T

Table

HANS HINTERBERGER
ETH Zurich, Zurich, Switzerland

Synonyms

Table; List; Matrix

Definition

A data structure to organize the tuples of a relation: $\{<value_{ix},\ value_{iy},...>,\ <value_{jx},\ value_{jy},...>,\ ...\}$ in a structured way.

An arrangement of numbers, words, symbols or items of any kind, in columns and rows.

One-way table. The values of one or more variables (sets of data) are displayed either horizontally or vertically as a list. Each row or each column represents one data item.

Two-way table. Also known as a contingency table or cross tabulation (cross tab). The rows are labeled with values of one of two variables and the columns with values of the other. The cell at each row and column intersection displays the value of a characteristic of the data that is shared by the two variables when they have the value displayed in the row and column headings.

Multi-way table. The general idea of cross tabulation extended to more than two variables. The rows or columns or both are grouped by values of additional variables.

Key Points

In database terminology a (one-way) table is the data structure used to represent data that are organized with the relational model.

Two-way and multi-way tables are widely used tools to explore data by examining frequencies of observations that belong to specific categories on more than one variable.

In the context of data visualization, tables are an effective way to show exact numerical values and are usually better suited than graphical methods to represent small data sets. They assist users in making comparisons and provide them with a convenient way of storing data for rapid reference. Tables can be formatted in many different ways to suit the type of data, the purpose of the table and its intended use.

Ehrenberg [3] discusses tables in the context of readability, Bertin [1] considers tables as part of an information processing system, Card et al. [2] use them as a step in the process of mapping data to visual form while Harris [4] lists terms and key elements used in the design of tables.

The terminology for tables has also been influenced by the widespread use of table-based spreadsheet programs. One example is the notion of a *pivot table*, a practical feature to reorganize lists as user specified tables, particularly useful for cross tabulations.

Cross-references

► Chart
► Data Visualization
► Tabular Data

Recommended Reading

1. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin, New York, 1981.
2. Card S.K., MacKinlay J.D., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann, San Francisco, CA, 1999.
3. Ehrenberg A.S.C. A Primer in Data Reduction. Wiley, Chichester, UK, 1982.
4. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference, Oxford University Press, New York, 1999.

Table Design

► Physical Database Design for Relational Databases

Table Normalization

► Physical Database Design for Relational Databases

Tabular Data

JOSEP DOMINGO-FERRER
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms
Table

Definition
From microdata, *tabular data* can be generated by crossing one or more categorical attributes. Formally, a table is a function

$$T : D(V_{i1}) \times D(V_{i2}) \times \cdots \times D(V_{il}) \to \mathbb{R} \text{ or } \mathbb{N}$$

where $l \leq t$ is the number of crossed categorical attributes V_{ij} and $D(V_{ij})$ is the domain of attribute V_{ij}.

Key Points
There are two kinds of tables: *frequency tables* that display the count of respondents at the crossing of the categorical attributes (in $\mathbb{N}$) and *magnitude tables* that display information on a numerical attribute at the crossing of the categorical attributes (in $\mathbb{R}$). For example, given some census microdata containing attributes "Job" and "Town," one can generate a *frequency table* displaying the count of respondents doing each job type in each town. If the census microdata also contain the "Salary," attribute, one can generate a magnitude table displaying the average salary for each job type in each town. The number n of cells in a table is normally much less than the number r of respondent records in a microdata file. However, tables must satisfy several linear constraints: marginal row and column totals. Additionally, a set of tables is called *linked* if they share some of the crossed categorical attributes: for example "Job" × "Town" is linked to "Job" × "Gender."

Cross-references
► Inference Control in Statistical Databases
► Microdata

Tamper-Proof Hardware

► Trusted Hardware

Tape Libraries

► Storage Devices

Tapes

► Storage Devices

Task

► Activity

Taxonomies

► Lightweight Ontologies

Taxonomy: Biomedical Health Informatics

VIPUL KASHYAP
Partners Healthcare System, Wellesley, MA, USA

Synonyms
Health informatics; Healthcare informatics; Biomedical informatics

Definition
Health informatics or medical informatics is the intersection of information science, computer science, and health care [2]. It deals with the resources, devices, and methods required to optimize the acquisition, storage, retrieval, and use of information in health and biomedicine. Health informatics tools include not only computers but also clinical guidelines, formal medical terminologies, and information and communication systems. Subdomains of (bio) medical or health care informatics include: clinical informatics, nursing

informatics, imaging informatics, consumer health informatics, public health informatics, dental informatics, clinical research informatics, bioinformatics, veterinary informatics, pharmacy informatics, and healthcare management informatics. An alternative characterization refers to this field as biomedical informatics [1] takes a broader perspective including the application of computer and information science, informatics, cognitive science and human computer interaction in the practice of biological research, biomedical science, medicine and healthcare [3].

