closer t is to 1 the more similar the objects have to be and the more rigid is our similarity predicate.

Distance or dissimilarity predicate: $a \sim b \Leftrightarrow dist(a,b) \leq k$,
where $dist(a,b)$ is a distance or dissimilarity measure and $k \in \mathbb{R}^+$ is a distance threshold $0 \leq k \leq \infty$ expressing the required closeness of objects, i.e. the closer k is to 0 the more rigid is our similarity predicate.

Normalised similarity measures can further directly be used to specify predicates in a fuzzy logic, where the result of the logic operator returns a degree of truth between 0 and 1.

Fuzzy similarity predicate: $a \sim b := sim(a,b)$,
where the result is the degree of similarity between the objects.

Such an interpretation can for instance be used for duplicate detection based on similarity between different representations, where the result of the predicate is interpreted as the probability of sameness. Furthermore, such an approach is useful when specifying logic expressions to describe the similarity between more complex objects based on the similarity of components such as attributes as well as contained or related objects. In this case, several fuzzy similarity predicates can be combined using standard logical operators. Using the previously introduced two-valued predicates would require specifying a threshold for each atomic predicate.

3.2.2 Metrics as Similarity Measures

The most common usage of similarity measures refers to distances in metric space defined as follows based on [Wei99].

Definition 3.4 *A metric space is a set S with a global distance function (the metric g) which for every two points $a, b \in S$, gives the distance between them as a non-negative real number $g(a,b) \in \mathbb{R}^+$. A metric space must also satisfy*

1. $\forall a,b \in S : g(a,b) = 0 \Leftrightarrow a = b$ *(Constancy of Self-similarity)*
2. $\forall a,b \in S : g(a,b) = g(b,a)$ *(Symmetry)*
3. $\forall a,b,c \in S : g(a,b) + g(b,c) \geq g(a,c)$ *(Triangular Inequality)*

Accordingly we define a similarity metric, which is a special case of the dissimilarity or distance measure introduced before:

Definition 3.5 *A similarity metric is a similarity measure that satisfies all axioms for a metric.*

Other related terms for a metric are:

Positivity: $\forall a,b \in S : g(a,b) \geq 0$,
which conforms to the definition of a metric as a non-negative function.

Minimality: $\forall a,b \in S : g(a,b) \geq g(a,a) \Leftrightarrow a \neq b$,
which results from the constancy of self-similarity and positivity.

The typical example for a metric space is the n-dimensional *Euclidean space* $\mathbb{R}^n$, consisting of all points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, and the *Euclidean metric* or *distance*. As the points in the Euclidean space are represented by n-dimensional vectors, the Euclidean space is also referred to as *vector space* or *n-dimensional space*. A generalised form of metrics for the Euclidean Space is the *Minkowski distance*. The *Manhattan* or *City-block distance* like the Euclidean Distance is a specialisation of the Minkowski distance. There are other distance measures for Euclidean Spaces, some of them satisfying the condition for metrics, e.g. the *Chebyshev distance*. The previously mentioned metrics for any two points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are defined as the following functions.

Minkowski distance: $dist_p(x,y) = \sum_{i=1}^{n} \sqrt[p]{|x_i - y_i|^p}$,
describes a general class of distance measures of various orders $p \in \mathbb{N}^+$, also called L_p distance.

Euclidean distance: $dist_2(x,y) = \sum_{i=1}^{n} \sqrt{|x_i - y_i|^2}$,
Minkowski distance with $p = 2$, or L_2 distance.

Manhattan distance: $dist_1(x,y) = \sum_{i=1}^{n} |x_i - y_i|$,
Minkowski distance with $p = 1$, or L_1 distance.

Chebyshev distance: $dist_\infty(x,y) = max_{i=1}^{n}|x_i - y_i|$,
maximum distance in any dimension, and the upper bound for Minkowski distances of growing order p.

Because metrics in vector spaces are an useful and easy to understand measure for similarity, or, more precisely, dissimilarity between objects, early research on similarity in psychology and computer science was based on vector spaces. As the objects considered were merely not points in such a space, a key aspect of measuring similarity was mapping objects to vector representations in any given space. Examples for such mappings are multi dimensional scaling, described later on and used for instance in early psychological research, as well as the extraction of feature vectors from multi-media data in computer science.

But metric spaces do not only include spaces of a fixed dimensionality. Every set with a distance measure satisfying the metric axioms is a metric space, e.g. the *edit* or *Levenshtein distance* for strings first described in [Lev66]. It is defined as the minimal number of insertions, deletions, or substitutions of single characters necessary to transform one string to another. If $x_i = x[1]x[2]\dots x[i]$ and $y_j = y[1]y[2]\dots y[j]$ are strings with all characters $x[k] \in \Sigma, 1 \leq k \leq i$ and $y[l] \in \Sigma, 1 \leq l \leq j$ over one alphabet Σ, the edit distance of the two strings can be computed as follows:

$$edist(x_i,y_j) = \begin{cases} 0 & \text{if } i = j = 0 \\ \infty & \text{if } i < 0 \vee j < 0 \\ edist(x_{i-1},y_{j-1}) & \text{if } x[i] = y[i] \\ min\begin{pmatrix} edist(x_i,y_{j-1})+1 \\ edist(x_{i-1},y_j)+1 \\ edist(x_{i-1},y_{j-1})+1 \end{pmatrix} & \text{else} \end{cases}$$

In the original paper Levenshtein shows that this distance measure satisfies the metric axioms and, therefore, is a metric on the set of all strings over a given alphabet. While similarity based on distance in Euclidean spaces is often referred to as *Geometrical similarity*, the Levenshtein distance and and similar edit distance approaches for graphs and composite objects are referred to as *Transformational similarity* according to a classification given by Goldstone in [Gol99].

The first use of geometrical models for analysing similarity was in psychology. Perceptual stimuli were considered measurable in several dimensions and the distance between measured values was used as a measure of similarity of the stimuli. General problems of measurement were for instance discussed by Stevens in [Ste46]. In 1950 Attneave stated in [Att50]:

The question ’What makes things seem alike or seem different?’ is

> one so fundamental to psychology that very few psychologists have been naive enough to ask it.

Furthermore, in the same publication he argued for the Manhattan distance between measurable stimuli, while Torgerson argued for the Euclidean distance in [Tor52]. In this latter publication Torgerson first applied *Multidimensional scaling* (MDS), an overview of which is given in [LH82], to a special case of similarity analysis. Instead of actually measuring the stimuli very often only information on the distance between objects is available. To derive a representation in a Euclidean space MDS can be applied on a computable distance metric, similarity matrices, or confusion matrices, the latter two resulting for instance from experiments. The mapping is done by continuously adjusting coordinates and this way minimising the *stress* of the resulting space, where the stress is a measure for the differences between real distances and distances in the mapped space. The result of MDS is a representation of all the objects from the input set as points in a space of dimension n, where n is an input parameter. If n is equal to the number of objects, an optimal solution without stress is always possible.

This work on scaling and metric spaces by Torgerson (see also [Tor58, Tor65]) and later on by Shepard ([She62a, She62b]) was most influential on following geometric views on similarity, including the discussion on the usability of various metrics under various circumstances, e.g. the correlation between dimensions. A comprehensive overview of geometric models of similarity used in psychology, as well as others described later on, is given by Navarro in [Nav02].

In computer science the research on similarity is mostly based on Euclidean spaces. Such spaces of a fixed dimensionality n can be indexed efficiently using well-known techniques like *R-trees* and derivatives ([Gut84, SRF87], *Grid Files* ([NHS84]), *z-ordering* ([Ore90], etc., due to the neighbourhood preserving nature of these structures. The usability of these approaches, though, is limited by the number of dimensions n. This effect is known as the *Curse of dimensionality*. Mapping objects to a Euclidean space is for instance addressed for complex data objects such as multimedia data through terms of *Feature extraction* ([Kit86, Jag91], where certain measurable aspects of objects are used to derive the vector representation directly from one object. A concurrent approach is based on the previously mentioned Multidimensional scaling, which itself is computationally expensive and not suitable for large datasets. Therefore, Faloutsos et al. in [FL95] described *FastMap*, also deriving a Euclidean representation of objects for which a distance function or matrix is given, but applying some reasonable simplifications. Jin et al. in [JLM03] use this approach for approximate string matching.

There are several advantages of similarity metrics resulting from the metric axioms, especially when the metrics are used for data processing. Considering the definition of similarity relations described before, the constancy of self-similarity and the symmetry directly translate to a reflexive and symmetric similarity relation. For similarity based operations that would mean, we do not have to check whether an object is similar to itself, and if the similarity of object a to object b also exists from object b to object a. Current data processing is often based on equivalence relations, which are reflexive, symmetric, and transitive. All the optimisations resulting from the former two properties can be applied, if a similarity operation is based on a similarity metric.

Unfortunately, similarity relations are in general not considered transitive. The triangular inequality can be considered a mitigation of transitivity for distance measures by at least preserving the neighbourhood of different objects, which is useful for efficient access to similar objects. For instance, this is used by index structures conserving the notion of closeness that is expressed through the triangular inequality, as mentioned above. Nevertheless, transitivity does not hold for similarity relations. Considering the previously introduced derivation of similarity predicates and relations from a distance measure as

$$(a,b) \in SIM \Leftrightarrow dist(a,b) < k$$

we immediately see that symmetry and reflexivity hold for the relation SIM, i.e.

$$\begin{gathered}(a,a) \in SIM \Leftrightarrow dist(a,a) = 0 \leq k \\ (a,b) \in SIM \Leftrightarrow (b,a) \in SIM \Leftrightarrow dist(a,b) = dist(b,a) \leq k\end{gathered}$$

which always evaluate to $true$ if $dist$ is a metric. Considering three objects a, b and c where the relation holds for a and b as well as for b and c, i.e.

$$\begin{gathered}dist(a,b) \leq k \vee \\ dist(b,c) \leq k \Rightarrow \\ dist(a,b) + dist(b,c) \leq 2k\end{gathered}$$

and now additionally consider the triangular inequality

$$dist(a,c) \leq dist(a,b) + dist(b,c) \leq 2k$$

transitivity obviously does not hold for the derived similarity relation SIM. Hence, in this case SIM is a reflexive, symmetric, atransitive relation.

3.2.3 Problems with Common Models

At this point we are at the most common understanding of the terms related to similarity: similarity is expressed by distance measures which are metrics, i.e. they satisfy the metric axioms of constancy of self-similarity, symmetry, and the triangular inequality. Similarity relations are reflexive, symmetric, and atransitive. Most definitions refer to these properties.

Nevertheless, there are problems with these models and almost all properties were refuted by results of psychological experiments and diverging properties of specific similarity measures and similarity relations. Obviously, the real comprehension of similarity is less rigid than the properties introduced so far. The constancy of self-similarity, symmetry, and the triangular inequality were attacked by psychological research, for instance by Krumhansl in [Kru78] and Tversky in [Tve77]. If these do not hold, we also have to reconsider the symmetry and reflexivity of similarity relations.

Though often neglected, these observations are relevant in data processing. As a motivation, we have to remember that similarity always depends on the context of similarity, which in data processing may be a given application or value domain. For such a context a specific similarity measure can be used, which may be provided by the designer or the user of the given application, for instance as a user-defined function. As shown later on, it is easy to specify a similarity predicate that is neither reflexive nor symmetric, and still may make perfect sense for a certain application.

The model developed by Tversky is a featural model of similarity according to the classification by Goldstone in [Gol99]. Instead of representing stimuli by a number of measurable or derivable coordinates to some Euclidean space, he characterised them by a set of features that they possess. Let $\mathcal{A}$ and $\mathcal{B}$ be the set of features of objects a and b, respectively, then Tversky described the similarity of a and b in his *Feature contrast model* ([Tve77]) as a similarity measure

$$sim(a,b) = f(\mathcal{A} \cap \mathcal{B}) - \alpha f(\mathcal{A} - \mathcal{B}) - \beta f(\mathcal{B} - \mathcal{A})$$

where $\alpha, \beta \geq 0$ and f is a non-negative function. Tversky showed that this model is closer to the human cognition of similarity. Results of psychological experiments had shown tendencies towards asymmetric similarities, which could not be represented by geometrical models based on metrics. A typical experimental result was that variants were considered more similar to a prototype than vice versa. Santini and Jain applied and refined Tversky's feature contrast model for similarity based operations on image data in [SJ97].

In [Kru78] Krumhansl argued that the constancy of self-similarity does not hold if a distance function also considers the density of stimuli, which was shown to have an impact on the dissimilarity by experimental results. The triangular inequality was refuted by Ashby and Perrin in [AP88] as well as Tversky and Gati in [TG82]. In the latter article Tversky and Gati introduced weaker properties of distance functions in a fixed dimensional feature space as an alternative to the metric distance axioms. These so called *Monotone proximity structures* have the following properties satisfied by most distance measures:

Dominance: the distance between two objects is greater than the distance between their projections on the coordinate axes. This is a weaker form of the triangular inequality.

Consistency: the ordinal relation of distances between objects in one dimension is independent of the values in the other dimensions.

Transitivity: if the projection of an object b on one dimension is between the projections of objects a and c, written as $a|b|c$, and furthermore c is between b and d, that means $b|c|d$, then $a|b|d$ and $a|c|d$ also hold.

This framework is less rigid than the metric axioms and better explains the human cognition of similarity, while still preserving the intuitive notion of closeness through the ordinal relations of dissimilarities along dimensions. Unfortunately, monotone proximity structures and their implications are not well researched, especially in computer science.

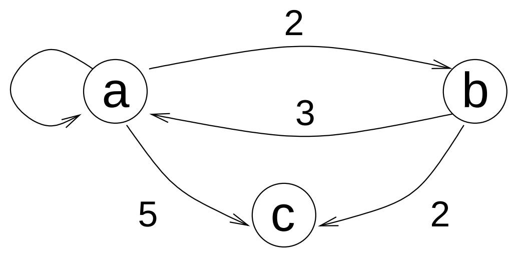


Figure 3.2: Distances in a weighted directed graph

Again, it is not the intention of this work to provide a new framework for similarity or show that any of the above mentioned approaches is correct in all given scenarios. Instead, we will discuss what implications the presence or absence of certain properties of similarity measures and similarity relations has on similarity-based operations. Missing metric axioms, for instance, may easily be the case for application-specific similarity measures. As a somewhat malicious example consider the weighted directed graph depicted in Figure 3.2. If we define a distance measure between nodes in this graph as the minimum sum of edge weights of all paths from one node to another or as ∞ when no such path exists, we see that constancy of self-similarity and symmetry do not hold, because $dist(a,a) = 1$ while $dist(b,b) = 5$, and $dist(a,b) = 2$ while $dist(b,a) = 3$. If we change our distance measure to not consider paths, but only direct edges, we also lose the triangular inequality, because $dist(a,c) = 5$ while $dist(a,b) + dist(b,c) = 4$. In this case, monotone proximity structures would not help either, because the model is not geometric. Multidimensional scaling or FastMap cannot yield meaningful results, because the definition of the distance as ∞ for non-existing paths makes the approach useless. Yet, if we consider the edge weights as transformation costs, a query like "find all nodes similar to a, i.e. nodes that can be the result of (a single or multiple) transformations with costs less than 3" is meaningful and would yield the result $\{a,b\}$.

Regarding the properties of similarity predicates, symmetry and reflexivity do not hold if the predicate is specified based on a distance measure, which does not provide constancy of self-similarity and symmetry. Consider objects X and Y representing sets, for instance of features or information retrieval terms. If we use a similarity measure like

$$sim(X,Y) = \frac{|X \cap Y|}{|X|}$$

to find objects similar to X, i.e. those who share a considerable number of elements with X, the similarity measure is asymmetric. This is because we do not take the cardinality of Y into account, where Y may contain many more elements not in X. One might argue, that the often used Jaccard set distance

$$sim(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

might be a better choice. Yet, the above scenario is commonly used in Information retrieval, where X represents a query and Y represents a document, and the symmetry of the predicate is of no concern. Also, in data integration scenarios we may not have full access to the set of queried objects, so we again would have to use the former approach.

Furthermore, we may not at all want our similarity relation to be reflexive, i.e. we do not want identical objects to be considered as similar. Looking for objects which are similar to object X, we may not want the object itself as part of the result. As an example consider record linkage in a Data Warehouse, where it is not necessary to link a record with itself, but rather only with these records, which are actually similar and not identical.

Finally, dealing with the context-specific nature of similarity requires similarity measures suitable for very specific applications. A common way to provide such tailor-made similarity measures is to implement user-defined functions according to the previously given definition of a similarity or distance measure. But because the efficient evaluation of predicates based on such measures to a high degree depends on according index support, which in theory can be given if for instance the triangular inequality holds, the implementation can be very problematic. In this case, a possible implementation of the similarity measure can be based more primitive predicates.

As an example, consider the matching of person names, such as “Albrecht Dürer”, “Albrecht Duerer”, “A. Dürer”, and “Dürer, Albrecht”. Though all names probably refer to the same German renaissance artist, using for instance the string similarity will only help with the former two representations. But because we know about the semantics of an attribute `ARTIST_NAME`, we may use techniques introduced in the following Section 3.3 to address for instance the problems of the German umlauts or abbreviations of first names. Furthermore, we can apply elementising to get tokens representing first or last names, and then use these for index based similarity lookup based on the standard edit distance to grant efficiency of predicate evaluation. This way techniques introduced in Chapters 5 and 6 can be used.

3.3 String Similarity

The similarity of strings is most important considering the fact that most conventional applications deal with simply structured data either expressed as string or numerical data values. Though the problems of string similarity and approximate string matching have been a topic of research for a very long time, there still is very little support for these concepts in current data management solutions. Likely reasons for this are problems of

- efficiency, because as though there are means for approximately matching strings, efficient support for such operations is either too complex or still in its infancy, and
- the context-specific aspects of approximate string matching, where efficient solutions often have to be based on the semantics of a string attribute considered in a predicate.

For instance, the SQL standard only supports phonetic similarity through the Soundex coding introduced in [OR18] as early as 1918 by Odell and Russel, and matching strings with patterns according to regular expressions. For the latter, commercial database management systems mostly fail to support the efficient processing of such predicates.

Nevertheless, there are numerous applications of approximate string matching, for instance related to data integration, text retrieval, signal processing, computational biology, and many more. Overviews of according research are given for instance by Hall and Dowling in [HD80], Jokinen et al. in [JTU96], and more recently by Navarro in [Nav01], on all of which the overview given here is based.

While the approaches presented in Chapters 5 and 6 are applying distance measures for strings like the Levenshtein distance already introduced in Section 3.2.2 and discussed in more detail later on, there are a number of techniques for transforming string values to canonical forms which may be helpful in certain application scenarios either for reducing the problem of approximate string matching to equivalence matching, or as pre-processing steps for similarity matching. Basically, such transformations are applied before internally processing, storing, or indexing strings. Transformations for canonising include for instance:

Character transformations: this includes for instance commonly used transformations to derive a canonical string representation like lower or upper case letters and the removal of special characters like white spaces or dashes. As an example consider the the transformation of "Multidatabase" and "multi-database" to the internal representation as "multidatabase".

Elementising: the deconstruction of complex strings to atomic elements like words also referred to as tokenising is sometimes required to perform further transformations like stemming, translation, etc. Furthermore, certain similarity measures on strings can be based on their decomposed representation, as for instance described by Luján-Mora and Palomar in [LMP01a].

Stemming: the reduction of words to a common root based on syntactical rules as outlined for instance in [BYRN99] is common practice in the field of Information retrieval. This allows for instance the matching of "respectively" and "respective" based on their common root "respect".

Translation: though the automatic translation of texts still is problematic, the translation of shorter string values expressed in possibly varying languages may make sense to a certain degree. This translation can be based on dictionaries or specialised mapping tables. As an example consider a categorical attribute of publication types in a German Digital library with values like "Buch", "Artikel", and "Konferenzband" which can be mapped to the according English terms "book", "article", and "proceedings", respectively.

Substitution of alternative spellings: multi-lingual and other language issues may lead to varying spellings of words, such as the difference between English and American spelling or the new German orthography introduced in the 1990s. An example would be "materialisation" or "materialization" in either English or American spelling, respectively, which can be resolved based on rules. Also, names of persons, places, etc. may have different transcriptions, like for instance "Al-Kaida" and "Al-Qaida", where mappings or thesauri seem more appropriate.

Substitution of synonyms: the replacement of certain terms with a most common term describing a class of real-world entities based on thesauri can be applied in various scenarios. This allows for instance the matching of "alike" and "akin" based on the more common word "similar". The problem of synonyms is addressed extensively in the field of Information retrieval as outline for instance in [BYRN99].

Abbreviation expansion or reduction: a common problem in string matching is the matching of terms to according abbreviations or between diverging abbreviations. This can be done applying mappings or rules for the expansion from or reduction to abbreviations. The information loss and possibly resulting ambiguity make this a hard problem.

Phonetic transformation: the previously mentioned Soundex coding as well as derived methods transform words to equivalence classes of similarly sounding words. Though this is helpful to deal with erroneous data input, the transformed representation can only be used for equivalence matching and no further processing.

All these techniques in general imply an equivalence relation, though for instance the replacement of synonyms or abbreviations may lead to problems resulting in similarity relations. Though a number of problems can be resolved applying these techniques, the general problem of approximate string matching still persists, resulting in a similarity relation between string representations.

The majority of problems addressed by approximate string matching result from erroneous or imprecise data, though some of the basic problems mentioned above, like for instance alternative spellings, character transformations, word stem matching, or phonetic matching can partially be dealt with based on string similarity. The approximate string matching problem is defined by Navarro in [Nav01] from a text retrieval point of view as follows:

Definition 3.6 *Let Σ be a finite alphabet of size $|\Sigma| = \sigma$. Let $t \in \Sigma^*$ be a text of length $n = |t|$. Let $p \in \Sigma^*$ be a pattern of length $m = |p|$. Let $k \in \mathbb{R}$ be the maximum error allowed. Let $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$ be a distance function. The problem of approximate string matching in texts is: given t, p, k, and d, return the set of positions j such that there exists i such that $d(p, t[i]..t[j]) \leq k$.*

Rather than finding positions within texts, we focus on finding similar strings in sets, which may for instance be the values of a string attribute in a relation. Hence, our definition is slightly modified.

Definition 3.7 *Let $s \in \Sigma^*$ be a search string and $T \subset 2^{\Sigma^*}$ be a set of strings over the same alphabet. The problem of approximate string matching in string sets is: given s, T, k, and d, return the set of all strings $t \in R$ such that $d(s,t) \leq k$.*

Suitable distance measures for string values are transformational measures according to the classification given by Goldstone in [Gol99], i.e. they measure the dissimilarity in terms of operations necessary to transform one string to another. Various distance measures can be distinguished based on

- the kinds of operations allowed, and
- the costs assigned to these operations.

Typical operations are the deletion, insertion, replacement, or transposition of characters. Other considered operations for instance include reversals or the permutation of complete substrings, such as for instance the *Block edit distance* introduced by Tichy in [Tic84]. Similarly, Ukkonen in [Ukk92] described similarity of strings in terms of common substrings of a fixed length called *q*-grams, which are used for a related purpose in Chapter 6. The most common string distance measures are based on the typically considered operations mentioned above.

Levenshtein distance: the Levenshtein or simple edit distance *edist* introduced formally in Section 3.2.2 considers the minimal number of

- insertions – $edist('ac','abc') = 1$,
- deletions – $edist('abc','ac') = 1$, and
- substitutions – $edist('abc','adc') = 1$

to transform one string to another.

Extended edit distance: A commonly used extension of the Levenshtein distance also considers transpositions, i.e.

$$edist('ab','ba') = 1$$

because it covers typical typing errors. This extended version like the simple edit distance is a metric.

Hamming distance: the Hamming distance described for instance by Kruskal and Sankoff in [KS83] only allows substitutions and, therefore, is used mostly for strings representing specific sequences and not natural language.

Episode distance: the Episode distance introduced by Das et al. in [DFGG97], similarly to the Hamming distance was designed for strings representing sequences, e.g. of events in a temporal space. It allows only insertions and, therefore, is not a symmetric measure and no metric.

Longest common subsequence distance: introduced by Needleman and Wunsch in [NW70] the longest common subsequence distance targets specific sequences in computational biology. It allows only insertions and deletions, representing the distance as the number of unpaired characters in both strings. This measure satisfies the metric axioms.

All these distance measures in their common form use fixed costs of 1 for all operations. More specific distance measures may assign costs to operations on various levels, to better reflect the actual difference of both strings. This includes the following costs.

Operational costs which are assigned to deletions, insertions, etc. without further considering the characters or their context they are applied on. This makes sense, because the errors addressed by the operations have varying statistical frequencies, e.g. false characters appear more often in strings than missing or added characters.

Character costs refine operational costs by assigning costs for certain transformations based on the considered characters. This reflects phonetic, linguistic, or mistyping aspects, e.g. it is more likely to replace an ’n’ with an ’m’ because of their phonetic similarity than ’t’ and ’m’.

Context-specific costs as a further extension also consider the context of a character for deriving the costs of an applicable operation. For example, “Saunders” is less different from “Sanders” than “Sanderus”, because the insertion of a vowel next to another vowel implies a greater phonetic similarity than the introduction of a new syllable by inserting it between consonants.

Though very helpful to better express the actual similarity, such costs are often neglected in research. A careful assignment of costs does not impact the overall concepts of the distance measures and can be integrated with existing algorithms. Nevertheless, problems such as asymmetry etc. can be introduced, e.g. if the insertion and deletion costs do not match.

A detailed description of efficient algorithms, for instance based on dynamic programming and bit parallelism, is beyond the scope of this work and we refer the reader to Jokinen in [JTU96] and Navarro in [Nav01]. Furthermore, approximate string matching using indexes is a rather new research field, but mainly addressed from an Information retrieval point of view dealing with longer texts, e.g. Navarro and Baeza-Yates in [NBY99] and Ukkonen in [Ukk93]. A similar approach by Shang and Merret described in [SM96] is applied in Chapter 5 to support string similarity-based operations in materialised data sets.

3.4 Conclusions

This chapter gave an overview of similarity, its general importance, as well as the problems that still hinder its broad application in computer science. For computer systems to come near the human capabilities of cognition and intellectual processes requires a better integration of similarity-based concepts. This is very problematic, mainly due to the currently limited understanding of the nature of similarity and the strong dependence on a context of similarity that may vary widely for every application.

In this chapter quite loose definitions and descriptions of similarity measures, relations, and predicates were given to allow a great number of applications. The properties of these concepts that may or may not hold were discussed and the consequences were to some degree left open for further discussion in this thesis and related work. On the other hand, the importance of fixed properties such as established through the usage of distances in metric space, which are the current state of the art for similarity-based operations in computer science, was pointed out.

Some commonly used similarity measures were introduced to illustrate the mentioned concepts and related aspects. A special focus was on similarity measures for string data values, which are used throughout this thesis to define similarity predicates and discuss aspects of operations based on these predicates. Accordingly the problem of approximate string matching based on string similarity was described.

The research on similarity and its integration in computer science remains a challenging area with many problems currently unresolved. Just as the current view on similarity applied in data management draws heavily from early research done in the field of psychology, open problems more recently addressed in the latter area are only rarely considered in computer science. Furthermore, even on the most general level similarity relations and the often implied probabilistic aspects do not well integrate with current data management solutions, which are based on equivalence relations and the assumption of data representing unquestionable facts. Therefore, further research on a conceptual framework addressing these issues is required.

Chapter 4

Similarity-based Operations

In recent years the concept of similarity has found its way into computer science via various areas, mainly driven by new applications working with complex or imprecise data. All these applications share the need to improve the abilities of data processing towards human cognition and intellectual processes. Therefore, the usage can roughly be distinguished according to the two main areas of the usage of similarity in human cognition described in Section 3.1. Again, we have to consider real world objects and their representations. The latter this time exist in computer memory and are referred to for reasons of simplicity as objects.

Identification: objects may be representations of the same real world object, yet the representations may differ due to different formats, precisions, input conventions, etc. or incomplete and erroneous data. In this case the identification has to be based on similarity. As data is becoming more complex and the availability from various sources increases, similarity-based identification becomes more and more essential.

Abstraction: though objects may not be representations of the same real world objects, they may be related by some derivable abstract concept representing user information requirements. This includes finding relations between objects, classification, generalisation, clustering, etc. based on similarity.

To support this usage, suitable operations have to be provided by the systems. This problem concerns for instance

- database management systems,
- data integration solutions,
- information and multi-media retrieval systems,
- expert systems,

and others, where the focus here will be on the former two. Therefore, the following discussions as well as the remainder of this thesis will heavily lean on concepts from database theory such as operations from the relational algebra.

This chapter will describe some of the foundations of similarity-based operations introduced in Chapters 5 and 6 and relate it to other approaches addressing the same or other relevant issues.

4.1 Introduction

Similarity as outlined in Chapter 3 requires severely different techniques than those used commonly in current systems to address the following issues:

- similarity relations contrary to equivalence relations on which for instance the relational model is based, do have **distinct properties** like atransitivity, and even symmetry and reflexivity may not hold,
- it heavily depends on the **context of similarity**, which is in most cases very specific to the current application and may be very complex to describe,
- the usage of similarity introduces a **probabilistic aspect** if for instance the measured similarity is used as a degree of truth for identification purposes.

Therefore, we consider the following levels of support for similarity-based operations.

Level 1 - Predicates: similarity predicates are used to decide about the similarity of data values and this way or through their logical combination about the similarity of objects like tuples. Similarity predicates therefore must comprehensively describe the context of similarity. The support for similarity predicates is currently becoming state of the art for instance by means of multi-media, spatial, and information retrieval extensions to database management systems. On the other hand, support for the similarity of simple data types like strings as described in Section 3.3 is not part of core database functionality. Because such support is a strong requirement in data integration, it is the major focus of this thesis.

Level 2 - Operations: due to the specific properties of similarity relations, operations usually based on equivalence relations must be revisited and adjusted to atransitivity and the possible occurrence of asymmetry and irreflexivity. Furthermore, the efficient processing of operations based on similarity-predicates may require the application of different algorithms and index structures on an internal level. From a database point of view this concerns selection, join, and grouping operations. For the latter, apart from handling atransitivity etc. the implicitly specified equivalence predicate must be replaced by an explicit similarity predicate. The impact of similarity relations on database operations is currently rarely considered in existing systems, mainly because a limited view on similarity and according operations – basically only selections – are supported.

Level 3 - Query and data model: the introduction of probabilistic aspects may require changes or extensions to the underlying query and data model of the system to express the possible vagueness of facts derived by similarity-based operations. Though this is currently not addressed in existing systems and, furthermore, not a focus of this thesis, the problem was addressed in research. In [DS96] Dey et al. propose an extended relational model and algebra supporting probabilistic aspects. Fuhr describes a probabilistic Datalog in [Fuh95]. Especially for data integration issues probabilistic approaches were verified and yielded useful results, as described by Tseng et al. in [TCY92].

Each level builds on the respective lower levels, such as similarity based operations only can be applied based on actual similarity predicates, and a probabilistic data model does only make sense, if according operations are supported. On the other hand, the support of one level does not necessarily imply any higher levels. Though similarity predicates to some degree may be supported by database systems, the processing of operations can be carried out in a conventional way, possibly with decreased efficiency or accuracy. And, similarity-based operations as proposed in this thesis can be used without any explicit modifications to the data model. So, the focus of this thesis will be on the levels 1 and 2 described above and introduced in more detail in the following sections.

4.2 Similarity Predicates

Similarity predicates were already introduced and described in Section 3.2 as

- basic similarity predicates conforming to a two-valued logic returning either true or false and may be derived from similarity or distance measures by applying a certain threshold, or alternatively as
- fuzzy similarity predicates returning a degree of truth between 0 and 1.

While the former can be supported by current database management systems, the latter require extended operations and implementations and possibly extended, probabilistic data models.

In this section the semantics of similarity predicates are described as extensions to the standard relational algebra assuming the following basic notations: let r be a relation with the schema $R = \{A_1, \ldots, A_m\}$, $t^r \in r$ is a tuple from the relation r and $t^r(A_i)$ denotes the value of attribute A_i of the tuple t^r.

If we furthermore distinguish between predicates defined between attributes, used for instance in join conditions, and those defined between an attribute and a constant value, as typically used in selection conditions, basic similarity predicates *<sim_pred>* can be specified as follows.

$$\textit{<sim\_pred>} := \begin{cases} sim(A_i, const) > l \\ dist(A_i, const)) \leq k \\ sim(A_i, A_j) > l \\ dist(A_i, A_j) \leq k \end{cases}$$

where the predicate is specified using either a normalised similarity or a distance measure according to the description in Section 3.2. The semantics of the similarity and distance predicates are as follows.

- Normalised similarity predicate on attribute and constant value:
  <sim_pred> $(t) \Leftrightarrow sim(t(A_i), const) > l$

- Distance predicate on attribute and constant value:
  <sim_pred> $(t) \Leftrightarrow dist(t(A_i), const)) \leq k$

- Normalised similarity predicate on attributes:
  <sim_pred> $(t^r, t^s) \Leftrightarrow sim(t^r(A_i), t^s(A_j)) > l$

- Distance predicate on attributes:
  <sim_pred> $(t^r, t^s) \Leftrightarrow dist(t^r(A_i), t^s(A_j)) \leq k$

The last two cases explicitly include the $r = s$ and even $i = j$. The latter is for instance useful when expressing a predicate for similarity-based grouping as introduced below, where the implicit equality predicate given in the conventional `GROUP BY`-clause must be replaced by a similarity predicate. We use $sim(t^r(A_i)) > l$ and $dist(t^r(A_i)) \leq k$ as shorthands for such predicates on one attribute within one relation.

Due to being based on two-valued logic, conventional predicates based on operators such as equality $=$, inequality $\neq$, or order comparison $<, \leq, >, \geq$, etc. can be considered conceptually on the same level. We use *<conv_pred>* as a shorthand for such conventional predicates. Both kinds can be combined freely through logical operators $\vee, \wedge$, and $\neg$. We refer to conditions containing at least one similarity predicate as similarity conditions <*sim_cond*>, i.e.

$$\text{<}sim\_cond\text{>} := \begin{cases} \text{<}sim\_pred\text{>} \\ \neg\ \text{<}sim\_cond\text{>} \\ \text{<}sim\_cond\text{>}\ \theta \begin{cases} \text{<}sim\_pred\text{>} \\ \text{<}conv\_pred\text{>} \end{cases} \end{cases}$$

where $\theta \in \{\wedge, \vee\}$. A special case considered for purposes of the evaluation of predicates during query processing are conjunctive similarity conditions, where only the logical conjunction operator $\theta = \wedge$ is used to combine predicates.

This basic concept of similarity predicates can be used in most current database management systems by applying user-defined functions for implementing similarity or distance measures. Yet, the recognition or explicit qualification of these similarity predicates is necessary if special support for similarity-based operations through algorithms and indexes is intended.

If this is the case, further considerations regarding the properties of the applied measure are required. Similarity predicates which are not reflexive or symmetric have a severe impact on the operations they are used in. Atransitivity must generally be considered. The consequences of this are discussed in relation to the operations later on. But, at this point it is necessary to mention that these aspects must be known to the system performing according operations. For system-defined predicates this is straightforward. For user-defined predicates, which will often be required due to the strong context dependence of similarity, there must be ways to declare these properties to the system.

Because the previous definition of predicates is an extension of the standard relational algebra, we do not have to deal with probabilities in conditions – by using a similarity threshold we can always rely on boolean values of $true$ or $false$ for such predicates and derived complex conditions. The alternatively considered approach of using fuzzy similarity predicates returning a degree of truth between 0 and 1 would require a special handling of complex conditions, for which a probabilistic result must be derived. Given two fuzzy predicates p and q, two often used ways of computing this score are:

Minimum/maximum combination which is for instance applied during query optimisation when dealing with selectivities in commercial database management systems:

$$\begin{aligned} P(p \wedge q) &= min(P(p), P(q)) \\ P(p \vee q) &= max(P(p), P(q)) \\ P(\neg p) &= 1 - P(p) \end{aligned}$$

Probabilistic combination assuming independence between the predicates as for instance used in Information retrieval, probabilistic database approaches like the one by Fuhr in [Fuh95], or data integration approaches like Cohen's WHIRL described in [Coh98]:

$$\begin{aligned} P(p \wedge q) &= P(p)P(q) \\ P(p \vee q) &= 1-(1-P(p))(1-P(q)) \\ P(\neg p) &= 1-P(p) \end{aligned}$$

Throughout this thesis the latter approach is used. A discussion of the approaches regarding data integration is given by Cohen in [Coh98]. The integration of predicates which are not fuzzy can easily be done by assigning the values 0 and 1 if the result is $false$ or $true$, respectively.

The score of such a complex condition including fuzzy predicates is again between 0 and 1 and can for instance be used to specify a global threshold for the condition instead of the single fuzzy predicates. To gain the expressive power when using thresholds for each fuzzy predicate a weighting of predicates would have to be introduced. Alternatively, the score can be used for further processing, such as for ranking the results for an output, or for ordered and pipelined processing in following operations.

4.3 Similarity-based Operations

Based on similarity predicates as the common notion of similarity on the data level, similarity-based operations can be introduced. In principle, only predicate-based operations like the θ-join and selection have to deal with aspects of similarity. On the other hand, operations like the natural join, grouping, and union with duplicate elimination are based on implicit equivalence relations within the domains of all attributes of a relation, single attributes, or two attributes of the same name. In this section we describe the basic semantics of similarity based operations discussed in Chapters 5 and 6, i.e. selection, join, and grouping as known from the relational algebra or according extensions. Though aspects of a similarity union in conjunction with similarity-based duplicate removal were considered during early research related to this thesis, the concept of grouping and aggregation better fit the two aspects of duplicate detection and the according reconciliation of discrepant values.

4.3.1 Similarity-based Selection

Selections based on similarity conditions including basic similarity predicates, which are mainly used in this thesis, have the general form

$$\sigma_{<sim\_cond>} r(R) := \{t \mid t \in r \wedge <sim\_cond> (t) = \textbf{true}\}$$

To simplify discussions in the later Chapters of the thesis similarity conditions are either seen as conjunctive similarity conditions – for instance as a result of a transformation to a disjunctive normal form without losing the generality of the approach – or as simple constant selection. Then, a conjunctive similarity selection is

$$\sigma_{\bigwedge_{i=1}^{n} <sim\_pred>_i} r(R) := \{t \mid t \in r \wedge \forall i = 1..n : <sim\_pred>_i(t) = \textbf{true}\}$$

Regarding complex similarity conditions, another aspect typical to the evaluation of similarity predicates has to be pointed out. A common approach to efficiently process similarity-based operations is to provide a pre-selection which is specified as part of the condition but evaluated before the probably expensive evaluation of the similarity predicates takes place. This is even more effective if the pre-selection predicate is supported by a conventional index structure. The pre-selection in the most simple case can be specified by a user familiar with the internal processing of the query. A number of research approaches, including the one presented in Chapter 6 of this thesis, consider the automatic expansion of similarity predicates to similarity conditions including pre-selection predicates which can be evaluated efficiently.

Similarity predicates may include the similarity between attributes, as later on used for similarity joins, or as predicates comparing attribute values to a constant specified within the query according to constant selections in the relational algebra. Such similarity constant selections on an attribute $A \in R$ of relation $r(R)$ are either based on a similarity measure

$$\sigma_{sim(A,constant)>t} r(R) := \{t \mid t \in r \wedge sim(t(A), constant) > l\}$$

or on a distance measure

$$\sigma_{dist(A,constant)\leq k} r(R) := \{t \mid t \in r \wedge dist(t(A), constant) \leq k\}$$

as introduced above.

The latter conforms to a *range* or ε-*range query* as known from spatial data access and information and multimedia retrieval. Other approaches in these fields related to similarity selections are nearest neighbour queries and k-nearest neighbour queries, which are also based on distance measures and described for instance in [SL91] and [BYRN99]. Instead of returning reasonably similar or close objects, such queries return either the 1 or k closest objects to a query point, in our case represented by the constant value. A more recent type of distance-based queries are skyline queries as described for instance by Kossmann et al. in [KRR02], where the result of a query is a set of objects and each of them represents an optimal solution regarding a combination of criteria expressed as dimensional values.

Another aspect typical to information and multimedia retrieval is to apply the fuzzy nature of similarity predicates by not presenting a boolean result, i.e. tuples either are or are not element of an unordered result set, but instead use the result of predicate evaluation to present a possibly truncated ordered result list. In this case, thresholds are not part of the predicate as in our notion of fuzzy similarity predicates, and the algorithms and operations have to be adapted to this modification of the retrieval data model. An overview of according techniques is given for instance in [BYRN99].

Finally, for a similarity-based selection the consequences of properties a similarity measure used to specify a predicate has have to be considered. While on the conceptual level neither the lack of constancy of self-similarity and symmetry nor violations of the triangular inequality have any consequences regarding the semantics of the operation, a missing notion of closeness as for instance expressed through the triangular inequality or by monotone proximity structures as introduced in Section 3.2.3 may hinder the efficient processing of the predicate, because supporting index structures are not conceivable in this case.

4.3.2 Similarity-based Join

Based on similarity conditions introduced above the semantics of a similarity join between two relations $r_1(R_1)$ and $r_2(R_2)$ can be described in a straightforward way for a given similarity condition <*sim_cond*> as

$$\begin{aligned} r_1 \tilde{\bowtie}_{<sim\_cond>} r_2 := \{t \mid \quad & t(R_1 \cup R_2) \wedge \\ & \exists t_1 \in r_1 : t_1(R_1) = t(R_1) \wedge \exists t_2 \in r_2 : t_1(R_2) = t(R_2) \wedge \\ & <sim\_cond>\ (t_1, t_2) = \textbf{true}\} \end{aligned}$$

This simply means, the concatenation of a pair of tuples from the relations r_1 and r_2 appears in the result of the join operation if the similarity condition is fulfilled for these two tuples. There is a slight simplification in this description by assuming non-overlapping schemas R_1 and R_2, which can always be realised by considering the relation name as part of the attribute namespace. The semantics of a similarity join as given above conforms to a θ-join, only differing in the kind of predicates allowed.