Foundations

This field is now characterized by means of a taxonomy comprising of the various subdomains identified above. The key sub domains are:

1. *Clinical Informatics* focuses on computer applications that address medical data (collection, analysis, representation), and is a combination of information science, computer science, and clinical science designed to assist in the management and processing of data, information and knowledge to support the practice and delivery of clinical care. Key activities covered by this area include Electronic Medical Records (EMRs), Decision Support Systems, Medical Data Mining, Hospital Information Systems and Laboratory Information Systems.

2. *Nursing Informatics* is the multidisciplinary scientific endeavor of analyzing, formalizing, and modeling how nurses collect and manage data, process data into information and knowledge, make knowledge-based decisions and inferences for patient care, and use this empirical and experiential knowledge in order to broaden the scope and enhance the quality of their professional practice. The scientific methods central to nursing informatics are focused on: (i) Using a discourse about motives for computerized systems, (ii) Analyzing, formalizing and modeling nursing information processing and nursing knowledge for all components of nursing practice: clinical practice, management, education and research, (iii) Investigating determinants, conditions, elements, models and processes in order to design, and implement as well as test the effectiveness and efficiency of computerized information, (tele)communication and network systems for nursing practice, and (iv) Studying the effects of these systems on nursing practice.

3. *Imaging Informatics* combines knowledge and capabilities from the fields of medicine, medical imaging (also known as diagnostic radiology), biomedical informatics and Information Technology. Imaging informatics includes the mining of information or knowledge from medical image databases, the use of technology to enhance the medical image interpretation process, and the utility of computer software to digitize medical imaging. The more commonly recognized areas of Imaging Informatics include Picture Archiving and Communications Systems (PACS), Radiology Information Systems (RIS), and Computer-Aided Detection and Diagnosis (CAD).

4. *Consumer Health Informatics* has been defined as a branch of medical informatics that analyses consumers' needs for information; studies and implements methods of making information accessible to consumers; and models and integrates consumers' preferences into medical information systems.

5. *Public Health Informatics* has been defined as the systematic application of information and computer science and technology to public health practice, research, and learning. It is distinguished from healthcare informatics by emphasizing data about populations rather than that of individuals. The activities of public health informatics can be broadly divided into the collection, storage, and analysis of data related to public health.

6. *Dental Informatics* is the understanding, skills and tools that enable the sharing and use of information to promote oral health and improve dental practice, research, education, and management. It encompasses electronic health records, CAD/CAM technology, diagnostic digital imaging and administrative information for all dentistry disciplines.

7. *Clinical Research Informatics* is concerned with the application of informatics theory and methods to design, conduct and improve clinical research and disseminate the knowledge gained. It overlaps considerably with the related rapidly developing domain of Translational Research Informatics. Clinical research is defined by the National Institutes of Health (NIH) as being comprised of studies and trials in human subjects that fall into the three sub-categories: (i) Patient-oriented research conducted with human subjects (or on material of human origin such as tissues, specimens and cognitive phenomena) for which an investigator (or

colleague) directly interacts with human subjects. It includes research on mechanisms of human disease, therapeutic interventions, clinical trials, or development of new technologies; (ii) Epidemiologic and behavioral studies; and (iii) Outcomes research and health services research.

8. *Translational Research Informatics* as defined by the NIH includes two areas of translation. One is the process of applying discoveries generated during research in the laboratory, and in preclinical studies, to the development of trials and studies in humans. The second area of translation concerns research aimed at enhancing the adoption of best practices in the community. Cost-effectiveness of prevention and treatment strategies is also an important part of translational science.

9. *Bioinformatics and Computational Biology* involves the use of techniques including applied mathematics, informatics, statistics, computer science, artificial intelligence, chemistry, and biochemistry to solve biological problems usually on the molecular level. The core principle of these techniques is the use of computing resources in order to solve problems on scales of magnitude far too great for human discernment. Research in computational biology often overlaps with systems biology. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, and the modeling of evolution.

10. *Pharmacy Informatics* is the application of computers to the storage, retrieval and analysis of drug and prescription information. Pharmacy informaticists work with pharmacy information management systems that help the pharmacist make excellent decisions about patient drug therapies with respect to medical insurance records, drug interactions, as well as prescription and patient information. Pharmacy informatics is the study of interactions between people, their work processes and engineered systems within health care with a focus on pharmaceutical care and improved patient safety.

Key Applications

This field is characterized by the key aspects and applications, some of which are discussed in more detail through cross-referenced encyclopedia entries:

- Systems and architectures for electronic medical records.
- Health information systems used for billing, scheduling and other financial operations.
- Information systems used for biomedical and clinical research.
- Decision support systems in healthcare, including clinical decision support systems.
- Standards (e.g., DICOM, HL7) and integration profiles (e.g., Integrating the Healthcare Enterprise) to facilitate the exchange of information between healthcare information systems.
- Information Models such as the HL7/RIM and the openEHR for storage and representation of clinical data in a standardized manner.
- Controlled medical vocabularies (CMVs) such as the Systematized Nomenclature of Medicine, Clinical Terms (SNOMED CT), Logical Observation Identifiers Names and Codes (LOINC), OpenGALEN Common Reference Model or the highly complex UMLS used to allow a standardized way of characterizing medication, laboratory results, and clinical findings.
- Use of hand-held or portable devices to assist providers with data entry/retrieval or medical decision-making, sometimes called mHealth.

Cross-references

▶ Biomedical Data/Content Acquisition, Curation
▶ Biomedical Image Data Types and Processing
▶ Biomedical Scientific Textual Data Types and Processing
▶ Clinical Data Acquisition, Storage and Management
▶ Clinical Data and Information Models
▶ Clinical Data Quality and Validation
▶ Clinical Decision Support
▶ Clinical Document Architecture
▶ Clinical Event
▶ Clinical Knowledge Repository
▶ Clinical Observation
▶ Clinical Order
▶ Computerized Physician Order Entry
▶ Data Privacy and Patient Consent
▶ Data, Text, and Web Mining in Healthcare
▶ Electronic Health Record
▶ Enterprise Terminology Services
▶ Evidence Based Medicine
▶ Executable Knowledge

Recommended Reading

1. Biomedical Informatics, http://en.wikipedia.com/wiki/Biomedical_informatics
2. Health Informatics, http://en.wikipedia.org/wiki/Health_informatics
3. Shortliffe E.H. and Cimino J.J. eds. Biomedical Informatics: Computer Applications in Health Care and Biomedicine, 3rd edn. Springer, New York, 2006.

Telic Distinction in Temporal Databases

Vijay Khatri[1], Richard T. Snodgrass[2], Paolo Terenziani[3]
[1]Indiana University, Bloomington, IN, USA
[2]University of Arizona, Tucson, AZ, USA
[3]University of Turin, Turin, Italy

Synonyms

Point-versus period-based semantics

Definition

In the context of temporal databases, telic (atelic) data are used to store telic (atelic) facts, and the distinction between telic and atelic data are drawn using the properties of downward and upward inheritance.

- *Downward inheritance.* The *downward inheritance* property implies that one can infer from temporal data d that holds at valid time t (where t is a time period) that d holds in any sub-period (and sub-point) of t.
- *Upward inheritance.* The *upward inheritance* property implies that one can infer from temporal data d that holds at two consecutive or overlapping time periods t_1 and t_2 that d holds in the union time period $t_1 \cup t_2$.

In temporal databases, the semantics of atelic data implies that both downward and upward inheritance holds; on the other hand, neither downward nor upward inheritance holds for telic data.