Just like for selections we consider the following cases of simplified join conditions consisting of only one similarity or distance predicate, i.e.

$$\begin{aligned} r_1 \bowtie_{sim(A_i,A_j)>l} r_2 := \{t \mid \quad & t(R_1 \cup R_2) \wedge \\ & A_i \in R_1 \wedge A_j \in R_2 \wedge \\ & \exists t_1 \in r_1 : t_1(R_1) = t(R_1) \wedge \exists t_2 \in r_2 : t_1(R_2) = t(R_2) \wedge \\ & sim(A_i,A_j) > l\} \end{aligned}$$

and

$$\begin{aligned} r_1 \bowtie_{dist(A_i,A_j)\leq k} r_2 := \{t \mid \quad & t(R_1 \cup R_2) \wedge \\ & A_i \in R_1 \wedge A_j \in R_2 \wedge \\ & \exists t_1 \in r_1 : t_1(R_1) = t(R_1) \wedge \exists t_2 \in r_2 : t_1(R_2) = t(R_2) \wedge \\ & dist(A_i,A_j) \leq k\} \end{aligned}$$

respectively. When similarity joins are addressed in research, as for instance in the approaches described later on, the description most often refers to these limited interpretations, basically because most research is focused on the evaluation of one specific similarity predicate. Though the research presented in this thesis is also focused on specific predicates, namely string similarity as expressed based on the edit distance, in Chapter 5 a discussion of complex join conditions is included.

Spatial and similarity joins were first addressed for data values that either represented points in a multidimensional metric space or could be mapped to such a space, e.g. by Brinkhoff et al. in [BKS93] and Shim et al. in [SSA02]. A recent overview is given by Koudas and Sevcik in [KS00]. Based on Fuhr's probabilistic Datalog ([Fuh95]) in [Coh98] Cohen described an approach for performing joins based on textual similarity, contrary to the similarity of shorter strings used in this thesis. In [GIJ$^+$01] and [GIKS03] Gravano et al. present and refine an approach to perform joins based on similarity of string attributes through efficient pre-selections of materialised q-grams.

Contrary to the previously introduced selection, the properties of similarity measures used to specify predicates for a join may have a severe impact on the semantics of the operation, mainly due to missing constancy of self-similarity and symmetry. If a similarity measure is defined in a way such that the constancy of self-similarity does not hold, the resulting similarity relation may not be reflexive, i.e. $x = y \not\Rightarrow SIM(x,y)$ and this way one object may or may not match itself, which for instance can occur during self-joins. Furthermore, irreflexivity, i.e. $x = y \Rightarrow \neg SIM(x,y)$, may be a requirement, if similarity should be handled separately from identity as described in Section 3.2.3. This requires slight modifications and a minor lack of optimisation opportunities during join processing.

The asymmetry of a similarity measure is far more problematic, because it may imply an asymmetric similarity relation resulting in the following semantic problem of the similarity join:

$$r_1 \tilde{\bowtie}_{<sim\_cond>} r_2 \neq r_2 \tilde{\bowtie}_{<sim\_cond>} r_1$$

Many known optimisations are based on the commutativity of the join operator, for instance changing the join order for multi-way joins or choosing a small input relation as the left operand for index-based joins. This may not be possible, if an asymmetric similarity measure is involved.

Therefore, the system processing similarity-based joins needs to be aware of the properties of the underlying similarity relation either by declaration or by implicitly assuming the worst case for similarity-based operations.

4.3.3 Similarity-based Grouping

The semantics of the grouping operator are defined based on an extension of the relational algebra for standard grouping as presented in [EN94]:

$$_{<grouping\_attrs>}\mathcal{F}[<aggr\_func\_list>](r)$$

Here $<grouping\_attrs>$ is a list of attributes used for grouping relation r, $<aggr\_func\_list>$ denotes a list of aggregate functions (e.g., count, avg, min, max etc.) conveyed by an attribute of relation r. Because the proposed operation is intended for data integration scenarios, advanced aggregation functions suitable for the reconciliation of discrepant values are discussed later on in this thesis. As a further simplification, we assume that the name of an aggregated column is derived by concatenating the attribute name and the name of the function. An aggregate function f is a function returning a value $v \in \text{Dom}$ for a multi-set of values $v_1, \ldots v_m \in \text{Dom}$:

$$f(\{\!|v_1, \ldots, v_m|\!\}) = v$$

where Dom denotes an arbitrary domain of either numeric or alphanumeric values and the brackets $\{\!| \ldots |\!\}$ are used for multi-sets.

We extend this equality-based grouping operator $\mathcal{F}$ with regard to the grouping criteria by allowing a similarity condition and call this new operator Γ:

$$_{<sim\_cond>}\Gamma[<aggr\_func\_list>](r)$$

This operator again has a list of aggregate functions $<aggr\_func\_list>$ with the same meaning as above. However, the grouping criteria $<sim\_cond>$ is now a complex similarity condition as introduced above.

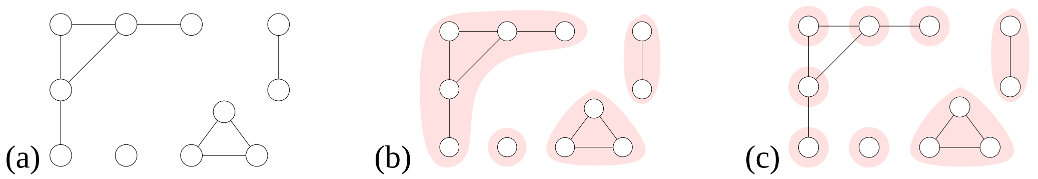


Figure 4.1: Derived equivalence relation for a given similarity relation (a) by (b) transitive and (c) strict similarity

The result of Γ is a relation r' where the schema consists of all the attributes referenced in *<sim_cond>* in equality predicates and the attributes named after the aggregates as described above. Contrary to the usual grouping operator, if a similarity predicate is specified in the grouping condition the according attribute values also have to be reconciled during aggregation, because of the conflicting values.

The relation r' is obtained by the concatenation of the two operators γ and ψ which reflect the two steps of grouping and aggregation. The first operator $\gamma_{<sim\_cond>}(r) = \mathcal{G}$ produces a set of groups $\mathcal{G} = \{G_1, \ldots, G_m\}$ from an input relation r. Each group is a non-empty set of tuples with the same schema R of the input relation r. The second operator $\psi_{A_1,\ldots,A_l,<aggr\_func\_list>}(\mathcal{G}) = r'$ aggregates the attribute values of all tuples from each group and produces exactly one tuple for each group of $\mathcal{G}$ according to the given aggregate functions. Thus, it holds $\forall G \in \mathcal{G}$ with $G = \{t_1^G, \ldots, t_n^G\}$ there is exactly one tuple $t^{r'} \in r'$ with

$$\forall i = 1 \ldots l : t^{r'}(A_i) = t_1^G(A_i) = t_2^G(A_i) = \cdots = t_n^G(A_i)$$

where $A_1, \ldots, A_l$ are attributes referred by the equality predicates of the similarity condition, (i.e., for these attributes all tuples have the same value), and for the remaining attributes either referenced in a similarity predicate or not referenced in the grouping condition

$$\forall j = 1 \ldots m : t^{r'}(A_j) = f_j(\{\!\{t_1^G(A_j), \ldots, t_n^G(A_j)\}\!\})$$

where $f_1, \ldots, f_m$ are aggregate functions from *<aggr_func_list>*. Based on these two operators we can define the Γ operator for similarity-based grouping as follows:

$$_{<sim\_cond>}\Gamma[<aggr\_func\_list>](r) = \psi_{A_1,\ldots,A_l,<aggr\_func\_list>}(\gamma_{<sim\_cond>}(r))$$

Except for the different handling of attributes referenced in similarity predicates, so far this corresponds to the semantics of the standard grouping operation.

But, we have not yet dealt with the fact, that the partitioning of the input relation r into the set of groups $\mathcal{G}$ implies the requirement of an equivalence relation

EQ within r, though the similarity condition <*sim_cond*> may imply a similarity relation SIM that will not provide transitivity. Therefore, throughout this thesis we use the simple strategy of constructing an equivalence relation SIM_{EQ} by building the transitive closure $SIM_{EQ} := SIM^{+}$, i.e. the partitions of our relation r in $\mathcal{G}$ are maximal sets of tuples that are similar according to <*sim_cond*> either directly or indirectly. A more rigid but still simple approach considered during early research on this thesis is to establish SIM_{EQ} such that pairwise similarity of all objects in a partition is required. We refer to the latter as the *strict* strategy. Both strategies are outlined in Figure 4.1. As an example, consider the strings "ODBMS", "OODBMS", and "DBMS". With an allowed edit distance threshold of 1 they would all be found similar, if we apply the transitive closure strategy, but this would not be the case if the strict strategy is applied, because "DBMS" and "OODBMS" have a distance of 2 without the connection via "ODBMS". Related to entity identification, record linkage, etc. a number of other approaches to address this problem were considered. Centroid or density-based clustering techniques proved to be useful strategies for dealing with atransitivity and provide a high level of accuracy, as for instance described in [LMP01b] and [ME97].

According to the transitive partitioning strategy, we can further refine the semantics of our similarity-based grouping operator. All tuples t_i^G of a group $G \in \mathcal{G}$ have to be transitively similar to each other regarding the similarity condition <*sim_cond*>:

$$\forall G \in \mathcal{G} : \forall t_i^G, t_j^G \in G : t_j^G \in tsim_{<sim\_cond>}(t_i^G)$$

where $tsim_{<sim\_cond>}(t)$ denotes the set of all tuples which are in the transitive closure of the tuple t with regard to *sim_cond*:

$$\begin{aligned} tsim_{<sim\_cond>}(t) = \{t' \mid\ & sim\_cond(t,t') = \textbf{true} \lor \\ & \exists t'' \in tsim_{<sim\_cond>}(t) : sim\_cond(t',t'') = \textbf{true}\} \end{aligned}$$

and no tuple is similar to any other tuple of other groups

$$\begin{aligned} \forall G_i, G_j \in \mathcal{G}, i \neq j : \forall t_k^{G_i} \in G_i\ \nexists t_l^{G_j} \in G_j : \\ sim\_cond(t_k^{G_i}, t_l^{G_j}) = \textbf{true} \end{aligned}$$

Figure 4.2 illustrates the application of these operators for a simple example query:

$$_{diff(A_1) \leq 0.2}\Gamma[\mathrm{avg}(A_1), \mathrm{min}(A_2)](r)$$

The input relation r based on a schema R consisting of two attributes A_1, A_2 has to be grouped by similar values of A_1, e.g. using the approximation condition "$diff(A_1) \leq 0.2$".

A_1	A_2
1.0	5
1.1	6
2.0	7
2.1	8
2.2	4

$\xrightarrow{\gamma}$

	A_1	A_2
G_1	1.0	5
	1.1	6
G_2	2.0	7
	2.1	8
	2.2	4

$\xrightarrow{\psi}$

A_1avg	A_2min
1.05	5
2.1	4

Figure 4.2: Application of the grouping operator

In the first step, the γ operator produces two groups G_1 and G_2. Let us now assume an aggregation function list "avg(A_1), min(A_2)". Then, the ψ operator derives for each of these groups a single tuple as shown in the table at the right-hand side.

The importance of extended concepts for grouping and aggregation in data integration was emphasised by Hellerstein et al. in [HSC99]. Though the problem of atransitivity during duplicate detection was for instance addressed by Galhardas et al. in [GFSS00], the work presented in this thesis is the only one based on an extended grouping operator. User-defined aggregation (UDA) are part of the current version of the SQL standard and are now supported by several commercial database systems, e.g. Oracle9i, IBM DB2, Informix. In [WZ00] the SQL-AG system for specifying UDA is presented, which translate to C code and the usage of this approach called AXL in data mining is discussed. The approach presented here builds on this work for the purpose of data reconciliation as described in Chapter 5.

4.4 Conclusions

In this chapter the foundations for similarity-based operations described regarding their implementation and efficiency in Chapters 5 and 6 were introduced. First, a general introduction to the purpose and level of support for such operations was given. It was pointed out, that in this thesis similarity is addressed on the predicate and operation level and further aspects of data models supporting similarity and probabilistic aspects are beyond the scope of the presented work.

Based on a general introduction given in Section 3.2 the scope for similarity predicates considered in the later chapters was adjusted. Furthermore, the specification of complex similarity conditions and aspects of their evaluation were described. For this purpose, special requirements regarding fuzzy predicates were included in the discussion.

Based on similarity predicates and similarity conditions, a number of similarity-based operations and their semantics were described as extensions to the relational algebra. This includes similarity-based selection, join, and grouping.

A simple formal description of the semantics of similarity-based selections was given and related to other approaches for similarity- and distance-based queries. The semantics of the join operation introduced subsequently are quite straightforward, too, but here the effect of the properties of similarity relations such as possible irreflexivity and asymmetry have to be considered.

Atransitivity, which is a typical property of similarity relations, is especially critical for similarity-based grouping. This operation, which was also formally described, is an approach to address problems of duplicate identification and reconciliation between conflicting data values.

The view on similarity predicates and similarity-based selections and joins presented in this chapter mostly conforms to a condensation of approaches currently considered and referred to in the respective sections. The introduction of similarity-based grouping is new and, therefore, was discussed in more detail.

Chapter 5

Similarity-based Operations for Materialised Data

In the previous chapter the foundations of similarity predicates and similarity-based operations were introduced. In this chapter implementations of two of these operations – join and grouping – are introduced for the specific case of materialised data sets. While on the one hand extensions to query languages are proposed and used to illustrate possible queries, and were also implemented as part of the query language FRAQL introduced by Sattler et al. in [SCS00], the actual implementation described was done using the extensibility interface of the commercial DBMS Oracle8i. Furthermore, a string similarity predicate supported by a trie index with specialised algorithms to support similarity-based lookup is used. Based on this implementation the operations are evaluated and general aspects of the efficiency of similarity-based operations are discussed. Moreover, further considerations regarding the usage of similarity predicates and reconciliation aspects are described.

5.1 Introduction

The target scenario of the implementations presented in this Chapter are materialised integration scenarios, i.e. the input data of the operations is either

- fully materialised on secondary storage managed by a database management system, such as for instance it would be the case in a staging area of a Data warehouse system, or

- materialised in main memory as a temporary result of previous operations, which may include results of distributed query processing in virtual integration scenarios.

Nevertheless, contrary to the approach presented in Chapter 6 the evaluation of the operation and the underlying similarity predicates cannot be processed in a distributed way across various data sources.

Accordingly, the operations can be used in the same way in centralised DBMS as well as virtual integration solutions like FDBMS or mediators. Both approaches were implemented during the research on this thesis, but only the former is presented here in detail and used for evaluation purposes. On the other hand, the proposed join and grouping operations have a significant difference to common operations used in database management systems: we can rely on equality of attribute values only in some cases, and have to deal with discrepancies in data objects representing the same or related real-world objects. Such discrepancies can exist due to errors made during data input or different conventions for data representation, and have to be addressed on the predicate level and by specifying possibly complex similarity conditions as introduced in Section 4.2.

As such, the implementation of the proposed similarity-based operations can for instance be used for duplicate elimination as a sub-task of data cleaning. Assuming SQL-based systems, the natural choice for duplicate elimination is the `GROUP BY` operator using the key attributes of the tuples in combination with aggregate functions for reconciling divergent non-key attribute values. However, this approach is limited to equality of the key attributes and, therefore, has to be extended according to the description given in Chapter 4. The same is true for linking complementary data, which in a SQL system would be done based on equality by the **`join`** operator. Both operations are based on extended concepts for similarity-based predicates. Major concerns to be considered during their implementation are the new requirements resulting from the characteristics of similarity relationships, most of all atransitivity, and support for the efficient processing of similarity predicates.

As outlined before, efficiency of operations and especially similarity-based operations used in data integration is very important, because the amount of data to be processed can be equal to or even greater than from a single source. Apart from the used algorithms to implement a certain operation, the efficiency mainly depends on the evaluation of similarity-predicates. For certain predicates such as the one based on the string edit distance used here, index support is possible, because the metric axioms and most of all the triangular inequality and its implied notion of closeness provide the basis for access paths. Yet, the measurable complexity of similarity-based operations contrary to according algorithms and index structures of operations based on equivalence relations also depends on other criteria such as the similarity or distance threshold applied in a predicate. This is outlined in more detail in the evaluation presented in Section 5.4 of this chapter.

Concurrent and related approaches to the work presented in this chapter include the WHIRL system and language providing similarity-based joins described in [Coh98] by Cohen, which is based on Fuhr's work on a probabilistic Datalog described in [Fuh95]. The WHIRL system uses text-based similarity and logic-based data access as known from Datalog to integrate data from heterogeneous sources. Cohen describes an efficient algorithm to compute the top scoring matches of a ranked result set. The implementation of the similarity predicate uses inverted indices common in the field of information retrieval.

Contrary to the WHIRL approach, the approach presented here is based on the similarity of string attributes, as introduced in Section 3.3 of this thesis. Though the concept of string similarity is covered by comprehensive research, the efficient and index-based evaluation in large data sets – for instance managed in database management systems – is a current research topic. In [GIJ$^+$01] Gravano et al. present an approach concurrent to the one presented here, where for similarity-based joins on string attributes an efficient pre-selection based on q-grams is used for optimisation. In short, the approach is based on down-sizing the data sets on which a similarity predicate is evaluated by first doing an equality-based join on substrings of fixed length q. The authors extend and modify this approach to support string tokens based on techniques similar to those used by Cohen in [Coh98] in [GIKS03]. Both approaches require fully materialised data sets and index structures, hence they are not applicable in virtual integration scenarios and introduce a huge space overhead.

Though the basic framework of predicates and operations described in Chapter 4 is not limited to string based predicates, we implemented an edit distance string similarity predicate using a trie as an index structure based on results by Shang and Merret described in [SM96] for evaluation purposes.

Other work related to the contents of this chapter which was not previously mentioned regards duplicate detection, which is addressed in various research areas like database and information system integration [ZHKF95, LSPR93], data cleaning [CdGL$^+$99, GFSS00], information dissemination [YGM95], and others. Early approaches were merely based on the equality of attribute values or derived values. Newer research results deal with advanced requirements of real-life systems, where identification very often is only possible based on similarity. Those approaches include special algorithms [ME96, HS95], the application of methods known from the area of data mining and even machine learning [Li95]. Other interesting results came from specific application areas, like for instance digital libraries [GBL98, Hyl96]. While these approaches are mostly very specific regarding certain applications, the goal here is to provide a more general view on the process of duplicate detection, that may well include these approaches on the predicate level.

An overview of problems related to entity identification is given in [Ken91]. In [LSPR93] Lim et al. describe an equality based approach, include an overview of other approaches and list requirements for the entity identification process. Monge and Elkan describe an efficient algorithm that identifies similar tuples based on a distance measure and builds transitive clusters in [ME97]. In [GFSS00] Galhardas et al. propose a framework for data cleaning as a SQL extension and macro-operators to support among other data cleaning issues duplicate elimination by similarity-based clustering. The similarity relationship is expressed by language constructs, and furthermore, clustering strategies to deal with transitivity conflicts are proposed. Luján-Mora and Palomar propose a centroid method for clustering in [LMP01b]. Furthermore, they describe common discrepancies in string representations and derive a useful set of pre-processing steps and extended distance measures combining edit distance on a token-level and similarity of token sets. In [HS95] Hernández et. al. propose the sliding window approach for similarity-based duplicate identification where a neighbourhood conserving key can be derived and describe efficient implementations.

5.2 Principles of the Implementation and Optimisation

In this section the principles of the implementation and optimisation of similarity-based join and grouping operations introduced in Chapter 4 are outlined for the previously described scenario of materialised data integration. For an efficient realisation dedicated plan operators are required, which implement the described semantics. That means for instance for the similarity join, even if one formulates a query as follows

```
select *
from r1, r2
where edist(r1.title, r2,title) < 2
```

the similarity join implementation exploiting special index support and considering the special semantics of the predicates has to be chosen by the query optimiser instead of computing the Cartesian product followed by a selection. In the case of similarity grouping, a simple user-defined function is not sufficient as grouping function, because during similarity grouping the group membership is not determined by one or more of the tuple values but depends on already created groups. In addition, processing a tuple can result in merging existing groups.

Thus, in the following the implementation of these two plan operators SIMJOIN and SIMGROUPING are described, assuming that the query optimiser is able to recognise the necessity of applying these operators during generating the query plan. This could be supported by appropriate query language extensions, e.g. for the similarity join

```
select *
from r1 similarity join r2
     on edist(r1.title, r2,title) <= 2
```

and for the similarity grouping this could be formulated as follows:

```
select *
from r1
group by similarity on edist(title) <= 2
```

For evaluation purposes we used an index-supported similarity predicate on string attributes using edit distance and tries, that is also described briefly. The following description refers to conjunctively combined, reflexive, and symmetric similarity predicates and the transitive closure strategy for the grouping operator introduced in the previous chapter. More complex similarity conditions and required changes are briefly discussed.

5.2.1 A Trie-based Similarity Predicate for Strings

At first, the edit distance predicate and an according index structure used throughout this chapter is shortly introduced. In this approach a similarity predicate consists of a distance measure and an according threshold. Hence, the index lookup performed requires the actual value $t^r(A_i)$ of an involved attribute A_i, the indexed attribute A_j and the threshold k as

$$dist(t^r(A_i), t^s(A_j)) \leq k$$

where $i = j$ and $r = s$ are included as special cases, applicable for instance for grouping predicates. Currently, for the implementation given here the focus is on edit distances as the primary similarity measure. For this purpose, the approach proposed in [SM96] of using a trie in combination with a dynamic programming algorithm for computing the edit distance was adopted.