Historical Background

The distinction between telic and atelic facts dates back to Aristotle's categories [2] and has had a deep influence in the Western philosophical and linguistic tradition. In particular, the distinction between different classes of sentences (called aktionsart classes) according to their linguistic behavior and temporal properties is at the core of the modern linguistic tradition (consider, e.g., the milestone categorization by Vendler [11]). For instance, the upward and downward inheritance properties were used (without adopting terminology, which is imported from Shoham [7] and, more generally, from the artificial intelligence tradition) by Dowty [4] in order to distinguish between Vendler's accomplishments (telic facts) and states along with processes (atelic facts). Starting from the pioneering work by Bennet and Partee [3], several linguistic approaches have pointed out that the traditional point-based semantics, in which facts can be evaluated at each time point, properly applies only to atelic facts, while a period-based semantics is needed in order to properly cope with telic facts. Starting from Allen's milestone approach [1], the telic/atelic dichotomy has also played a major role in the area of artificial intelligence. In the field of temporal databases, the point-based vs. period-based dichotomy was initially related to representation and query evaluation issues (rather than to data semantics); later the connection was made between Aristotle's categories and the telic/atelic data semantics [9,10]. It is the emphasis on data semantics that renders "telic" and "atelic" the preferred term.

Foundations

The distinction between telic and atelic data regards the time when facts hold or occur, i.e., their valid time. The following discussion focuses on the temporal *semantics* of data and queries, independent of the *representation* that is used for time. Moreover, while in several database approaches the semantics of data is not distinguished from the semantics of the query, this presentation follows the logical tradition, stating that data has its own semantics independently of any query language and operators just in the same way in which a knowledge base of logical formulæ have their own semantics – usually expressed in model-theoretic terms.

Data Semantics

In the linguistic literature, most approaches classify facts (or sentences describing facts) according to their temporal properties. In particular, most approaches distinguish between telic and atelic facts, and prior research (see, e.g., [3]) points out that, while the point-based semantics is useful to cope with atelic facts, it is not suitable for telic ones, for which a period-based semantics is needed.

Point-based semantics of data: The data in a temporal relation is interpreted as a sequence of states (with each state a conventional relation, i.e., a set of tuples) indexed by points in time. Each state is independent of every other state.

Such temporal relations can be encoded in many different ways (data language). For example the following are three different encodings of the same information, within a point-based semantics, of John being married to Mary in the states indexed by the times 1, 2, 7, 8, and 9:

$$(i) < John, Mary||\{1, 2, 7, 8, 9\} > \in R$$

$$(ii) < John, Mary||\{[1-2], [7-9]\} > \in R$$

$$(iii) < John, Mary||[1-2] > \in R \text{ and}$$
$$< John, Mary||[7-9] > \in R$$

Independently of the representation, the point-based semantics implies that the fact denoted by <John, Mary> includes 5 individual states as follows:

$$1 \rightarrow \{< John, Mary >\}$$
$$2 \rightarrow \{< John, Mary >\}$$
$$7 \rightarrow \{< John, Mary >\}$$
$$8 \rightarrow \{< John, Mary >\}$$
$$9 \rightarrow \{< John, Mary >\}$$

Notice that the point-based semantics naturally applies to atelic facts, since both downward and upward inheritance are naturally supported.

Period-based semantics of data: Each tuple in a temporal relation is associated with a multiset of time periods, which are the temporal extents in which the fact described by the tuple occur. In this case, time periods are atomic primitive entities in the sense that they cannot be decomposed. Note, however, that time periods can overlap, unlike time points.

For example, let <John || {[10–20]}> represent the fact that John started to build a house at time 10 and finished at time 20. If a period-based semantics is adopted, the period [10–20] is interpreted as an atomic (indivisible) one.

$$[10, 20] \rightarrow \{< John >\}$$

Note that a period-based semantics does not imply that John built the house in [12–15], or at the time point 12, or at any other time period other than [10–20]. As a consequence, the period-based semantics is naturally suited to cope with telic facts, for which (by definition) neither downward nor upward inheritance hold.

Although several query and data representation languages include time periods, most temporal database approaches adopt, explicitly or implicitly, the point-based semantics, interpreting a temporal database as a set of conventional databases, each one holding at a specific snapshot of time. This is the approach followed, e.g., by the Bitemporal Conceptual Data Model (BCDM) [5], a model that has been proven to capture the semantic core of many prior approaches in the temporal database literature, including the TSQL2 "consensus" approach [8]. While point-based semantics works perfectly when coping with atelic data, the problem with using it to cope with telic fact is illustrated by the following example.

Example

Phone calls are durative telic facts. For instance, if John made a call to Mary from time 10 to time 12, he didn't make it from 10 to 11. Similarly, two consecutive calls, one from 10 to 12 (inclusive) and the other from 13 to 15 (inclusive), are clearly different from a single call from 10 to 15. However, such a distinction cannot be captured at the semantic level, if the point-based semantics is used. In fact, the point-based semantics for the two phone calls of John is as follows:

$$10 \rightarrow \{< John, Mary >\}$$
$$11 \rightarrow \{< John, Mary >\}$$
$$12 \rightarrow \{< John, Mary >\}$$

$$13 \rightarrow \{< John, Mary >\}$$
$$14 \rightarrow \{< John, Mary >\}$$
$$15 \rightarrow \{< John, Mary >\}$$

Based on point-semantics, there is no way of grasping that two different calls were made. In other words, there is a loss of information. Note that such a loss of information is completely independent of the representation language used to model data. For instance, the above example could be represented as

(i) $<$ John, Mary$||\{10, 11, 12, 13, 14, 15\} >\in$ R

(ii) $<$ John, Mary$||\{[10 - 12], [13 - 15]\}\} >\in$ R

(iii) $<$ John, Mary$||[10 - 12] >\in$ R and
$<$ John, Mary$||[13 - 15] >\in$ R

But, as long as the point-based semantics is used, the data semantics is the one elicited above.

On the other hand, independently of the representation formalism being chosen, the semantics of telic facts such as phone calls is properly coped with if the period-based semantics is used. For instance, in the phone example, the semantics

$$[10 - 12] \rightarrow \{< John, Mary >\}$$
$$[13 - 15] \rightarrow \{< John, Mary >\}$$

correctly expresses the distinction between the two consecutive phone calls.

In an analogous way, period-based semantics is not suitable to model atelic facts. In short, both upward and downward inheritance holds for them, and the period-based semantics does not support such properties.

Terenziani and Snodgrass have proposed a two-sorted data model, in which telic data can be stored in telic relations (i.e., relations to be interpreted using a period-based semantics) and atelic data in atelic relations (i.e., relations to be interpreted using a point-based semantics) [9].

Query Semantics

It is interesting that the loss of information due to the treatment of telic data in a point-based (atelic) framework is even more evident when queries are considered. Results of queries should depend only on the data semantics, not on the data representation. For instance, considering the phone example above (and independently of the chosen representation), queries about the number or duration of phone calls would not provide the desired answers. For instance, the number of calls from John to Mary would be one, and a call from John to Mary would (incorrectly) be

provided to a query asking for calls lasting for at least five consecutive units.

In order to cope with a data model supporting both telic and atelic relations, temporal query languages must be extended. Specifically, queries must cope with atelic relations, telic relations, or a combination of both.

Furthermore, linguistic research suggests a further requirement for telic/atelic query languages: *flexibility*. It is widely accepted within the linguistic community that while basic facts can be classified as telic or atelic, natural languages provides several ways to switch between the two classes. For instance, given a telic fact (such as "John built a house"), the progressive form (e.g., "John *was* build*ing* a house") coerces it into an atelic one, stripping away the culmination (and, in fact, "John was building a house" does not imply that "John built a house," i.e., that he finished it) [6]. For the sake of expressiveness, it is desirable that a database query language provides the same flexibility.

Queries About Atelic Data

As already mentioned above, most database approaches are interpreted (implicitly or explicitly) on the basis of the point-based semantics. Therefore, the corresponding algebraic operators already cope with atelic data. As an example, in BCDM, the union of two relations is simply obtained by taking the tuples of both relations, and "merging" the valid time of value equivalent tuples performing the union of the time points in their valid time. This definition is perfectly consistent with the "snapshot-by-snapshot" view enforced by the underlying point-based (atelic) semantics.

However, the algebrae in the literature also contain operators which contrast with such a "snapshot-by-snapshot" underlying semantics. Typical examples are temporal selection operators. For instance, whenever a duration is asked for (e.g., "retrieve all persons married for at least n consecutive time units"), the query implicitly relies on a telic view of data, in which snapshots are not taken into account independently of each others.