The main idea of their approach is to traverse the trie containing the set of string attribute values of all tuples indexed in the trie in depth-first order trying to find a match with the search pattern, i.e., the attribute value of the currently processed tuple (Algorithm 1). Because the similarity predicate does imply an approximate match with a maximum of k differences instead of an exact match, we must not stop the traversal after a mismatch is found. Instead, an edit operation (insert, remove, or replace) is assumed and the search is continued in the child nodes of the current trie node. The current number of found differences is stored for each search path. Only after exceeding the given threshold, the traversal of a given path can be stopped, and the search can go back to the next sub-trie. Hence, the threshold is used for cutting off sub-tries containing strings not similar to the pattern.

In addition, the effort for computing the dynamic programming tables required for determining the edit distance can be reduced, because all strings in one subtree share a common prefix and therefore the same edit distance. We omit further details of this algorithm and refer instead to the original work.

Algorithm 1: *Approximate trie searching*

```
Globals
  Threshold $k$
  Pattern string $p$, target string $w$

Procedure approxSearch(TrieNode $n$, int $level$)
  begin
    for all child nodes $c$ of $n$
      $w[level]$ := character $z$ of $c$
      if $c$ is a leaf node $\wedge$ $edist(w, p, level) < k$
        output tuple-ids for node $c$
      if $edist(w, p, level) > k$
        continue /* cut off */
      approxSearch ($c$, $level$ + 1)
    end for
  end
```

In the following implementations of the previously introduced operations tries are created on the fly for each grouping attribute or join predicate which appears together with an edit distance predicate. Such a trie stores not only the actual string values but also the tuple-id of the associated tuple. Therefore, besides inserting new string values no updates on the trie are necessary.

5.2.2 Similarity-based Join

The implementation of a similarity join outlined in this section is quite straightforward. Like for conventional join operators index support for predicates can be exploited to improve performance by reducing the number of pairwise comparisons. However, the different predicates of a similarity expression require different kinds of index structures:

- For equality predicates common index structures like hash tables or B-trees can be utilised.
- Simple numeric approximation predicates like $diff(A_i, A_j) \leq k$ can also be supported by B-Trees.
- For string similarity based on edit distances $edist(A_i, A_j) \leq k$ tries are a viable index structure, as previously introduced.
- For the other similarity predicates discussed in Chapter 3 index support is given for instance through multi-dimensional indexes like R-trees and its derivatives on data mapped to a metric space.

Given such index structures a join algorithm can be implemented taking care of the various kinds of indexes. In Algorithm 2 a binary join for two relations r_1 and r_2 is shown, assuming that indexes for relation r_2 either exist or were build on the fly in a previous processing step. The result of this algorithm is a table of matching tuples for usage described later on. Alternatively, result tuples can be produced for pipelined query processing directly at this point. The notations I_{p_i} and k_{p_i} refer to the index on predicate p_i and the specified threshold, respectively. A_{p_i} refers to the involved attribute.

Algorithm 2: *Processing a tuple from join relation r_1 during similarity join*

```
 Globals
   Conjunctive join condition $c = p_1 \wedge \ldots \wedge p_n$
   Set of indexes $I_{p_i}, 1 \leq i \leq n$ on join relation $r_2$
       for index supported predicates
   Mapping table $tid\_tid$ for matching tuples

Procedure processTuple(Tuple $t$)
   begin
       for all index supported equality predicates $p_i$
           set of tuples $s_{conj} := indexScan(I_{p_i}, t(A_{p_i}))$
       end for
       for all index supported similarity predicates $p_i$
           $s_{conj} := s_{conj} \cap indexScan(I_{p_i}, t(A_{p_i}), k_{p_i})$
       end for
       for all tuples $t_l \in s_{conj}$
           boolean $similar$ := true
           for all non-index supported predicates $p_i$
```

```
            $similar := similar \wedge$
                $evaluate(p_i, k_{p_i}, t(A_{p_i}), t_l(A_{p_i}))$
            **if** not $similar$ **break**
        **end for**
        **if** similar insert ($t$,$t_l$) in $tid\_tid$
    **end for**
**end**
```

As a side note, more complex similarity conditions could easily be supported by adding disjunctions. The similarity condition c can be transformed to disjunctive normal form. For all conjunctions of $c = \bigvee_{i=1}^{m} conj_i$ the s_{conj_i} are computed and the set of relevant groups would be $s_{disj} = \bigcup_{i=1}^{m} s_{conj_i}$.

5.2.3 Similarity-based Grouping

Like the join operator, the similarity-based grouping operator is based on the efficient evaluation of similarity predicates, but in addition has to deal with problems arising from the atransitivity of similarity relations, as previously outlined in Section 4.3.3. A naive implementation of the similarity-based operator would work as follows:

1. Iterate over the input set and process each tuple by evaluating the similarity condition with all previously processed tuples. Because these tuples were already assigned to groups, the result of this step is a set of groups, where – assuming the transitive closure strategy – in each group there is at least one tuple similar to the current tuple.

2. If the result set is empty, a new group is created containing only this current tuple.

3. If one group is found, the current tuple is added to this group.

4. Otherwise, i.e. more than one group is found, the conflict is resolved by merging the found groups according to the transitive closure strategy.

Obviously, this naive implementation would lead to $O(n^2)$ time complexity for an input set of size n. Similar to processing a similarity join we assume that there are index-supported predicates for equality and similarity, and predicates that cannot be supported by indexes. Therefore, the following optimised Algorithm 3 was implemented. Please note that this algorithm implements only the γ operator as described in Section 4.3.3, because the ψ operation corresponds to the traditional projection/aggregation operation.

Algorithm 3: *Processing a tuple during similarity grouping*

```
 Globals
   Conjunctive similarity condition $c = p_1 \wedge \ldots \wedge p_n$
   Set of indexes $I_{p_i}, 1 \leq i \leq n$
       for index supported predicates
   Mapping table $gid\_tid$ assigning tuples to groups

Procedure processTuple(Tuple $t$)
   begin
       set of groups $r_{conj}$ := all groups from $gid\_tid$
       for all index supported equality predicates $p_i$
           set of tuples $s := indexScan(I_{p_i}, t(A_{p_i}))$
           $r_{conj} := r_{conj} \cap gid\_tid(s)$
       end for
       for all index supported similarity predicates $p_i$
           set of tuples $s := indexScan(I_{p_i}, t(A_{p_i}), k_{p_i})$
           $r_{conj} := r_{conj} \cap gid\_tid(s)$
       end for
       for all groups $g_j \in r_{conj}$
           boolean $member$ := false
           for all tuples $t_l \in g_j$
               boolean $similar$ := true
               for all non-index supported predicates $p_i$
                   $similar := similar \wedge$
                       $evaluate(p_i, k_{p_i}, t(A_{p_i}), t_l(A_{p_i}))$
                   if not $similar$ break
               end for
               $member := member \vee similar$
               if $member$ break
           end for
           if not $member$ $r_{conj} := r_{conj} - g_j$
       end for
       if $r_{conj} = \emptyset$ group $g$ := new group in $gid\_tid$
       else group $g$ := merge all $r_{conj}$ in $gid\_tid$
       insert $t$ in $g$
   end
```

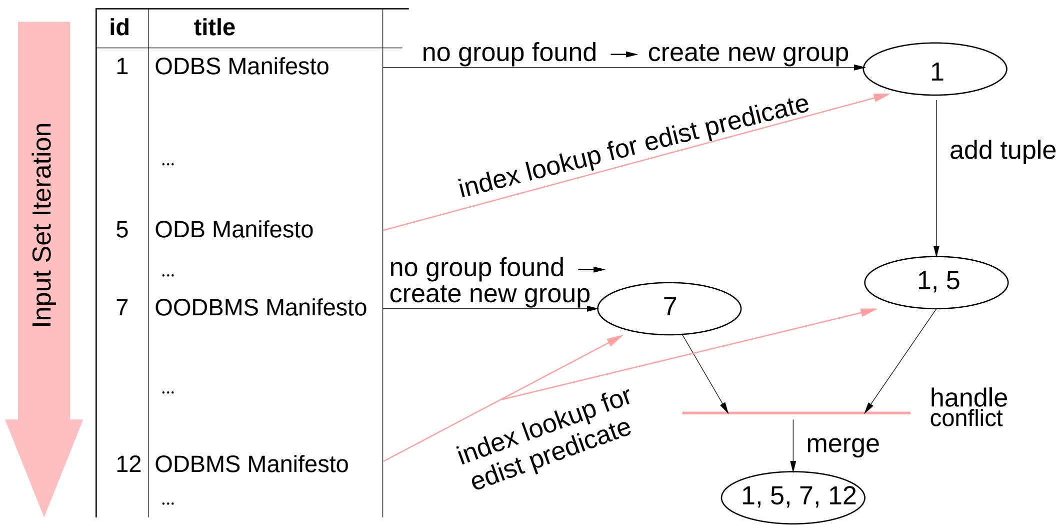


Figure 5.1: Building groups applying index support

Similar to join processing, for each tuple *t* the algorithm tries to find a minimal set r_{conj} of groups that relate to *t* by applying index-supported equality and similarity predicates first. This can even be improved, if information about the costs and selectivity of the index-based predicate evaluation exist and an according processing order is established. A pairwise comparison is only performed for tuples from this small subset of groups in the second half of the algorithm. As a result of this procedure the mapping table *gid_tid* is adjusted to the newly found group structure.

The index-based processing of such a predicate in the context of similarity-based grouping is illustrated in Figure 5.1 for the simple query

```
select pick_where_eq(src='DBLP', title)
from A union all B union all C
group by transitive similarity
on edist(title) < 2
```

As there are no previous tuples a new group is created for tuple **1**. During processing of tuple **5** tuple **1** is found using the trie on the **title** attribute, where the edit distance is 1 for the two tuples. When tuple **7** is processed, no match can be found, because the edit distance to previous tuples is at least 2, namely for both of the two tuples **1** and **5**. For tuple **12** two relevant groups are found and the conflict has to be resolved by merging the groups according to the strategy of transitive similarity.

5.3 Implementation using Oracle8i

The described similarity-based operations were implemented as extensions to the commercial DBMS Oracel8i. To implement such operations in a SQL DBMS as native plan operators supporting the typical iterator interface [Gra93] requires significant modifications to the database engine and therefore access to the source code. So, in order to add these operations to a commercial system the available programming interfaces and extensibility mechanisms should be used instead. Most modern DBMS support so-called table functions which can return tables of tuples, in some systems also in a pipelined fashion. In this way, the proposed operations can be implemented as table functions consuming the tuples of a query, performing the appropriate similarity operation and returning the result table.

For example, a table function `sim_join` implementing Algorithm 2 and expecting two cursor parameters for the input relations and the similarity join condition could be used as follows:

```
select *
from table (sim_join (cursor(select * from data1),
         cursor(select * from data2),
         'edist (data1.title, data2.title) <= 2'))
```

This query performs a similarity join with one similarity predicate on the title attributes in two given relations, where the edit distance between the string values in this field is less than or equal to 2. However, a problem of using table functions for implementing query operators are the strong typing restrictions: for the table functions a return type always has to be specified that prevents to use the same function for different input relations.

As one possible solution table functions using and returning structures containing generic tuple identifiers (e.g., Oracle's `rowid`) can be used. So, the SIM-GROUPING function produces a tuple of tuple identifier / group identifier pairs, where the group identifier is an artificial identifier generated by the operator. Based on this, the result type `gid_tid_table` of the table function is defined as follows:

```
create type gid_tid_t as object
  gid (int, tid int);
create type gid_tid_table
  is table of gid_tid_t;
```

Using a grouping function `sim_grouping` a query can be written as the following query:

```
select ...
from table(sim_grouping (
       cursor (select rowid, * from raw_data),
       'edist(title) < 2'))) as gt,
     raw_data
where raw_data.tid = gt.tid
group by gt.gid
```

This query groups tuples having an edit distance of less than 2 either directly or indirectly. A discussion of according aggregate functions is given in Section 5.5.

Our approach allows to implement the function in a generic way, i.e., without any assumption on the input relation. In order to apply aggregation or reconciliation to the actual attribute values of the tuples, they are retrieved using a join with the original relation, whereas the grouping is performed based on the artificial group identifiers produced by the grouping operator.

In the same way, the SIMJOIN operator can be implemented as a table function returning pairs of tuple identifiers that fulfil the similarity condition and are used to join with the original data.

5.4 Evaluation

The similarity-based grouping and join operators described in Chapter 4 were implemented according to the principles outlined in Section 5.2 as part of the FRAQL query engine by Sattler et al. (see [SCS00]) and, alternatively, according to Section 5.3 using the extensibility interfaces of the commercial database management system Oracle8i. For evaluation purposes the latter implementation was used. The test environment was a PC system with a Pentium III (500 MHz) CPU running Linux and Oracle 8i. The extended operators and predicates were implemented using C++. All test results refer to our implementation of the string similarity predicate based on the edit distance and supported by a trie index. A non-index implementation of the predicate is provided for comparison. Indexes are currently created on the fly and maintained in main memory only during operator processing time. The related performance impact is discussed below.

For the grouping operator test runs separate data sets containing random strings were created according to the grade of similarity to be detected, i.e. for one original tuple between 0 and 3 copies were created that fulfilled the similarity condition of the test query. The test query consisted of an edit distance predicate on only one attribute. Using the edit distance with all operations having a fixed cost of 1 and an edit distance threshold k on an attribute, each duplicate tuple had between 0 and k deletions, insertions, or substitutions. As the number of copies

Id	Data	CopyOf	Edist
1	abhfhfhhflk		
2	huiqwerzhads		
3	hdhhhhrrrr		
…	…	…	…
567	abhffhhflk	1	1
568	ahbfhfhhfk	1	2
569	huiqwerzhads	2	0
570	hdhhhrrrr	3	1
571	hdhhhhrr	3	2
…	…	…	…

Figure 5.2: Example input relation

and the numbers of applied operations on the string attributes were equally distributed, for n original tuples the total size of the data set to be processed was approximately $3*n$ with an average distance of $\frac{k}{2}$ among the tuples to be detected as similar. Furthermore, to check the accuracy of the approach, additional information about the creation of the input set were stored with duplicate tuples. Part of an input relation is shown in Fig. 5.2.

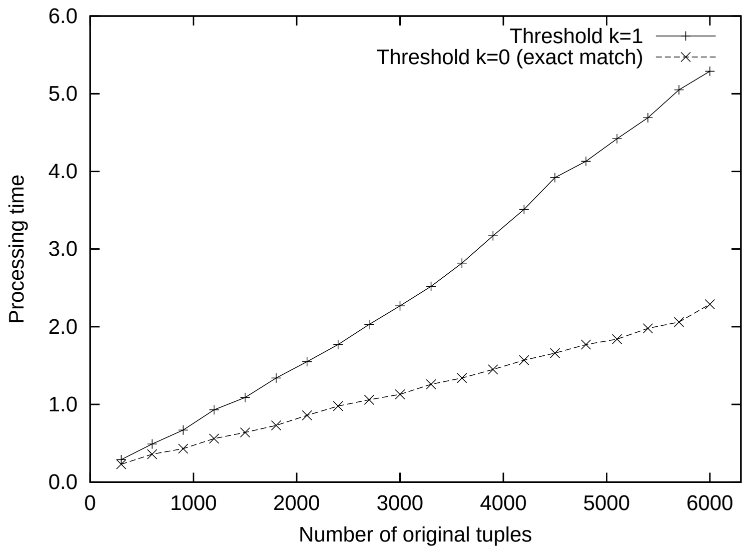


Figure 5.3: Grouping with exact match and threshold $k = 1$

Grouping based on an exact matching ($k = 0$) has the expected complexity of $O(n)$, which results from the necessary iteration over the input set and the trie lookup in each step, which for an exact match requires average word-length comparisons, i.e. can be considered $O(1)$. This conforms to equality based grouping with hash table support. For a growing threshold, the number of comparisons, i.e. the number of trie nodes to be visited, grows. This effect can be seen in Fig. 5.3, where the complexity for $k = 1$ appears to be somewhat worse than linear, but still reasonably efficient.

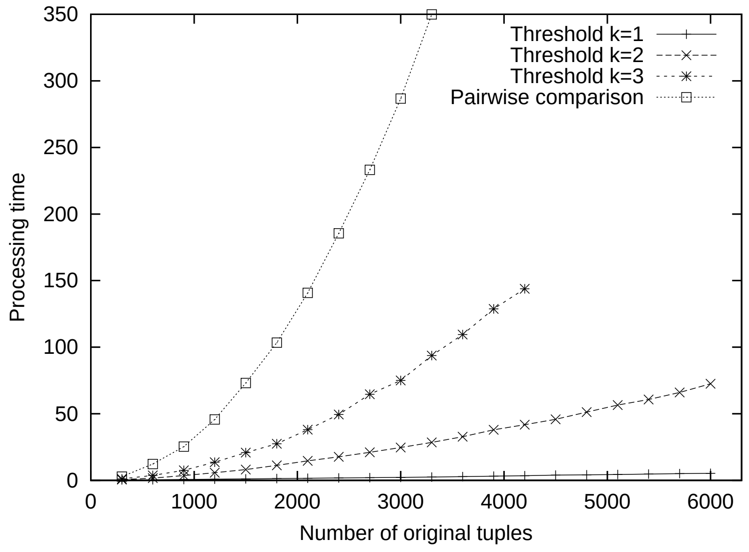


Figure 5.4: Grouping with varying thresholds $k \geq 1$ and the naive approach of pairwise comparisons

Actually, the complexity grows quickly for greater thresholds, as larger regions of the trie have to be covered. The dynamic programming approach of the similarity search ensures that even for the worst case each node is visited only once, which results in equal complexity as pairwise similarity comparison, not considering the cost for index maintenance etc. The currently used main memory implementation of the trie causes a constant overhead per insertion. Hence, the $O(n^2)$ represents the upper bound of the complexity for a growing threshold k, just like $O(n)$ is the lower bound. For growing thresholds the curve moves between these extremes with growing curvature. This is a very basic observation that applies to similarity based operations like similarity-based joins and selections as well, the latter having a complexity between $O(1)$ and $O(n)$. The corresponding test results are shown in Figure 5.4.

The previous test results were presented merely to make a general statement about the efficiency of the similarity-based grouping operator. An interesting question in real life scenarios would be, how the operator performs on varying ratios of duplicates in the tested data set. In Figure 5.5 the dependency between the percentage of duplicates and the required processing time is given for the threshold $k = 2$. While the relative time complexity remains, the absolute processing time decreases for higher percentages of detectable duplicates. Obviously, and just as expected, using a similarity measure is more efficient, if there actually is

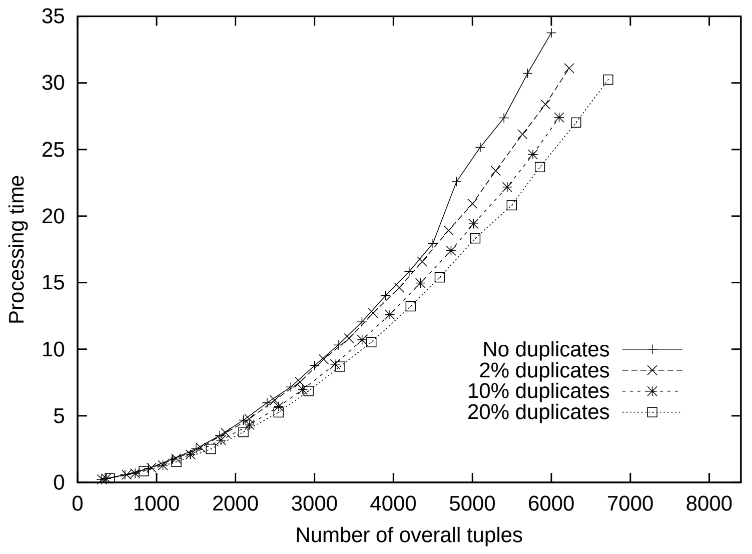


Figure 5.5: Grouping with varying percentage of duplicates in the test data sets

similarity to detect. Otherwise, searching the trie along diverging paths represents an overhead that will not yield any results.

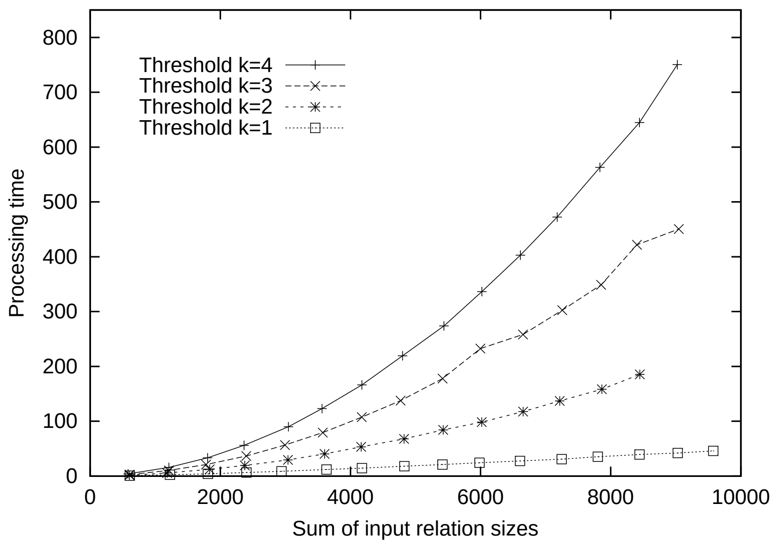


Figure 5.6: Results for varying thresholds $k \geq 1$ for a similarity join

Similar results were received for the described implementation of a similarity join. The test scenario consisted of two relations r_1 and r_2, with a random number of linked tuples, i.e. for each tuple in r_1 there were between 0 and 3 linked records in r_2 and the join attribute values were within a maximum edit distance.