Queries About Telic Data

Algebraic operators on telic data can be easily defined by paralleling the atelic definitions, and considering that, in the telic case, the basic temporal primitives are not time points, but time periods [9]. For instance, telic union is similar to atelic one, except that the

merging of valid times of value-equivalent tuples is performed by making the union of multisets of time periods, considered as primitive entities, e.g.,

$$\{[10 - 12], [13 - 15]\} \cup \{[10 - 14], [13 - 18]\}$$
$$= \{[10 - 12], [13 - 15], [10 - 14], [13 - 18]\}$$

Note that temporal selection operators perfectly fit with the telic environment. On the other hand, algebraic operators that intuitively involve a snapshot-by-snapshot view of data (e.g., Cartesian product, involving a snapshot-by-snapshot intersection between valid times) have an awkward interpretation in the telic context; for this reason, difference and "standard" Cartesian product have not been defined in the telic algebra in [9].

Queries Combining Telic and Atelic Data

In general, if a two-sorted data model is used, queries combining relations of both kinds are needed. In general, such queries involve the (explicit or implicit) coercion of some of the relations, to make the sort of the relations consistent with the types of the operators being used. For instance, the following query utilizes the example atelic relation modeling marriages and the telic one considering phone calls: Who was being married when John was calling Mary?

In such a case, the English clause "when" demands for an atelic interpretation: the result can be obtained by first coercing the relation about phone calls into an atelic relation, and then by getting the temporal intersection through the application of the atelic Cartesian product.

On the other hand, the query "list the marriage ceremonies that had a duration of more than 3 hours" requires a coercion of marriages into a telic relation, so that the whole valid time is considered as (a set of) time periods (instead as a set of independent points, as in the atelic interpretation), and the telic temporal selection operator can be applied.

In general, two coercion operators need to be provided [9]. Coercion from telic to atelic is easy: each time period constituting the (semantics of the) valid time is converted into the set of time points it contains, e.g., *to-atelic*($\{[10-12],[13-15]\}$) = $\{10,11,12,13,14,15\}$. Of course, since multisets of time periods are more expressive than sets of time points, such a conversion causes a loss of information. On the other hand, coercion from atelic to telic demands the formation of time periods out of sets of points: the

output is the set of maximal convex time periods exactly covering the input set of time points, e.g., to-telic($\{10,11,12,13,14, 15\}$) = $\{[10–15]\}$.

Key Applications

Most applications involve differentiating between telic and atelic data.

Cross-references

▶ Atelic Data
▶ Period-Stamped Temporal Models
▶ Point-Stamped Temporal Models
▶ Temporal Query Languages

Recommended Reading

1. Allen J.F. Towards a general theory of action and time. Artif. Intell., 23:123–154, 1984.
2. Aristotle. The Categories, on Interpretation. Prior Analytics. Harvard University Press, Cambridge, MA, 1938.
3. Bennet M. and Partee B. Tense and Discourse Location in Situation Semantics. Indiana University Linguistics Club, Bloomington, 1978.
4. Dowty D. The effects of the aspectual class on the temporal structure of discourse, tense and aspect in discourse. Linguist. Philos., 9(1):37–61, 1986.
5. Jensen C.S. and Snodgrass R.T. Semantics of time-varying Iinformation. Inf. Syst., 21(4):311–352, 1996.
6. Moens M. and Steedman M. Temporal ontology and temporal reference. Comput. Linguist., 14(2):15–28, 1998.
7. Shoham Y. Temporal logics in AI: semantical and ontological considerations. Artif. Intell., 33:89–104, 1987.
8. Snodgrass R.T. (ed.). The Temporal Query Language TSQL2. Kluwer, Norwell, MA, 1995.
9. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a proper treatment of the Telic/Atelic distinction. IEEE Trans. Knowl. Data Eng., 16(4):540–551, 2004.
10. Terenziani P., Snodgrass R.T., Bottrighi A., Torchio M., and Molino G. Extending temporal databases to deal with Telic/Atelic medical data. Artif. Intell. Med., 39(2):113–126, 2007.
11. Vendler Z. Verbs and times. In Linguistics in Philosophy. Cornell University Press, New York, NY, 1967, pp. 97–121.

Telos

MANOLIS KOUBARAKIS
University of Athens, Athens, Greece

Definition

Telos (From the Greek word $\tau\acute{\epsilon}\lambda o\varsigma\tau\Theta$ which means *end*; the object aimed at in an effort; purpose.) is a

knowledge representation language designed especially to support the development of information systems. Telos is based on the premise that information system development is knowledge-intensive and that the main design goal of any language intended for the task should be to formally represent the relevant knowledge. Telos is founded on core concepts from data modeling and knowledge representation, and shares ideas with semantic networks and frame systems, semantic and object-oriented data models, logic programming and deductive databases. The main features of Telos include: a structurally object-oriented framework which supports aggregation, generalization and classification; a novel treatment of attributes as first class citizens in the language; a powerful way of defining meta-classes; an explicit representation of time; and facilities for specifying integrity constraints and deductive rules.

Historical Background

The research on Telos follows the paradigm of a number of software engineering projects initiated around the 1990s with the premise that software development is *knowledge-intensive* and that the primary responsibility of any language intended to support this task is to be able to *formally represent the relevant knowledge*. Thus, Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle.

Telos has evolved from RML (a requirements modeling language developed in Sol Greenspan's Ph.D. thesis), and later CML (presented in the Master thesis of Martin Stanley at the University of Toronto). The main difference between RML and CML is that CML adopts a more sophisticated model for representing knowledge, and supports the representation of temporal knowledge and the definition of meta-classes. Telos is essentially a "cleaned-up" and improved version of CML which was originally defined and implemented in the Master thesis of Manolis Koubarakis at the University of Toronto. The original paper on Telos is [7]. Ontological and semantical issues for Telos are discussed in [10]. The history of knowledge representation languages for information systems development related to Telos is surveyed in [3]. An important dialect of Telos is O-Telos defined in the Ph.D. thesis of Manfred Jeusfeld at the University of Passau, and implemented in the ConceptBase system [4]. Since ConceptBase is the most mature implementation of Telos available today, this entry uses the ConceptBase syntax for Telos.

Foundations

The main (and essentially the only) concept of Telos is the *proposition*. Propositions are used to model any aspect of the application domain. Propositions have unique identities and are distinguished into *individuals* and *attributes*. Individuals are intended to represent entities in the application domain (concrete ones such as John Doe, or abstract ones such as the class of all persons). Attributes represent binary relationships between entities (concrete or abstract). Two special kinds of attribute propositions exist: instantiation propositions and specialization propositions. The proposition abstraction gives great flexibility to Telos users. Everything in the application domain that is represented by a proposition (e.g., an entity or a relationship) immediately becomes a first-class citizen of the knowledge base.

Propositions

Every proposition p consists of an *identifier*, a *source*, a *label* and a *destination*, denoted by the functions id(p), from(p), label(p) and to(p). For example, the following are propositions:

```
P1: [..., Martin, ...]
P2: [..., "21 Elm Avenue," ...]
P3: [Martin, homeAddress, "21 Elm
    Avenue"]
P4: [..., Person, ...]
P5: [..., GeographicLocation, ...]
P6: [Person, address,
    GeographicLocation]
```

Propositions in Telos are what *objects* are in object-oriented formalisms but also what *statements* are in logic-based formalisms. Thus, an application can use the above propositions to represent the following pieces of knowledge:

- P1: There is somebody called Martin.
- P2: There is something called "21 Elm Avenue."
- P3: Martin lives in 21, Elm Avenue.
- P4: There is an abstract concept, the class of all persons.
- P5: There is an abstract concept, the class of all geographic locations.
- P6: Persons have addresses that are geographic locations.

P1, P2, P4 and P5 are individual propositions while P3 and P6 are attribute propositions. The source and destination components of an individual

proposition are not important, thus they are shown as "…". Notice that while P1 and P2 represent concrete individuals, P4 represents an abstract one, the class of all persons. Similarly, P3 represents a concrete relationship (relating Martin with his address) while P6 represents an abstract one (relating the class of all persons with the class of all geographic locations).

Following are some examples of special propositions:

```
P7: [P1, *instanceOf, P4]
P8: [P3, *instanceOf, P6]
P9: [..., Employee, ...]
P10:[P9, *isA, P4]
```

P7 and P8 are *instantiation propositions*. P7 represents the fact that Martin is a member of the class of all persons. P8 represents the fact that the concrete relationship relating Martin with his address is an instance of the abstract relationship relating the class of all persons with the class of all geographic locations. Finally, P10 is a *specialization proposition* asserting that every employee is a person.