The results are shown in Figure 5.6. As the implementation of the join operation is similar to the grouping operation the complexity is between $O(n)$ and $O(n^2)$ depending on the edit distance threshold.

5.5 Applications

In this section some further aspects of the proposed operations related to their application are discussed. This includes results from a project concerning an Internet database for lost cultural assets, that facilitates the registration of and the search for lost art. Registering new objects in the database can result in "duplicates" if the particular object was already reported by another user or institution but with slightly different descriptions. Furthermore, the data in the database can be enriched by external information, e.g. from artist catalogues. Due to possible different transcriptions for example of artist names, join conditions are necessary. These problems were already used for illustration purposes in Section 1.1 of the thesis.

The problem of duplicates can be solved by applying the similarity-based grouping operations. Using an appropriate similarity predicate (see below for a discussion) potential redundant objects can be identified. In our application a similarity predicate consisting of a combination of artist and title comparisons produces good results. For the artist name special similarity predicates taking different variants of first-name/last-name combinations into account were implemented. So, a typical query is formulated as follows:

```
select ...
from data
group by similarity on sim_person(artist) > 0.9
                       and edist(title) <= 2
```

However, this is only the first step towards "clean" data: From each group of tuples a representative object has to be chosen. This merging or reconciliation step is usually performed in SQL using aggregate functions. But, in the simplest case of the builtin aggregates one is able only to compute minimum, maximum, average etc. from numeric values. As an enhancement modern DBMS provide support for user-defined aggregation functions (UDA) which allow to implement application-specific reconciliation functions. However, these UDAs are still too restricted for reconciliation because they support only one column as parameter. Here, the problem is to choose or compute a merged value from a set of possible discrepant values without looking at any other columns. We can mitigate this problem by allowing more than one parameter or by passing a structured value as parameter to the function.

Therefore, a number of enhanced aggregation functions were developed. In particular for reconciliation purposes, we have defined the following aggregate functions:

- `pick_where_eq (`*`v, col`*`)` returns the value of column *`col`* of the first tuple, where the value of *`v`* is true, i.e., $\neq 0$. In case of a group consisting of only one tuple, the value of this tuple is returned independently of the value of *`v`*.
- `pick_where_min (`*`v, col`*`)` returns the value of column *`col`* of the tuple, where *`v`* is minimal for the entire relation or group, respectively.
- `pick_where_max (`*`v, col`*`)` returns the value of column *`col`* of the tuple, where *`v`* is maximal.
- `to_array (`*`col`*`)` produces an array containing all values from column *`col`*.

With the help of these functions several reconciliation policies can easily be implemented as shown in the following. In a first example, we assume that the final value for column `col` of each group has to be taken from the tuple containing the most current date, which is represented as column `m_date`:

```
select max(m_date), pick_where_max(m_date, col), ...
from data
group by ...
```

In the second example, each tuple contains a column `src` describing the origin in terms of the source of the tuple and this way, realising a source-aware integration view. Assuming a "preferred source" reconciliation strategy, where in case of a conflict the value from source S_P is selected, we could formulate the query as follows:

```
select pick_where_eq(src = 'S_P', col), ...
from data
group by ...
```

Finally, for allowing the user to decide about the resolved value in an interactive way, the `to_array` can be used to collect the list of conflicting values:

```
select to_array(col), ...
from data
group by ...
```

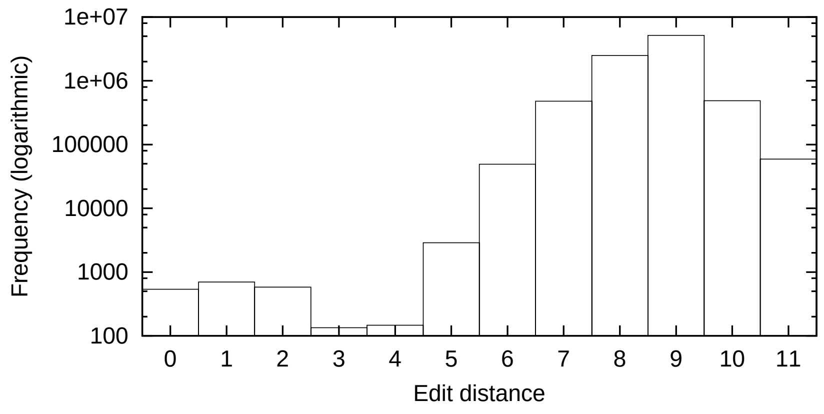


Figure 5.7: Edit distance distribution of random strings in the test data set with 20% duplicates of $k_{max} = 2$

As a summary, user-defined aggregation functions provide a viable way to implement specific reconciliation strategies, especially with the extension described above. Combined with powerful grouping operators they make it possible to support advanced cleaning tasks.

Another application-specific question raised within the previously mentioned project is, how to specify the similarity predicate for similarity joins or grouping consisting of the similarity or distance measure itself and the threshold. If the chosen threshold has such a major impact on the efficiency of similarity-based operations, as described in Section 5.4, the question is how to specify a threshold to meet requirements regarding efficiency and accuracy. Actually, this adds complexity to the well studied problem of over- and under-identification, i.e. falsely qualified duplicates. Information about the distance or similarity distribution can be used for deciding about a meaningful threshold, as well as for refining user-defined similarity predicates. Distance distributions usually conform to some natural distribution, according to the specific application, data types, and semantics. Inconsistencies, such as duplicates, cause anomalies in the distribution, e.g. local minima or points of extreme curvature. Figure 5.7 depicts the edit distance distribution for one of the sample sets from Section 5.4 of 4000 tuples having approximately 20% duplicates with an equal distribution of 0, 1, or 2 edit operations to some original tuple, which is apparent in the chart. To actually choose a threshold based on such a distribution, aspects of efficiency as well as quality of the duplicate detection process have to be considered. Hence, setting $k = 2$ could be a reasonable result drawn from this chart alone.

While the previous anomaly in Figure 5.7 was created intentionally, similar effects result from the integration of overlapping data sets in real applications.

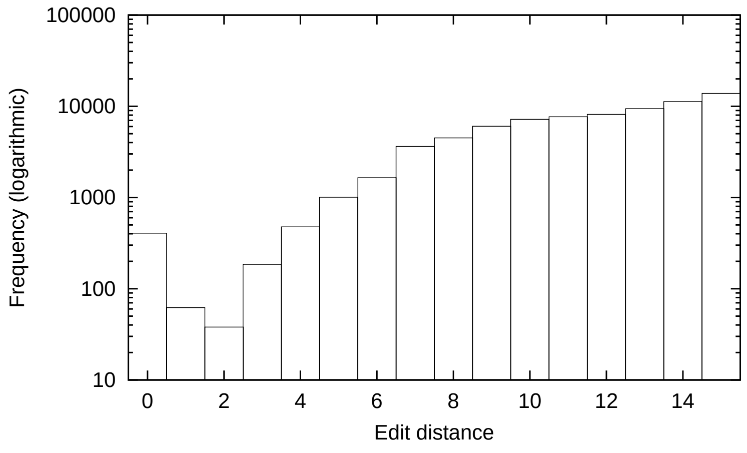


Figure 5.8: Edit distance distribution in an integrated and sampled data set on cultural assets

Figure 5.8 shows a result for a sample consisting of approximately 1.600 titles starting with an ”E” from integrated sources of data on cultural assets. Neverthe-less, drawing the same conclusion of setting the edit distance threshold to receive a useful similarity predicate would lead to a great number of falsely identified tu-ples. For short titles there would be too many matches, and longer titles often do not match this way, because the length increases the number of typos etc.

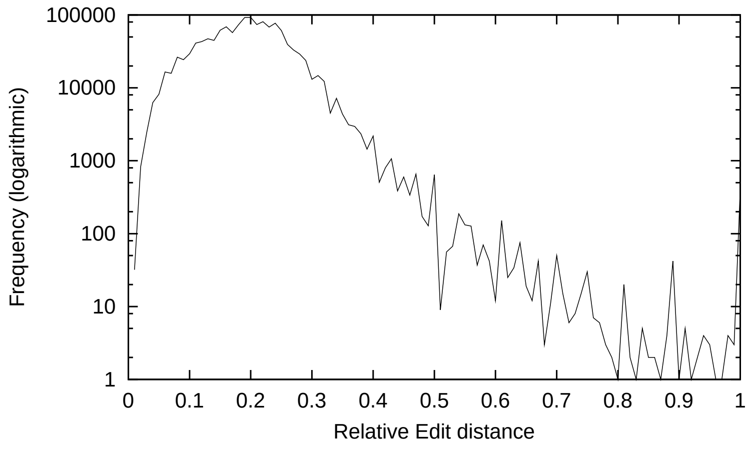


Figure 5.9: Relative edit distance distribution

Better results can be achieved by applying a relative edit distance $rdist(x,y) = 1 - \frac{edist(x,y)}{max(x.length,y.length})$, which is a normalised similarity measure as introduced in section 3.2. The algorithm introduced in section 5.2 can easily be adjusted to this relative distance. Figure 5.9 shows the distribution of the relative edit distances in the previously mentioned example relation. Using the first global minimum around 0.8 as a threshold, and analysing matches in this area shows that it provides a good ratio of very few over- and under-identified tuples.

A successive adjustment of similarity predicates using information from analytical data processing is also of interest for the creation of user-defined similarity predicates. For instance, directly using the edit distance on author names and their various representations will yield poor results. Combining analytical processing and a stepwise addition of canonising techniques like transformation of lower or upper case letters, tokenising, abbreviation matching, etc., as mentioned in Section 3.3 quickly leads to more meaningful distributions, that can be used to derive a threshold value.

5.6 Conclusions

In this Chapter similarity-based operations for finding related data and identifying duplicates based on similarity-based criteria suitable in materialised data integration scenarios were presented. Intended as an extended grouping operation and by combining it with aggregation functions for merging/reconciling groups of conflicting values the proposed grouping operator can be integrated into the relational algebra framework and the SQL query processing model. In a similar way, the extended join operator takes similarity predicates into consideration. These operations can be utilised in ad-hoc queries as part of more complex data integration and cleaning tasks. Moreover, a way to use these operations within existing DBMS using extensibility interface was shown by providing an implementation using the commercial DBMS Oracle8i.

Furthermore, it was shown that efficient implementations have to deal with specific index support depending on the applied similarity measure. For one of the most useful measures for string similarity we have presented a trie-based implementation. The evaluation results illustrate the benefit of this approach even for relatively large datasets. Though the focused in this thesis primarily is on the edit distance measure, the algorithms for similarity-based grouping and join as introduced in this Chapter are able to exploit any kind of index support.

Focusing on just one similarity predicate and keeping the strong context-dependency of similarity measures in mind, the most important question left unanswered probably is, how to find, specify, and efficiently support appropriate similarity predicates for a range of applications. In many cases, basic similarity measures like the edit distance are probably not sufficient. As described in Chapter 3, application-specific similarity measures implementing domain heuristics (e.g. permutation of first name and last name) based on basic edit distances is often a viable approach.

However, choosing the right thresholds and combinations of predicates during the design phase of an integrated system often requires several trial-and-error cycles. This process can be supported by analytical processing steps as shown in Section 5.5 and according tools.

Chapter 6

Re-writing Similarity-based Queries for Virtual Data Integration

While in Chapter 4 the foundations of similarity-based operations where introduced, and in Chapter 5 the implementation of such operations for temporarily or persistently materialised result sets was covered, this chapter addresses problems of a distributed processing of similarity-based operations in heterogeneous environments. For this purpose special concepts to handle similarity predicates probably not supported by integrated systems have to be applied. Again, the description is focused on string similarity measures and on re-writing queries containing according predicates in a way, that source systems can answer such queries.

Provided implementations include similarity-based selections and joins, but not the previously described similarity-based grouping. This is because the operation is hardly applicable across various sources when there are no further constraints on the input set. If the latter is the case, the source selections representing the constraints are processed first as introduced in this section, and then grouping and aggregation can take place as described in the previous chapter.

6.1 Introduction

To address the problem of data level conflicts in weakly related or overlapping data sets from different sources, similarity-based operations were integrated in data integration research. Unfortunately, the support for such operations in current data management solutions is rather limited. And worse, interfaces provided over the Web are even more limited and almost always do not allow any similarity-based lookup of information. The specification of query capabilities is addressed for instance by Vassalos et al. in [VP97] and by the author of this thesis and Endig in [SE00]. The majority of attributes used for querying are string attributes, but while string similarity can be expressed using for instance the Levenshtein distance, common interfaces only include the lookup based on equality or substring and keyword containment. While such predicates do not allow to perform similarity selections or joins directly, they can be used for efficiently finding candidate sets as described in this Chapter.

The principal idea of the presented approach is to provide a pre-selection for string similarity operations by using string containment operations as provided by all databases and most information systems. Regarding the pre-selection this approach is similar to those by Gravano et al. introduced in [GIJ$^+$01] and extended in [GIKS03]. Contrary to their pre-selection strategy, the one presented here is not only applicable in a scenario were integrated data sets or data sets in general are materialised in one database, but also allows re-writing string similarity queries for the virtual integration of autonomous sources. This way, it is applicable in Web integration scenarios.

Another related approach only applicable in materialised scenarios is described by Jin et al. in [JLM03], which is based on FastMap introduced by Faloutsos and Lin in [FL95] and shortly described in Section 3.2 of this thesis. Nevertheless, this approach requires the full domain of string values to define a mapping to an n-dimensional space, and according interfaces for efficient lookup.

The pre-selection proposed here is based on the edit or Levenshtein distance as introduced in Sections 3.2.2 and 3.3 of this thesis, which expresses the dissimilarity of two strings by the minimal number k of operations necessary to transform a string to a comparison string. A basic observation described for instance by Navarro and Baeza-Yates in [NBY98] is, that if we pick any $k+1$ non-overlapping substrings of one string, at least one of them must be fully contained in the comparison string. This corresponds to *Count Filtering* as introduced by Gravano, where the number of common q-grams (substrings of fixed length q) in two strings is used as a criterion. So, searching a large pool of string data we may find a candidate set by selecting all strings containing at least one of these $k+1$ chosen substrings. Based on this observation, Navarro and Baeza-Yates in their approach use q-gram indexes for approximate searches within texts in an Information retrieval context.

The problem with selecting according substrings for pre-selection is, we cannot use Length Filtering and Position Filtering like described in [GIJ$^+$01] to further refine the pre-selection, because we cannot access the necessary information in a non-materialised scenario. And, if we choose inappropriate substrings, the candidate sets can be huge. In this case, the question is: which substrings are appropriate? Obviously, we can minimise the size of the intermediate result by finding the $k+1$ non-overlapping substrings having the best selectivity when combined in one disjunctive query. Then, processing a string similarity predicate requires the following steps:

1. Transform the similarity predicate to an optimal disjunctive substring pre-selection query considering selectivity information

2. Process the pre-selection using standard functionality of the information system yielding a candidate set

3. Process the actual similarity predicate within a mediator or implemented as a user defined function in standard DBMS

While this sketches only a simple selection, we will describe later on, how for instance similarity joins over diverse sources can be executed based on bind joins as described by Roth and Schwarz in [RS97]. Furthermore, we will discuss the advantages and disadvantages of the kind of substring used, whether it is arbitrary substrings, q-samples as fixed length substrings, or tokens.

We have to point out that though substring queries can easily be optimised, many systems including well-known relational DBMS fail to do this. Hence, step 2 in the above mentioned processing may or may not be efficiently executed by integrated source systems. Nevertheless, in virtual integration the key aspect very often is to minimise the size of intermediate results that have to be transferred from a source to the mediator. But most of all, in such scenarios we cannot expect the source systems to provide any interface for similarity searches.

6.2 Mapping Similarity predicates

We consider a predicate like $edist(x,y) \leq k$ as part of a similarity condition, where x and y represent attribute names, or where one may represent a literal search string. First we have to make clear, what kind of edit distance definition we use. The simple definition as outlined in 3.3 includes only insertion, deletion, and substitution. In this case, for a threshold k the number of required non-overlapping substrings is $n = k + 1$, because all of the above mentioned operations can only modify one substring each, i.e. after k operations there is a maximum of k modified substrings. A commonly used derivative of the edit distance in addition allows transpositions of characters, i.e. 'abc' and 'acb' would have an edit distance of only 1 compared to 2 using the simple definition. Considering transpositions increases the number of sub-strings to be considered to $n = 2k + 1$, because every transposition can modify two substrings if they are adjacent in the original string. In the remainder of this paper we consider only the classical definition.

Considering what kind of substring is most suitable, let us assume a predicate

$$edist('Vincent\ van\ Gogh', stringAttribute) \leq 1.$$

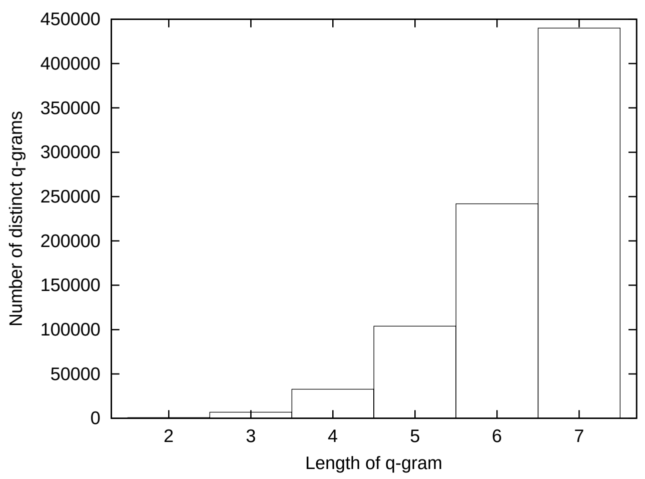


Figure 6.1: Number of distinct q-grams for varying q in the King James Bible

Such a predicate with a fixed search string $s =' Vincent\ van\ Gogh'$ can either be a direct similarity selection or the bound predicate during a bind-join. Assuming we have selectivity information $sel(a)$ about any substring $a = s[i, j]$, $0 \leq i < j < length(s)$ of $s \in \Sigma^*$ over an alphabet Σ available as discussed later in Section 6.3, we may choose the following substrings for pre-selection predicates:

- **Arbitrary Substrings:** 'Vincent van' $\vee$ ' Gogh'
- **Fixed length substrings (q-samples):** 'Vinc' $\vee$ 'Gogh' (here $q = 4$)
- **Tokens:** 'Vincent' $\vee$ 'Gogh'

All three obviously must yield a candidate set including the correct result, but they differ largely regarding their selectivity. Intuitively, longer strings have a better selectivity, because every additional character refines the query. This consideration would render the transformation to q-samples as the least effective one. On the other hand, there is an overhead for managing and using selectivity information. Storing such information for arbitrary strings requires complex data structures to be efficient and considerable memory resources. In general, choosing a suitable substring paradigm implies a trade-off between several aspects.

Selectivity: as mentioned above, the selectivity of longer substrings is always better than or, in the unlikely worst case, equal to a shorter substring, $sel(s[i, j]) \geq sel(s[k, l]), 0 \leq k \leq i \leq j \leq l < length(s)$. Choosing a small q as for instance 3 or 4 will likely return more intermediate results and this way introduce a high overhead for transfer and local processing.

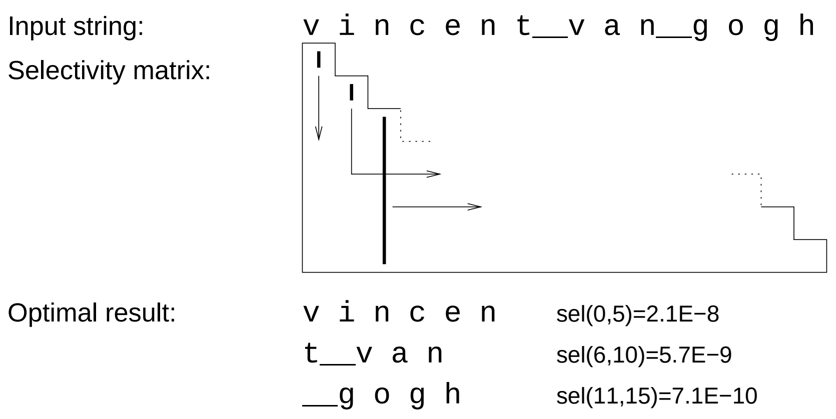


Figure 6.2: Finding selective substrings for $k = 2$, hence $n = k + 1 = 3$

Maintenance: independently of what data structure we use for maintaining selectivity information, the required data volume grows dramatically with the (possible) length of the substrings due to a combinatoric effect for each additional position. For q-grams this effect is shown for varying q based on data from the King James Bible in Figure 6.1. Hence, a greater q increases the necessary overhead for global processing and the global resource consumption.

Applicability: we run into problems if a comparison string is not long enough to derive the necessary number of substrings such as tokens or q-samples. For instance, if the allowed edit distance is $k = 3$ and $q = 5$ a disjunctive pre-selection must contain $n = k+1 = 4$ q-samples of length 5, i.e. the minimal required length of the mapped search string is $l_{min} = n * q = 20$. Obviously, it is not possible to derive the necessary 5-samples from the string 'Vincent van Gogh'. We will discuss later on, what can be done if this problem occurs.

Source capabilities: we consider two kinds of sources regarding the query capabilities, those allowing substring and those allowing keyword searches. For the latter, only tokens are suitable for composing pre-selection queries.

6.2.1 Substring Decomposition

The optimal solution to the addressed problem regarding selectivity performs the mapping in terms of a complete decomposition of the search string s into $n = k+1$ non-overlapping substrings. The decomposition consists of positions $pos[0] \dots pos[n]$ with $pos[0] = 0$ and $pos[n] = length(s)$ such that the concatenation $s = s[pos[0], pos[1]-1]s[pos[1], pos[2]-1] \dots s[pos[n-1], pos[n]-1]$ of the substrings is equal to the search string. An optimal decomposition would yield a selectivity $\min 1 - \Pi_{i=0}^{n-1}(1 - sel(s[pos[i], pos[i+1]-1]))$. Here we assume independence between the selected query strings. We will show in the evaluation in Section 6.5 that this actually yields a reasonable estimation.

The algorithm sketched in Figure 6.2 uses a lower triangular matrix A where a_{ij} represents the selectivity of substring $s[i, j]$, hence, $0 \leq i \leq j < length(s)$. If a count suffix trie is used for storing selectivity information, as shown in Section 6.3, this matrix can be generated from $length(s)$ path traversals in the trie. An exhaustive search is quite expensive for long strings, but it can be tuned by skipping high selectivities in the upper region of the triangular matrix. Furthermore, starting with a decomposition of equal length substrings and stepwise adjusting this decomposition by moving adjacent cut positions represents a greedy approach yielding sufficient results regarding the selectivity quickly.

The disadvantage here is that we need selectivity information on the variable length substrings $s[pos[i], pos[i+1]-1]$. Possible solutions and problems for the storage and retrieval of this information is outlined in Section 6.3, but obviously it requires much more resources than managing the same information for q-samples as introduced in the following.

6.2.2 q-samples

The main advantage of using q-samples, i.e. non-overlapping q-grams of fixed length q, for mapping an edit distance predicate to a disjunctive source query results from the straightforward maintenance of according selectivity information, as shown later on in Section 6.3.

To find the best possible combination of n q-samples from a single string s with $length(s) \geq n * q$ an algorithm basically works as shown in Figure 6.3. In a first step selectivity information for all contained q-grams is retrieved from data structures described in Section 6.3 and represented in an array $sel[i] = sel(s[i, i+q]), 0 \leq i < length(s) - q$. As shown later on, this can be accomplished in $O(length(s))$ time. Among the number of all possible combinations we have to find the positions $pos[i], 0 \leq i < n$ with $\forall j, k : 0 \leq j < k < n \wedge pos[k] - pos[j] > q$ that optimises the selectivity of the disjunctive source query, i.e. yields $\min 1 - \Pi_{i=1}^{n}(1 - sel[pos[i]])$.

This selectivity estimation can further be used to decide, if the pre-selection actually should be performed on the data source. If the selectivity exceeds some selectivity threshold and cannot be performed efficiently, i.e. it yields too many intermediate results, the query can be rejected. As the number of possible combinations is $\Pi_{i=1}^{n}(length(s) - (n * q))$ an exhaustive search can become very expensive, especially if the mapping has to be applied during a bind-join on a great number of long strings as shown in Section 6.4. Alternatively, a greedy algorithm

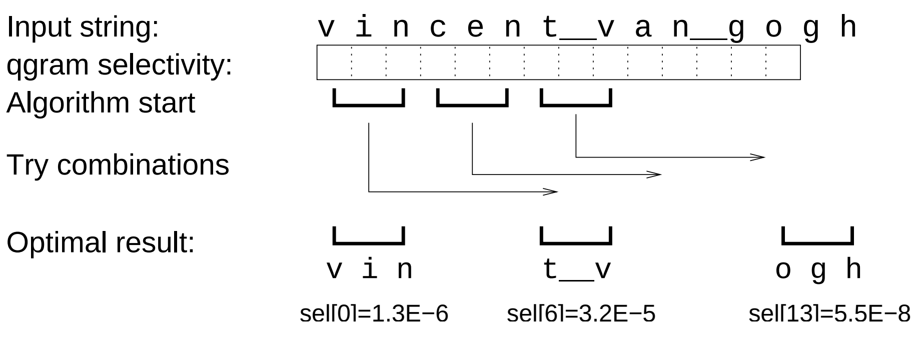


Figure 6.3: Finding selective 3-samples for $k = 2$, hence $n = k + 1 = 3$

with $O(length(s))$ was implemented yielding sufficiently selective combinations, in most cases equal to the result of the exhaustive search.

The selectivity of the resulting pre-selection

$$\sigma_{\bigvee_{i=1}^{n} substring(s[pos[i],pos[i]+q],stringAttribute)}$$

can further be improved by not only considering the retrieved q-samples at $pos[i]$, but also the bounding substring, resulting in a complete decomposition of s. In the given example this may be '**vin**cen' and '**t_v**an_g' and '**ogh**', which can easily be derived. Though we cannot estimate the selectivity of this query based on the given information, unless we move to the approach presented in the previous subsection, it must be better or at least equal to our estimation made based on q-gram selectivity. Another refinement of the presented approach would be to dynamically determine q based on the string length and the number of required q-samples, e.g. $q := \lfloor length(s)/n \rfloor$. This would solve the problem of applicability for shorter strings mentioned above, and improve the selectivity of the pre-selection for longer strings. The disadvantage is that we would need selectivity information for various length q-grams.

Finally, if q is fixed and the applicability condition $length(s) \geq n * q$ does not hold, we may decide to nevertheless send a disjunctive query to the source, containing $m = \lfloor length(s)/q \rfloor < n$ substrings. Though this may not yield all results to the query, it still yields the subset containing $k - (n - m)$ differences in the string representations. Of course, the source query should only be executed, if the estimated selectivity $1 - \Pi_{i=1}^{m}(1 - sel[pos[i]])$ is below a threshold granting efficient processing and transfer of the pre-selection.

6.2.3 Tokens

Considering only substrings of a fixed or variable length would neglect the query capabilities of a great number of sources providing keyword search instead of substring search. To support such interfaces we can choose a set of tokens $T = \{t\}$

derived from our search string s using common delimiters like spaces, commas, etc. Managing and retrieving selectivity information for keywords can be based on standard approaches from information retrieval like the $TF * IDF$ norm. Therefore, it is quite straightforward as outlined in Section 6.3. Finding an optimal combination is also easier than with q-samples or substrings.

The disadvantages of the approach are the in general worse selectivity of keywords compared to the other approaches, a relatively big space overhead for managing selectivity information compared to q-grams, and problems with the applicability. The latter results from the fact that $k+1$ tokens have to be derived, which often may not be possible, e.g. it is impossible to derive a pre-selection for a query like like

$$\sigma_{edist('ErnestHemingway',authorName)\leq 2}$$

because the threshold $k = 2$ implies the need of $n = 3$ tokens, which are not available. The selectivity problems occur because we cannot take advantage of longer substrings, we cannot take advantage of token-spanning substrings, and a probability growing with n of having one or more relatively un-selective keywords in our pre-selection.

6.3 Managing Selectivity Information

In the previous section we described the mapping of similarity-based predicates to substring and keyword queries. These mappings are based on the estimation of the selectivities of arbitrary substrings, q-samples, and tokens. Choosing the most selective pre-selection query requires the storage of selectivity information about these kinds of substrings. In this section we shortly review and adapt data structures to store these information and algorithms to extract the selectivity information. Furthermore, we describe a method to obtain and maintain the required information from uncooperative sources. The work presented in this section is the result of co-operative research with Ingolf Geist.

An overview of data structures to store information for approximate string matching is for instance given by Navarro in [NBYST01]. For the purpose of matching, these structures hold pointers to the occurrences of substrings in text documents, which is the common approach in Information retrieval. Contrary to such approaches, for an estimation of string or substring selectivity as required in this approach the number of occurrences is interesting, and not the positions themselves. Therefore, the data structures were adapted to hold count or frequency information.

According to the description in the previous section, possible data structures for the different substring types are

- full count-suffix trees (FST) or pruned count-suffix trees (PST),
- count tries (CT) or pruned count tries (PCT), that store count information of tokens or q-grams of variable length q, and
- hash tables or pruned hash tables which store the q-grams or tokens and their corresponding counts.

These data structures and their potential usage are described in the following.

A suffix tree is a trie that stores not only all strings s of a database, but also all suffixes $s[i, length(s)-1], 0 \leq i < length(s)$ of s. The *count-suffix tree* is a variant of a suffix tree which does not store pointers to the occurrences of the substrings $s[i,j]$ but maintains the count $C_{s[i,j]}$ of substrings $s[i,j]$. The count assigned to the root node N is the number of all suffixes in the database and can be used to derive a relative frequency value.

The space requirements of a full count-suffix tree can be prohibitive for estimating substring selectivity. Therefore, the *pruned count-suffix tree* was presented by Krishnan et al. in [KVI96]. This data structure maintains only the counts of substrings that satisfy a certain pruning rule, e.g.

- maintain only the top-n levels of the suffix-tree, i.e. retain only substrings with a length $length(a) \leq l_{max}$, or
- retain all nodes that have a count $C_a > p_{min}$, where p_{min} is the pruning threshold.

Count-suffix trees and their pruned version can be used to store selectivity information for arbitrary substrings, as described in the previous section.

To support the storage of selectivity information for q-samples simple and efficient *hash tables* can be applied with in general smaller space requirements. These hash-tables contain q-grams extracted from the strings in the database. Each entry in a hash table $\mathcal{H}_q$ consists of a q-gram as the key and the assigned count information C_{qgram}. In order to reduce the storage costs, the hash table can also be pruned using count-based pruning rules, e.g. maintain only those q-gram entries with a count greater than a given threshold, i.e. $C_{qgram} > p_{min}$.

To support q-samples of varying length selectivity information of q-grams with different q_i has to be maintained. A simple solution can use several hash tables for different length q_i, but this approach causes a considerable redundancy. Alternatively, count-suffix trees pruned on the string length can be applied. Hash tables can also be used for tokens, but tokens are usually of different lengths and non-overlapping. Therefore, the resulting hash table would be similar to an inverted list as commonly used in Information retrieval, with the one difference, that, again, counts instead of pointers are stored.

As mentioned in [JKNS00] a count-suffix tree can be pruned by different rules other than minimum counts. In order to find a compressed representation of q-grams of different lengths by the maximum height of the count-suffix tree, a pruning rule $p \leq q$ means: for each suffix s_i of a string s only the part $s_i[0,q]$ is stored in the count-suffix tree. For each suffix, which now represents a q-gram, the count of occurrences is maintained. Furthermore, if only q-grams of a certain minimum length p_{min} should be maintained, the pruning rule can be extended to $p_{min} \leq q \leq p_{max}$. As almost all q-grams are a prefix of (q+i)-grams, the compression rate is very high. Furthermore, pruning based on the counts can be performed as described above.

Based on information stored in these data structures, selectivity information for a substring a can be derived for instance as

$$sel(a) := \frac{C_a}{N}$$

where C_a is the count value found in the data structure, and N is for instance the separately maintained sum of all occurrences of all substrings managed in a hash table. As previously mentioned, N can be maintained in the root nodes of the introduced tree structures. If a substring is not found, because the index may be incomplete as discussed later on or pruned by a count limit p_{min}, its selectivity can be assumed to be $\frac{1}{N}$ or $\frac{p_{min}-1}{N}$, respectively. Furthermore, the lookup of the count information for all the introduced approaches has the worst complexity of $O(length(a))$, i.e. the frequency of a single substring can be computed very efficiently.

For building and maintaining these index structures, two general approaches are considered.

1. If there are one or more co-operative sources providing full access to their managed data, the described structures or an initial version can be build from the according string sets.

2. For uncooperative sources, like Web databases, building and maintaining these structures has to be based on *query-based sampling* as described for instance in [CC01].

The idea behind query-based sampling are the following. At first, the selectivity information can be seen as an ordered "stop word list", i.e. we want to avoid substrings with a bad selectivity. But, substrings occurring with a high frequency are extremely well approximated with query based sampling, as shown in the evaluation in Section 6.5. This way we can avoid big result sets even with a relatively small ratio of sampled tuples.

However, we may want to maintain and improve the initial frequency information continuously. For this purpose, the result sets during query processing can be used. Updating selectivity information continuously may seem problematic for structures which are pruned based on a count or frequency threshold. Each new entry in an already established structure would fall prey to the pruning rule and does not have a chance to reach the threshold. A solution currently developed by Ingolf Geist and not described here is based on an aging algorithm for the count information in the data structures.

6.4 Similarity-based Operations

The selectivity-based mapping of similarity predicates introduced in the previous sections can be used for re-writing and executing similarity queries on sources with limited query capabilities. This way string similarity predicates can be supported in global queries even if the source systems support only primitive predicates such as

- *substring(a, b)*, e.g. in form of SQL's "`a like '%b%'`" predicate or
- *keyword(a, b)* representing an IR-like keyword containment of phrase *b* in string *a*.

In the following we use a generalised form *contains(a, b)* that has to be replaced by the specific predicate supported by the source system.

Based on the descriptions given in Chapter 4, we focus on two operations:

- the similarity-based selection returning tuples satisfying a string similarity predicate, and
- the similarity-based join combining tuples from two relations based on an approximate matching criterion.

In the following we describe strategies for implementing these operators using selectivity-based mapping.

6.4.1 Similarity-based Selection

As introduced in Section 4.3.1, a similarity selection can for instance be an operation returning all tuples satisfying a similarity predicate based on a distance function $dist(s, attr)$ where $attr \in R$ with a distance less than or equal to a given maximum distance k:

$$\tilde{\sigma}_{dist(s,attr)\leq k} r(R) = \{t \mid t \in r(R) \wedge dist(s, t.attr) \leq k\}$$

The variant of such a similarity predicate considered here is based on the edit distance of strings $edist$:

$$\tilde{\sigma}_{edist(s,attr)\leq k} r(R) = \{t \mid t \in r(R) \wedge edist(s, t.attr) \leq k\}$$

Without loss of generality we focus on simple predicates only. Complex predicates, e.g. connected by $\vee$ or $\wedge$ can be handled by applying the following steps to each atomic predicate and taking into account query capabilities of the sources. Furthermore, we assume that source systems do not support such predicates but only the primitive predicate *contains(a, b)* introduced above. Now, the problem is to rewrite a query containing $\tilde{\sigma}_{SIM}$ in the following form:

$$\tilde{\sigma}_{SIM} \rightarrow \tilde{\sigma}_{SIM}(\sigma_{PRESIM}(r(R)))$$

where σ_{PRESIM} is pushed to the source system and $\tilde{\sigma}_{SIM}$ is performed in the mediator.

Assuming *SIM* is an atomic predicate of the form $edist(s, attr) \leq k$ the selection condition *PRESIM* can be derived using the mapping functions *map_qgram*, *map_substring*, *map_token* from Section 6.2 which we consider in the generalised form *map*. This mapping function returns a set $\{q\}$ of q-samples, substrings, or keywords according to the mappings described in Section 6.2. The disjunctive query represented by this set in general contains $k+1$ strings, unless the length of s does not allow to retrieve this number of substrings. In this case, a the next possible smaller set is returned, representing a query returning a partial result as described before. In any case, the estimated selectivity of the represented query must be better than a given selectivity threshold.

Based on this we can derive the expression *PRESIM* from the similarity predicate as follows:

$$PRESIM := \bigvee_{\forall q \in map(s)} contains(q, attr)$$

In case of using the edit distance as similarity predicate we can further optimise the query expression by applying length filtering. This means, we can omit the expensive computation of the edit distance between two strings s_1 and s_1 if $|\,\text{length}(s_1) - \text{length}(s_2)| \geq k$ for a given maximum distance values k. This holds, because in this case the edit distance value is already $\geq k$. Thus, the final query expression is

$$\tilde{\sigma}_{edist(s,attr)<k}(\sigma_{|\,\mathrm{length}(s)-\mathrm{length}(attr)|\geq k}(\sigma_{PRESIM}(r(R)))$$

where the placement of the length filtering selection depends on the query capabilities of the source.

A second optimisation rule deals with complex disjunctively connected similarity conditions of the form $SIM(s_1, attr) \vee SIM(s_2, attr)$. In this case the pre-selection condition can be simplified to

$$\bigvee_{\forall q_1 \in map(s_1)} contains(q_1, attr) \vee \bigvee_{\forall q_2 \in map(s_2)} contains(q_2, attr)$$

A general problem that can occur in this context are query strings exceeding the length limit for query strings given by the source system. This has to be handled by splitting the query condition into two or more parts $PRESIM_1 \ldots PRESIM_n$ and building the union of the partial results afterwards:

$$\tilde{\sigma}_{SIM}(\sigma_{PRESIM_1}(r(R)) \cup \cdots \cup \sigma_{PRESIM_n}(r(R)))$$

Obviously, the above mentioned optimisation of applying length filtering can be used here, too.

6.4.2 Similarity Join

Based on the idea of implementing similarity operations by introducing a pre-selection we can realise similarity join operations, too. A similarity join $r_1(R_1) \bowtie_{SIM} r_2(R_2)$ where the join condition is an approximate string criterion of the form $SIM(R_1.attr_1, R_2.attr_2) > threshold$ or $edist(R_1.attr_1, R_2.attr_2) \leq k$. As in the previous sections we consider in the following only simple edit distance predicates.

A first approach for computing the join is to use a bind join implementation. Here, we assume that one relation is either restricted by a selection criterion or can be scanned completely. Then, the bind join works as shown in Algorithm 4. For each tuple of the outer relation r_1 we take the (string) value of the join attribute $attr_1$ and perform a similarity selection on the inner relation.

This is performed in the same way as described in Section 6.4.1 by

1. mapping the string to a set of *q*-grams,
2. sending the disjunctive selection to the source,
3. post-process the result by applying the similarity predicate, and then
4. combining each tuple of this selection result with the current tuple of the outer relation.

Algorithm 4: *Bind join*

```
**foreach** $t_1 \in r_1(R_1)$ **do**
    $s := t(R_1.attr_1)$
    **foreach** $t_2 \in \tilde{\sigma}_{edist(s,attr_2)}(\sigma_{PRESIM}(r_2(R_2)))$ **do**
        output $t_1 \circ t_2$
    **od**
**od**
```

The roles of the participating relations (inner or outer relation) are determined by taking into account relation cardinalities as well as the query capabilities. If a relation is not restricted using a selection condition and does not support a full table scan it has to be used as inner relation. Otherwise, the smaller relation is chosen as the outer relation in order to reduce the number of source queries.

A significant reduction of the number of the source queries can be achieved by using a semi-join variant. Here, the following principal approach is used.

1. One of the relations is first processed completely.
2. The string values of the join attribute are collected and the *map* function is applied to each of them.
3. The resulting set $\mathcal{S}$ of *q*-grams, tokens, or substrings is used to build a single pre-selection condition.
4. The result of the according query is joined with the tuples from the first relation using the similarity condition.

This is shown in Algorithm 5.

Algorithm 5: *Semi join*

```
$\mathcal{S} := \emptyset$
foreach $t_1 \in r_1(R_1)$ do
    $\mathcal{S} := \mathcal{S} \cup map(t(R_1.attr_1))$
od
$r_{tmp} := \sigma_{\bigvee_{s \in \mathcal{S}} contains(s, attr_2)}(r_2(R_2))$
foreach $t_1 \in r_1(R_1)$ do
    foreach $t_2 \in r_{tmp}$ do
        if $edist(t_1(R_1.attr_1), t_2(R_2.attr_2)) < k$
            output $t_1 \circ t_2$
        fi
    od
od
```

If the pre-selection condition exceeds the query string limit of the source, the pre-selection has to be performed in multiple steps. In the best case, this approach requires only 2 source queries assuming that the first relation is cached in the mediator or 3 source queries otherwise. The worst case depends on the query length limit as well as the number of derived q-grams. However, if the number of queries is greater than $|r_1| + 1$ one can switch always to the bind join implementation.

A further kind of join operation can be used if none of the both input relations are restricted by a selection condition. Assuming that a full fetch / scan is not possible or not allowed, one could use the index containing frequent q-grams / tokens / substrings together with the selectivity for retrieving possibly matching data from both relations. By processing the results (i.e. extracting q-grams) the index can be adjusted and extended and in this way the following retrieval operations can be focused to promising q-grams. Of course, this *discovery* join cannot guarantee a complete result but is helpful in identifying existing approximate matches.

6.5 Evaluation

For evaluation purposes a real-life data set containing detailed information about cultural assets lost or stolen from private persons and museums in Europe during and after the Nazi regime was used. Because the gathered data is often imprecise or erroneous, similarity based operations are important in this application scenario and are already part of the application. This current research targets the integration with similar databases available over the Web.

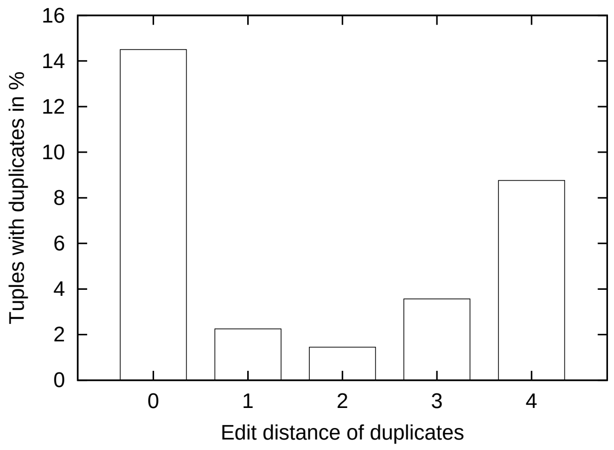


Figure 6.4: Duplicate distribution in test data set

The experiments only dealt with a collection of approximately 60000 titles of cultural assets. The data set contains a great number of duplicates with identical and similar values as shown in Figure 6.4, i.e. for about 14% of the tuples there are tuples with an identical title, for 2% of the tuples there are (possible) duplicates with an edit distance of 1, etc. The distribution conforms to the aspects discussed in Section 5.5.

To evaluate the key criteria described in the following, this data set as well as necessary index structures were materialised in one local Oracle 9i database and queries were mapped to SQL substring queries for pre-selection. The required mapping and further evaluation was implemented in a mediator on top of the database system using Java. The considered queries were similarity self-joins on this one table.

The key criteria considered during evaluation are the selectivity of generated pre-selections, the quality of our selectivity estimation, and the applicability to actual data values. Furthermore, dealing with structures for maintaining selectivity information, the necessary space overhead and impact on the quality of our selectivity estimations for complete, incomplete, and pruned data structures were considered.

Figure 6.5 shows the average selectivity achieved with the proposed q-samples approach for a varying maximum edit distance k and varying q. The size of the candidate sets retrieved from the database were reasonable, especially for $q = 4$ and $q = 5$, 100 to 300 of the approximate 60000 original titles. For a growing edit distance threshold k the selectivity grew linear or better due to the growing number of required disjuncts.

To answer the question, how many queries provide a good selectivity, beneath

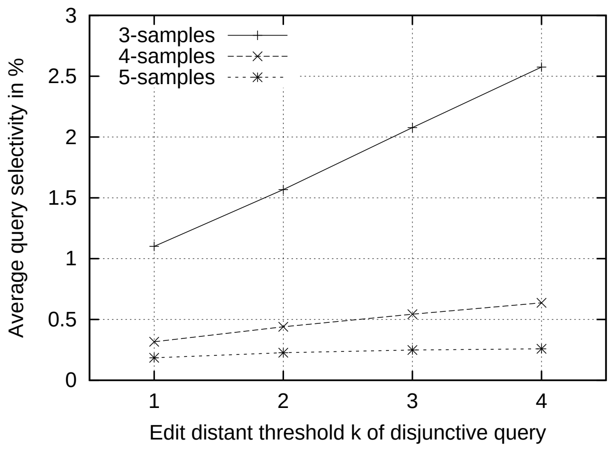
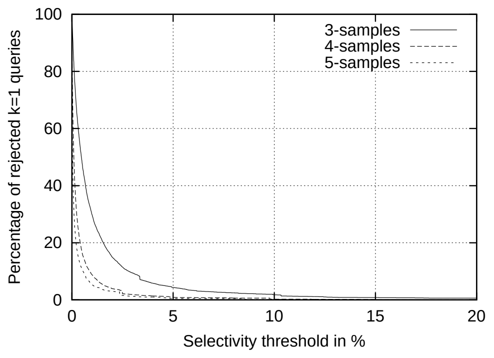


Figure 6.5: Average selectivity for varying q and k

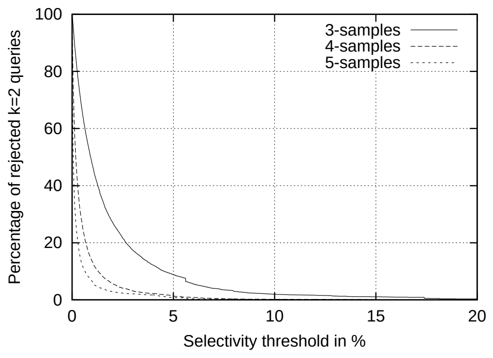
a given threshold, which also can be used to reject a query if the intermediate result would exceed a reasonable limit, the selectivity distribution of queries created from every tuple in the data set was investigated. The results are shown in Figure 6.6 for varying q and k. For example, in Figure 6.6(c) where $k = 3$, if we set the selectivity threshold to 5%, we have to reject approximately 3% of the queries using 4-samples and 5-samples and approximately 14% of queries using 3-samples.

Though the former observation may seem quite bad, actually the edit distance threshold of $k = 3$ is not realistic for most applications, where real duplicates often have a distance of 1 or 2. The effect improves for smaller k as shown in Figure 6.6(d), where for the the same selectivity threshold we see that for $k = 2$ we only have to reject 10% and for $k = 1$ only 5% of our queries. Again, for longer substrings with $q = 4$ and $q = 5$ the queries perform far better, as seen in Figures 6.6(a) and 6.6(b).

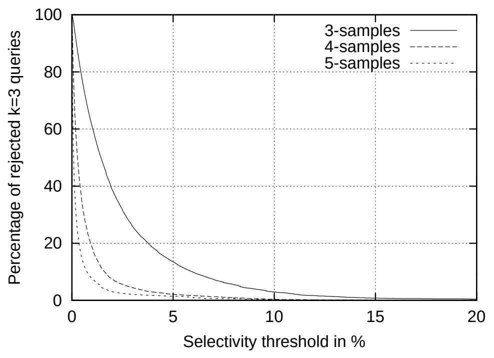
For the previous experiments un-pruned selectivity information stored in hash tables and gathered from the full input relation was used, i.e. for each q-gram the exact number of occurrences was used. Still, our selectivity estimation may be biased by the fact that we assume independence between disjuncts by computing the selectivity as $1 - \Pi_{i=1}^{k+1}(1 - sel(qgram_i))$. Figures 6.7(a) and 6.7(b) show that our estimation is actually quite good for $q = 3$ and $q = 5$ respectively, assuming $k = 2$. The estimation is rather conservative, i.e. we estimate the selectivity somewhat higher than it actually is. Comparing Figure 6.7(a) and Figure 6.7(b) shows that the estimation quality decreases for growing q towards more conservative estimates. Results are only shown for $k = 2, q = 3, q = 5$, but the results for other combinations of q and k were measured and turned out similar.



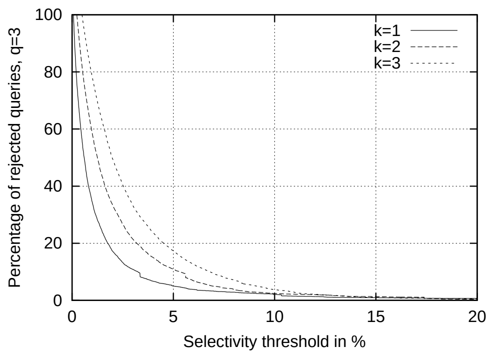
(a) k=1, varying q



(b) k=2, varying q



(c) k=3, varying q



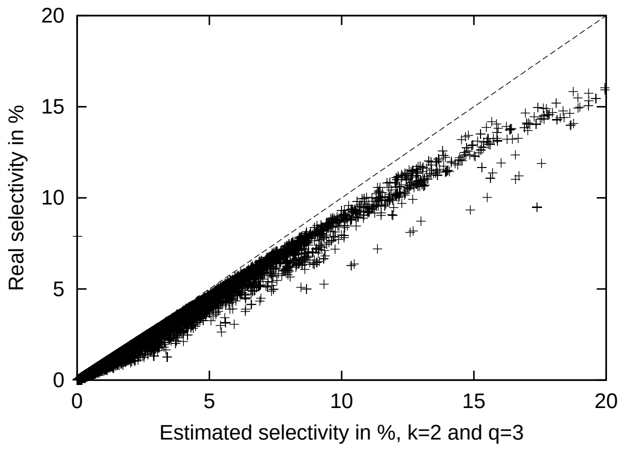
(d) varying k, q=3

Figure 6.6: Cumulative selectivity distribution for varying q and k

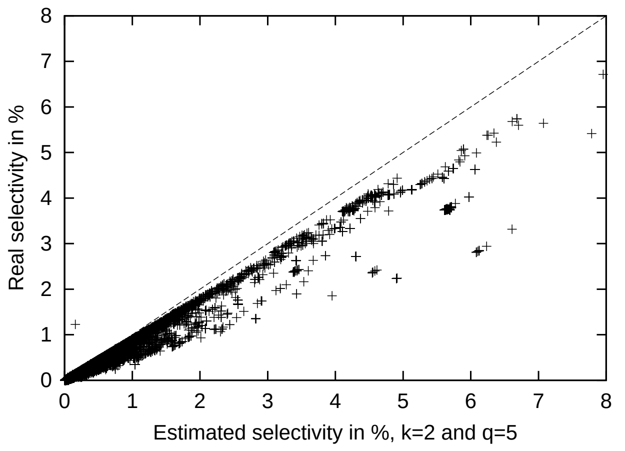
A problem with the presented approach occurs, if the number of required q-samples cannot be retrieved from a given search string, because the latter is not long enough for the decomposition. Figure 6.8 shows how often this was the case with our data set and for varying q and k, i.e. the query strings s did not fulfil the condition $length(s) \leq q * (k+1)$. Though, in this case we can still step back instead of reject and send a query containing less than $k+1$ q-samples providing at least a subset of the result as mentioned before.

Nevertheless, while greater q benefit the selectivity they hinder the applicability when many short query strings exist. Therefore, the parameters q, k, and a possible selectivity threshold have to be chosen carefully based on characteristics of a given application.

In Section 6.3 the usage of query-based sampling was described in order to build selectivity summaries for uncooperative sources. In joint work with Ingolf



(a) q=3



(b) q=5

Figure 6.7: Quality of selectivity estimation for q=3 and q=5

Geist the quality of these information and the influence to the pre-selection predicates was evaluated. First, the differences between full scan estimation and estimation based on a certain sample size are investigated. From all possible q-grams 2000 were selected into a query set Q. Subsequently, we computed the average

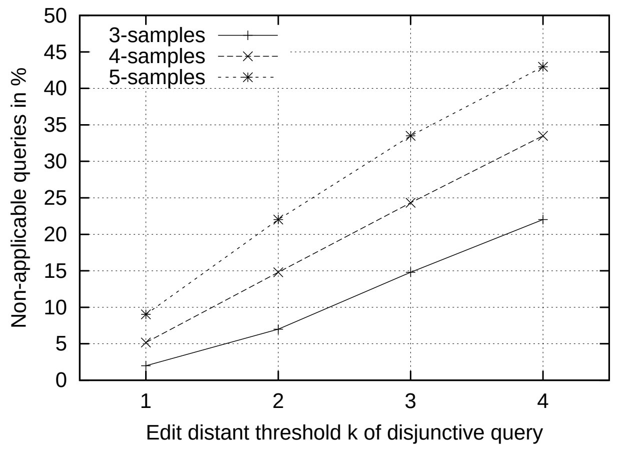


Figure 6.8: Applicability for varying q and k

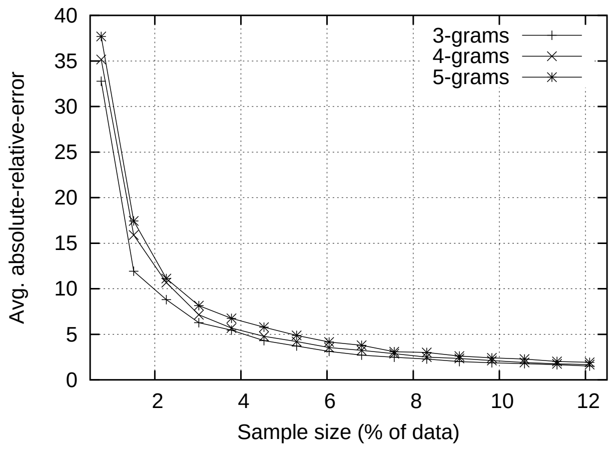


Figure 6.9: Sample size estimation errors

of the absolute-relative-errors defined as

$$e = \frac{1}{|Q|} \sum_{q \in Q} \frac{\text{abs}(\text{sel}(q) - \text{est}(q))}{\text{sel}(q)},$$

with $\text{sel}(q)$ the selectivity q compute based on full statistics, i.e. the real selectivity, and $\text{est}(q)$ the estimated selectivity based on a sample. The results are illustrated in Fig. 6.9.

As assumed the error is decreasing with bigger sample sizes. However, the error is quite high, around 5 with 5% sample size. But, the most important thing is, the relative order of q-grams is retained. Furthermore, high ranked q-grams, i.e. those, which have to be avoided, are approximated well in the sample.

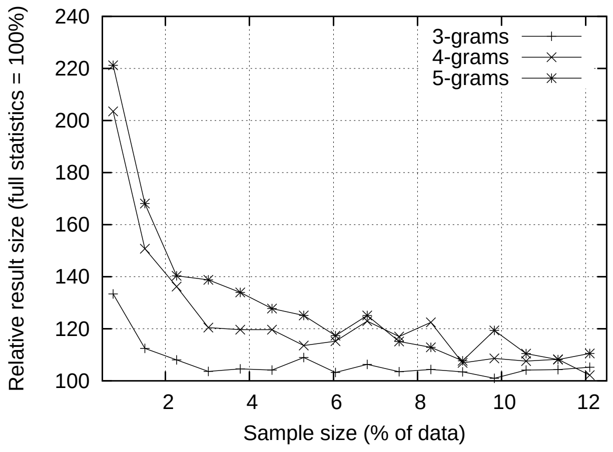


Figure 6.10: Quality of the pre-selection decisions

Following the evaluation of the estimation errors the influence of the errors to the pre-selection results have to be shown. Therefore, we generated a sample set of queries Q_2 which contains 500 strings randomly selected from the database. We measured the average number of tuples returned by the pre-selection condition for an edit distance $k = 2$, i.e. a disjunction of three q-sample substring queries. Here, we assumed the average result size of substring queries created with full statistics as 100%. Fig. 6.10 shows the result sizes of pre-selection queries created using selectivity information from different sample sizes.

Even the precision of query based sampling selectivity estimation is not very high, the query results are close to full statistics. That has several reasons. The selectivity estimation of high ranked q-grams is rather high and ranking similarity is high. Thus, even if the selectivity estimation is not perfect absolutely, the relative order of the q-grams is good.

Finally, we evaluated the influence of the pruning limit on sizes of the storage structures as well as on the quality of pre-selection. The results are illustrated in Figures 6.11 and 6.12 respectively. Especially for 4- and 5-grams pruning reduces the storage costs decisively, e.g. with a pruning limit $p_{min} = 15$ the size of the 5-gram table reduces to 10% of the original size. Nevertheless, the quality of the estimations and result set sizes based on the estimations are very good as seen in Figure 6.12. Because of the higher reduction for 5-grams the results are slightly worse than for 3- and 4-grams. But, the figures show that pruning is well applicable in our scenario.

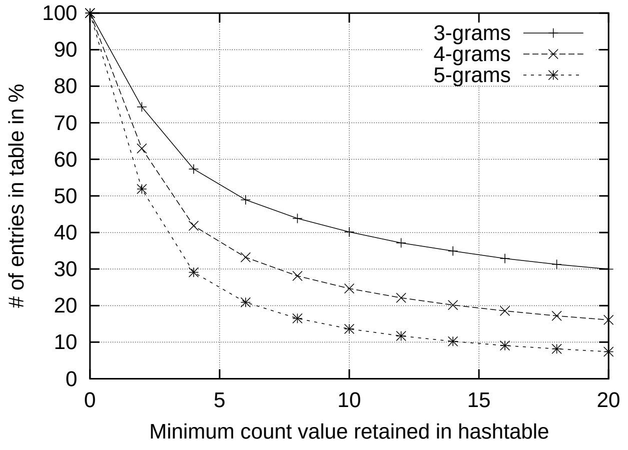


Figure 6.11: *q*-gram table sizes vs. pruning limits

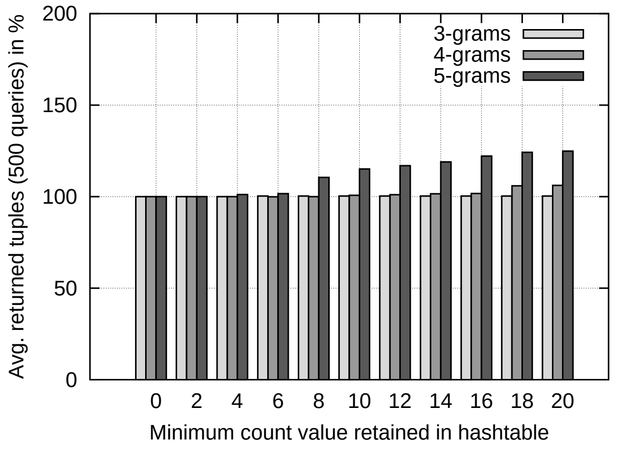


Figure 6.12: Quality of pre-selection vs. pruning limits

6.6 Conclusions

In this chapter an approach for querying based on string similarity in a virtually integrated and heterogeneous environment was presented. String similarity is expressed in terms of the edit distance, and global queries are re-written to

- a source query suitable for standard interfaces to efficiently select a candidate set, and
- a global part of the query that actually computes the correct result within a mediator or FDBMS.

To grant efficiency, queries are re-written to disjunctive source queries based on selectivity information for q-samples, arbitrary substrings, or tokens. The advantages and disadvantages of these three kinds of substrings and the necessary overhead for storing selectivity information was discussed. The latter was aspects and its evaluation is based on joint work with Ingolf Geist.

Based on the predicate mapping implementations for selection and join operations were discussed. For the evaluation of the introduced concepts the focus was on q-samples, because they seemed most suitable regarding applicability in various scenarios and provide a low overhead for query processing and required data structures. The results show that the proposed approach is well appropriate for the target problem, but to grant efficiency and result quality the parameters have to be chosen very carefully according to a given application scenario.

As this approach of string similarity based querying heterogeneous sources in general is quite new, there is of course a great number of open questions, which require further research. This includes a further improvement of source query selectivity. The currently achieved results of fetching only a small fraction of a percent of the original data in most scenarios may be suitable for many applications, but for large data volumes this already may be prohibitive. On the other hand, while a complete decomposition of a search string to substrings is optimal regarding the selectivity, the necessary overhead seems impractical in most applications. Here, mixed approaches and further ways of selectivity estimation are researched.

Using only the string edit distance for similarity operations does not fully reflect real-world requirements, were similarity is most often specific to attribute semantics of the given application, e.g. the similarity of presentations of a persons name can be judged much better using a specific similarity measure. Nevertheless, the general principle of pre-selection by query re-writing remains applicable, as well as many aspects of mapping a given value based on selectivity.

Related to a running project, the discovery join is currently more thoroughly investigated. The intention of this operation is to find related or duplicate information in Web sources, without having either a partial result or one full data set materialised. The current approach is based on the work presented here as shortly sketched in Section 6.4.

Chapter 7

Conclusions

This thesis addressed problems of similarity-based operations in data integration. Data integration has been a topic of research for more than twenty years and has gained a growing interest over recent years because of the continously increasing availability of data from sources in local or global scopes. While data integration in early research has tackled problems mostly on the conceptual level, such as for instance approaches targeting schematic heterogeneities of data from various sources, the research focus shifted toward aspects required to implement solutions for real-life applications, e.g. means for query processing in distributed and heterogeneous environments.

Once schematic heterogeneities are resolved, and techniques of materialised or virtual data integration were applied to provide unified access to an integrated data set, most likely there will still be conflicts on the data level. This is because the information provided by various sources may overlap or relate in some way, and data values representing one real-world object or related objects may vary for instance due to different representation conventions and errors made during input. If this is the case, operations to resolve such conflicts cannot be based on equality of values. Therefore, similarity-based operations are very useful to provide access to integrated data sets.

Furthermore, these operations have to be implemented in a way to efficiently deal with the special requirements in data integration scenarios. On the one hand, the size of an integrated data set may be huge, because the sizes of the source data sets are added. On the other hand, processing queries in a distributed and heterogeneous environment has to be based on the quite narrow bottleneck of current data networks. These two aspects also hold for equality-based operations, but similarity-based operations in addition are characterised by an in general slightly worse performance, because the search space is bigger. Therefore, efficiency of similarity-based operations is a key issue.

7.1 Summary of Contributions

The main contributions to the target research field of this thesis are starting in Chapter 4 with a description of the foundations of similarity-based operations. For this purpose, the semantics of predicate-based operations as known from the relational algebra are adapted to the inclusion of similarity predicates. According consequences and necessary modifications are outlined. In this way, the focus for considerations in the following chapters is constituted.

While the semantics of similarity predicates as well as similarity-based selection and join operations are introduced according to related approaches used in other research, the description of a similarity-based grouping operation is a novel approach, suitable especially for purposes of duplicate detection and reconciliation. Though these two aspects on the surface are well covered by the two stages of grouping and aggregation, for the intended purpose the actual grouping sub-operation required severe changes to deal with specific characteristics of similarity relations. Accordingly, strategies to deal with atransitivity introduced by similarity predicates were described, and a simple strategy of building the transitive closure was used later on.

Though the foundations, algorithms, and implementations provided are described in a way, such that generic similarity predicates and complex conditions can be handled, throughout the Chapters 5 and 6 a string similarity predicate based on the edit distance was used to illustrate possible index support and efficient evaluation during distributed query processing. This specific predicate was chosen because string attributes are most common in data management. On the other hand, efficient support for string similarity is not well researched compared to, for instance, similarity of complex multi-media objects. Hence, the support for such predicates is an urgent requirement, especially in data integration.

Chapter 5 described novel research results consisting of algorithms for the implementation of

- similarity-based join and
- similarity-based grouping

operations in materialised scenarios, i.e. the input relation of the operations exists either in secondary or primary storage of the integration system. In this case, index support for similarity predicates is conceivable, which was illustrated by using a trie index for evaluating string similarity predicates. Implementations were provided for a mediator query engine and as an extension for a commercial database management system. Based on the implementation of the algorithms and the string similarity predicate, the performance of the approach was evaluated. Further aspects of their application were discussed, including the design of similarity predicates and the usage of aggregation functions for reconciliation.

In Chapter 6 the according problems were addressed for virtual integration scenarios.

- Similarity-based selection and
- similarity-based join

operations based on string similarity were introduced suitable for distributed query processing in heterogeneous environments, i.e. the predicate has to be evaluated by source systems with possibly limited query capabilities. To deal with this problem, a new approach was presented, which is based on expanding predicates by deriving a disjunctive set of pre-selection predicates that can be evaluated by most kinds of sources. To grant the efficiency of this mapping, selectivity information on substrings was used. Finally, the mappings and algorithms were evaluated for various aspects of the string similarity predicate and the quality of the selectivity information.

The general intention behind the work presented in this thesis was to provide means to deal with data-level conflicts in a way, such that the operations can be implemented and used efficiently in a number of applications. Therefore, the most important consideration during the research was to introduce similarity-based operations that

- can be implemented as part of common data integration solutions like FDBMS, mediators, or Data warehouses,
- can be integrated with existing data management solutions where appropriate, e.g. in commercial DBMS used for instance in Data warehousing, and
- are implemented based on algorithms considering the specific requirements of efficiency resulting from similarity-based data processing.

Accordingly, prototype implementations of the proposed similarity-based operations were provided, and their efficiency was evaluated and discussed. In general, the evaluation results have shown that similarity-based operations can be performed with a reasonable efficiency. Yet, the strong dependence on the context of similarity and unclear properties of resulting similarity relations make efficient implementations of similarity predicates a difficult task.

7.2 Outlook and Open Problems

Specific summaries and important open problems regarding the presented approaches were already discussed in the conclusions of Chapters 5 and 6. Therefore, in this section an outlook from a broader perspective is given and more general problems in the research fields of interest are addressed.

The importance of similarity in computer science and especially in data management and data integration was outlined several times throughout this thesis. It is a valuable concept for identification and abstraction, which can be applied wherever great amounts of data have to be processed to make it suitable for human comprehension.

Yet, the support for according operations is still in its infancy and the focus is often very limited. To better deal with the requirements of current and future applications, similarity will play a key role. The current lack of similarity-based operations is mostly based on certain properties of similarity and the stark contrast with operations currently used in data management.

While the work presented in this thesis is based on the simple framework outlined in Chapter 4, a more comprehensive view on similarity on a conceptual level is required, explicitly including aspects of suitable data models. Furthermore, an agreement on the characteristics of similarity measures and relations is required as a basis for such a comprehensive framework. Based on this, according operations can be defined in a way, that allows a sound integration with existing or possible future data management and data integration solutions.

The strong dependence on a context of similarity leads to the requirement of different similarity measures that are specific to almost each given application. While this problem can be solved in data management on the low level of extensibility interfaces as for instance provided by current database management systems, the semantics of according user-defined functions are not clear and efficiency as a result is hard to accomplish. Therefore, these aspects have to be covered within the more general framework mentioned above.

While it is reasonable to provide operations like the ones introduced in this thesis in systems that are intended to be used in data integration scenarios, the inclusion in database management systems should be realised as optional extensions based on extensibility interfaces. This is because, the operations are often required in the former systems, but are currently not a key requirement in many standard applications. On the other hand, as shown in Chapter 5 the implementation based on existing extensibility interfaces can be cumbersome and not quite intuitive. Hence, new concepts of extensibility have to be provided by commercial database management systems to better suit current and future applications, including those requiring similarity-based operations.

Bibliography

[AB89] R. Alonso and D. Barbara. Negotiating data access in federated database systems. In *Proc. IEEE Int'l. Conf. on Data Eng.*, page 56, Los Angeles, CA, February 1989.

[AD77] M. Adiba and C. Delobel. The cooperation problem between different data base management systems. In *Architecture and Models in Data Base Management Systems, Nijssen(ed) (IFIP TC-2) Nice France*, 1977.

[AP88] F. G. Ashby and N. A. Perrin. Toward a unified theory of similarity and recognition. *Psychological Review*, 95(1):124–150, January 1988.

[Att50] F. Attneave. Dimensions of similarity. *American Journal of Psychology*, 63:516–556, 1950.

[BKLW99] S. Busse, R.-D. Kutsche, U. Leser, and H. Weber. Federated information systems: Concepts, terminology and architectures. Technical Report Technical report 99-9, Technische Universität Berlin, 1999.

[BKS93] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-Trees. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C.*, volume 22 of *ACM SIGMOD Record*, pages 237–246. ACM Press, June 1993.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, May 1999.

[CC01] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Trans. Inf. Syst.*, 19(2):97–130, 2001.

[CdGL$^+$99] D. Calvanese, G. de Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. A principled approach to data integration and reconciliation in data warehousing. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany*, 1999.

[Coh98] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In L. M. Haas and A. Tiwary, editors, *Proceedings ACM SIGMOD, 1998, Seattle, Washington, USA*, pages 201–212. ACM Press, 1998.

[Con97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, Berlin/Heidelberg, 1997.

[Dat90] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley Publishing Company, Reading , MA , USA, 5th edition, 1990.

[DFGG97] G. Das, R. Fleischer, L. Gasieniec, and D. Gunopulos. Episode matching. *Lecture Notes in Computer Science*, 1264, 1997.

[DS96] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3):339–369, September 1996.

[EN94] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, Redwood City, CA, 2 edition, 1994.

[FL95] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 May 1995.

[Fou97] IBM Community Development Foundation. The net result - report of the national working party for social inclusion., 1997.

[Fuh95] N. Fuhr. Probabilistic datalog – A logic for powerful retrieval methods. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Retrieval Logic, pages 282–290, 1995.

[GBL98] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *DL'98: Proceedings of the 3rd ACM International Conference on Digital Libraries*, pages 89–98, 1998.

[GFSS00] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: an extensible data cleaning tool. In Weidong Chen, Jeffery Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas*, volume 29(2), pages 590–590, 2000.

[GIJ$^+$01] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of the Twenty-seventh International Conference on Very Large Data Bases: Roma, Italy, 11–14th September, 2001*, pages 491–500, Los Altos, CA 94022, USA, 2001. Morgan Kaufmann Publishers.

[GIKS03] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *Proceedings of the twelfth international conference on World Wide Web*, pages 90–101. ACM Press, 2003.

[GL94] P. Gupta and E. Lin. Datajoiner: A practical approach to multidatabase access. In *Parallel and Distributed Information Systems (PDIS '94)*, pages 264–264, Los Alamitos, Ca., USA, September 1994. IEEE Computer Society Press.

[Gol99] R. L. Goldstone. Similarity. In R. A. Wilson and F. Keil, editors, *The MIT Encyclopedia of the Cognitive Sciences*, page 1312. MIT Press, 1999.

[Goo72] N. Goodman. Seven strictures on similarity. In *Problems and Projects*, pages 437–447. Bobbs-Merrill, New York, 1972.

[GPQ$^+$94] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, J. Sagiv, J. D. Ullman, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages (Extended Abstract). Technical Report, Stanford University, 1994.

[Gra93] G. Graefe. Query Evaluation Techniques For Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceeding of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 47–57, Boston, MA, June 1984.

[Ham] W. R. Hamilton. *The Mathematical Papers of Sir William Rowan Hamilton*. University Press, Cambridge. Vol.I Geometrical Optics (1931), Vol.II Dynamics (1940), Vol.III Algebra (1967).

[HD80] P. A. V. Hall and G. R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, 1980.

[HM79] M. Hammer and D. McLeod. On database management system architecture. Technical Report TR-LCS-T, Machine Intelligence, eds: Meltzer, and Michie, vars. PublishersT Laboratory.for CS,, October 1979.

[HS95] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 127–138, San Jose, California, 22–25 May 1995.

[HSC99] J. M. Hellerstein, M. Stonebraker, and R. Caccia. Independent, Open Enterprise Data Integration. *IEEE Data Engineering Bulletin*, 22(1):43–49, 1999.

[Hut81] J. Hutchinson. Fractals and self-similarity. *Indiana University Mathematics Journal*, 30:713–747, 1981.

[Hyl96] J. A. Hylton. Identifying and merging related bibliographic records. Technical Report MIT/LCS/TR-678, Massachusetts Institute of Technology, February 1996.

[Inm96] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 2 edition, 1996.

[Jag91] H. V. Jagadish. A retrieval technique for similar shapes. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(2):208–217, June 1991.

[Jam90] W. James. *The Principles of Psychology*. Holt, New York, 1890.

[JKNS00] H.V. Jagadish, O. Kapitskaia, R.T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal The International Journal on Very Large Data Bases*, 9(3):214 – 230, dec 2000.

[JLM03] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), March 26-28, 2003, Kyoto, Japan*. IEEE Computer Society, 2003.

[JTU96] P. Jokinen, J. Tarhio, and E. Ukkonen. A comparison of approximate string matching algorithms. *Software—Practice and Experience*, 26(12):1439–1458, December 1996.

[Ken91] W. Kent. The breakdown of the information model in multi-database systems. *SIGMOD Record*, 20(4):10–15, December 1991.

[Kit86] J. Kittler. Feature selection and extraction. In T. Y. Young and K. S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*, pages 59–83, Orlando, FL, 1986. Academic Press.

[Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[KRR02] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In Philip A. Bernstein et al., editors, *VLDP 2002: proceedings of the Twenty-Eighth International Conference on Very Large Data Bases, Hong Kong SAR, China, 20–23 August 2002*, pages 275–286, Los Altos, CA 94022, USA, 2002. Morgan Kaufmann Publishers.

[Kru78] C.L. Krumhansl. Concerning the applicability of geometric models to similar data: The interrelationship between similarity and spatial density. *Psychological Review*, 85(5):445–463, 1978.

[KS83] J. B. Kruskal and D. Sankoff. An anthology of algorithms and concepts for sequence comparison. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.

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

[KS00] N. Koudas and K. C. Sevcik. High dimensional similarity joins: Algorithms and performance evaluation. *IEEETKDE: IEEE Transactions on Knowledge and Data Engineering*, 12, 2000.

[KVI96] P. Krishnan, J.S. Vitter, and B.R. Iyer. Estimating alphanumeric selectivity in the presence of wildcards. In H.V. Jagadish and I.S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 282–293. ACM Press, 1996.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966.

[LH82] J. De Leeuw and W. Heiser. Theory of multidimensional scaling. In *Classification, pattern recognition and reduction of dimensionality*, pages 285–316. North-Holland, Amsterdam, 1982.

[Li95] Wen-Syan Li. Knowledge gathering and matching in heterogeneous databases. In *AAAI Spring Symposium on Information Gathering*, 1995.

[LMP01a] S. Luján-Mora and M. Palomar. Comparing string similarity measures for reducing inconsistency in integrating data from different sources. *Lecture Notes in Computer Science*, 2118, 2001.

[LMP01b] S. Luján-Mora and M. Palomar. Reducing Inconsistency in Integrating Data from Different Sources. In M. Adiba, C. Collet, and B.P. Desai, editors, *Proc. of Int. Database Engineering and Applications Symposium (IDEAS 2001)*, pages 219–228, Grenoble, France, 2001. IEEE Computer Society.

[LRO96] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman et al., editors, *Proceedings of the twenty-second international Conference on Very Large Data Bases, September 3–6, 1996, Mumbai (Bombay), India*, pages 251–262, Los Altos, CA 94022, USA, 1996. Morgan Kaufmann Publishers.

[LSPR93] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *International Conference on Data Engineering*, pages 294–301, Los Alamitos, Ca., USA, April 1993. IEEE Computer Society Press.

[ME96] A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International*

Conference on Knowledge Discovery and Data Mining (KDD-96), page 267. AAAI Press, 1996.

[ME97] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)*, 1997.

[MGG93] D. L. Medin, R. L. Goldstone, and D. Gentner. Respects for similarity. *Psychological Review*, 100(2):254–278, April 1993.

[MHLL99] W. May, R. Himmeröder, G. Lausen, and B. Ludäscher. A unified framework for wrapping, mediating and restructuring information from the web. In P. P. Chen, D. W. Embley, J. Kouloumdjian, S. W. Liddle, and J. F. Roddick, editors, *Advances in Conceptual Modeling: ER '99, Paris, France, Proceedings*, volume 1727 of *Lecture Notes in Computer Science*, pages 307–320. Springer, 1999.

[Mor96] William Morris, editor. *The American Heritage Dictionary of the English Language*. Houghton Mifflin, Boston, third edition, 1996.

[MRJ99] P. Missier, M. Rusinkiewicz, and W. Jin. Multidatabase Languages. In A. K. Elmagarmid, A. Sheth, and M. Rusinkiewicz, editors, *Management of Heterogeneous and Autonomous Database Systems*, pages 175–216. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[Nav01] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[Nav02] D. Navarro. *Representing Stimulus Similarity*. Dissertation, University of Adelaide, December 2002.

[NBY98] G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electonic Journal*, 1(2), 1998.

[NBY99] G. Navarro and R. Baeza-Yates. A new indexing method for approximate string matching. *Lecture Notes in Computer Science*, 1645, 1999.

[NBYST01] G. Navarro, R.A. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19 – 27, dec 2001.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search of similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[OR18] M. Odell and R. C. Russell. The soundex coding system, 1918. U.S. Patents 1261167 (1918) and 1435663 (1922).

[Ore90] J. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2):343–352, June 1990.

[ÖV99] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Upper Saddle River, 2 edition, 1999.

[RB01] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, December 2001.

[RC96] K. Ramamritham and P. K. Chrysanthis. A Taxonomy of Correctness Criteria in Database Applications. *The VLDB Journal*, 5(1):85–97, January 1996.

[RR99] S. Ram and V. Ramesh. Schema Integration: Past, Present, and Future. In A. K. Elmagarmid, A. Sheth, and M. Rusinkiewicz, editors, *Management of Heterogeneous and Autonomous Database Systems*, pages 119–155. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[RS97] M.T. Roth and P.M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, and M.A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[SCS00] K. Sattler, S. Conrad, and G. Saake. Adding Conflict Resolution Features to a Query Language for Database Federations. In M. Roantree, W. Hasselbring, and S. Conrad, editors, *Proc. 3nd Int. Workshop on Engineering Federated Information Systems, EFIS'00,* *Dublin, Ireland, June*, pages 41–52, Berlin, 2000. Akadem. Verlagsgesellschaft.

[SE00] E. Schallehn and M. Endig. Using Source Capability Descriptions for the Integration of Digital Libraries. In H.-J. Klein, editor, *Tagungsband 12. GI-Workshop Grundlagen von Datenbanken*, volume 2005, pages 86–90, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, June 2000.

[Sea79] J. R. Searle. Metaphor. In Andrew Ortnony, editor, *Metaphor and Thought*, pages 265–277. Cambridge University Press, Cambridge, England, 1979.

[She62a] R. N. Shepard. The analysis of proximities: multidimensional scaling with an unknown distance function. I. *Psychometrika*, 27:125–140, 1962.

[She62b] R. N. Shepard. The analysis of proximities: multidimensional scaling with an unknown distance function. II. *Psychometrika*, 27:219–246, 1962.

[SJ97] S. Santini and R. Jain. Similarity is a geometer. *Multimedia Tools and Applications*, 5(3):277–306, 1997.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SL91] B. Salzberg and D. B. Lomet. Spatial database access methods. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(3):5–15, September 1991.

[SM96] H. Shang and T. H. Merrett. Tries for approximate string matching. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):540–547, 1996.

[SN90] M. Schrefl and E.J. Neuhold. A Knowledge-based Approach to Overcome Structural Differences in Object Oriented Database Integration. In R. A. Meersman, S. Zhongzhi, and K. Chen-Ho, editors, *Artificial Intelligence in Databases and Information Systems, Proc. of the IFIP WG 2.6 Working Conf., DS-3, Guangzhou, China, July, 1988*, pages 265–304, Amsterdam, 1990. North-Holland.

[SP91] S. Spaccapietra and C. Parent. Conflicts and Correspondence Assertions in Interoperable Databases. *ACM SIGMOD Record*, 20(4):49–54, December 1991.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, July 1992.

[SRF87] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+–Tree: A Dynamic Index for Multi-Dimensional Objects. In P. M. Stocker and W. Kent, editors, *Proc. of the 13th Int. Conf. on Very Large Data Bases, VLDB'87, Brighton, England, September 1–4, 1987*, pages 507–518, Los Altos, CA, 1987. Morgan Kaufmann Publishers.

[SS03] E. Schallehn and K. Sattler. Using Similarity-based Operations for Resolving Data-level Conflicts. In A. James, B. Lings, and M. Younas, editors, *Advances in Databases, 20th British National Conf. on Databases, BNCOD 20, Coventry, UK, July 2003*, volume 2712 of *Lecture Notes in Computer Science*, pages 172–189, Berlin, 2003. Springer-Verlag.

[SSA02] K. Shim, R. Srikant, and R. Agrawal. High-dimensional similarity joins. *Knowledge and Data Engineering*, 14(1):156–171, 2002.

[SSS01] E. Schallehn, K. Sattler, and G. Saake. Advanced grouping and aggregation for data integration. In *Proc. 10th International Conference on Information and Knowledge Management, CIKM'01, Atlanta, GA*, pages 547–549, 2001.

[SSS02] E. Schallehn, K. Sattler, and G. Saake. Extensible and similarity-based grouping for data integration *Poster paper*. In Rakesh Agrawal, Klaus Dittrich, and Anne H.H. Ngu, editors, *8th Int. Conf. on Data Engineering (ICDE), 26 February - 1 March 2002, San Jose, CA*, page 277, 2002.

[SSS04] E. Schallehn, K. Sattler, and G. Saake. Efficient Similarity-based Operations for Data Integration. *Data and Knowledge Engineering Journal*, 48(3):361–387, 2004.

[Ste46] S. S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[TAH$^+$96] M. Tork Roth, M. Arya, L. M. Haas, M. J. Carey, W. Cody, R. Fagin, P. M. Schwarz, J. Thomas, and E. L. Wimmers. The Garlic Project.

In H. V. Jagadish and I. S. Mumick, editors, *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada*, volume 25 of *ACM SIGMOD Record*. ACM Press, June 1996.

[TCY92] F. Tseng, A. Chen, and W. Yang. A probabilistic approach to query processing in heterogeneous database systems. In *Proceedings of the 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pages 176–183, 1992.

[TG82] A. Tversky and I. Gati. Similarity, seperability, and the triangle inequality. *Psychological Review*, 89(2):123–154, 1982.

[Tho95] Della Thompson, editor. *The Concise Oxford Dictionary of Current English*. Oxford University Press, ninth edition, 1995.

[Tic84] W. F. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems*, 2(4):309–321, 1984.

[Tor52] W. S. Torgerson. Multidimensional scaling. I. Theory and method. *Psychometrika*, 17:401–419, 1952.

[Tor58] W. S. Torgerson. *Theory and Methods of Scaling*. John Wiley and Sons, New York, 1958.

[Tor65] W. S. Torgerson. Multidimensional scaling of similarity. *Psychometrika*, 30:379–393, 1965.

[Tve77] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[Ukk92] E. Ukkonen. Approximate string-matching with *q*-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[Ukk93] E. Ukkonen. Approximate string-matching over suffix trees. In A. Apostolico, M. Crochemore, and Z. Galil a. Udi Manber, editors, *Combinatorial Pattern Matching, 4th Annual Symposium*, volume 684 of *Lecture Notes in Computer Science*, pages 228–242, Padova, Italy, 1993. Springer.

[VP97] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In M. Jarke, M.J. Carey, K.R.

Dittrich, F.H. Lochovsky, P. Loucopoulos, and M.A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 256–265. Morgan Kaufmann, 1997.

[VPZ88] J. Veijalainen and R. Popescu-Zeletin. Multidatabase systems in iso/osi environments. In N.E. Malagardis and T.J. Williams, editors, *Standards and Economic Development in Information Technology*, pages 83–97, 1988.

[Wei99] E. W. Weisstein. *The CRC Concise Encyclopedia of Mathematics*. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1999.

[WG97] Gio Wiederhold and Michael R. Genesereth. The conceptual basis for mediation services. *IEEE Expert*, 12(5), 1997.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, March 1992.

[Wie93] G. Wiederhold. Intelligent integration of information. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):434–437, June 1993.

[Wie94] G. Wiederhold. Interoperation, mediation and ontologies. In *Int. Symposium on 5th Generation Computer Systems; Workshop on Heterogeneous Cooperative Knowledge-Bases*, volume W3, pages 33–48, Tokyo, Japan, 1994.

[WZ00] H. Wang and C. Zaniolo. Using sql to build new aggregates and extenders for object- relational systems. In *Proc. of 26th Int. Conf. on Very Large Data Bases (VLDB'00), Cairo, Egypt*, pages 166–175. Morgan Kaufmann, 2000.

[YGM95] T. W. Yan and H. Garcia-Molina. Duplicate removal in information dissemination. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95)*, pages 66–77, San Francisco, Ca., USA, September 1995. Morgan Kaufmann Publishers, Inc.

[ZHKF95] G. Zhou, R. Hull, R. King, and J. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of 3rd Intl. Conf. on Cooperative Information Systems (CoopIS-95), Vienna, Austria*, 1995.