A graphical view of some of the above propositions is given in Fig. 1.

Organizing Propositions

Propositions (individual or attribute ones) can be organized along three dimensions: *decomposition/aggregation*, *instantiation/classification* and *specialization/generalization*.

The aggregation dimension enables one to see an entity of the application domain as a collection of propositions with a common proposition as source. For example, individual Martin can be seen to be the following aggregation:

```
{Martin,
 [Martin, age, 35],
 [Martin, homeAddress, "21 Elm
 Avenue"],
 [Martin, workAddress, "10 King's
 College Road"] }
```

The classification dimension calls for each proposition to be an instance of one or more generic propositions or classes. Classes are themselves propositions, and therefore instances of other, more abstract classes. For example, Person is a class and Martin is an instance of this class. Similarly,

```
[Person, address, GeographicLocation]
```
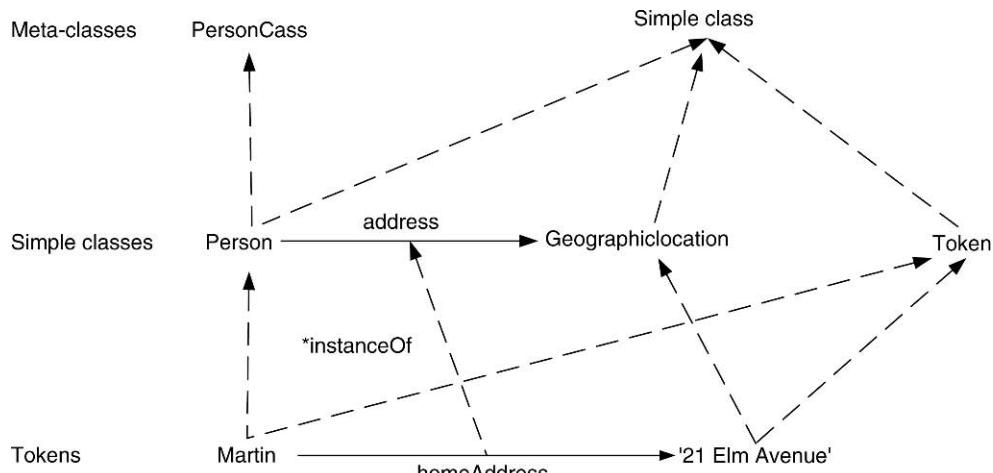
is a class and

```
[Martin, homeAddress, "21 Elm Avenue"]
```

is an instance of this class.

With respect to the classification dimension, propositions can be distinguished into:

- *Tokens:* propositions having no instances and intended to represent concrete entities in the application domain.
- *Simple classes:* propositions having only tokens as instances.

Telos. Figure 1. A graphical view of a set of Telos propositions.

- *Meta-classes:* propositions having only simple classes as instances.
- *Meta-meta-classes:* propositions having only meta-classes as instances.
- ...

Thus, classification in Telos defines an *unbounded* linear order of planes of ever more abstract propositions. Implementations restrict this unbounded hierarchy (e.g., ConceptBase restricts it to four levels: tokens to meta-meta-classes). There are also *ω-classes* with instances along more than one plane:

- `Proposition`. Contains all propositions as instances.
- `Class`. Contains all classes as instances.
- `Token`. Contains those individuals that may never have instances themselves.
- `SimpleClass`. Contains individuals that may have instances which are tokens.
- `MetaClass`. Contains individuals that may have simple classes as instances.
- `MetametaClass`. Contains individuals that may have meta-classes as instances.
- ...

Classification in Telos is a form of *weak typing*: the classes of which a structured object is an instance determine the kinds of attributes it can have optionally, and the properties it must satisfy. For example, by virtue of being an instance of `Person`, `Martin` can have attributes that are instances of the attribute class

`[Person,address,GeographicLocation].`

These *zero or more* attributes can have arbitrary labels, e.g., `homeAddress` and `workAddress`, but their values must be instances of `GeographicLocation`.

Finally, classes in Telos can be specialized along generalization or ISA hierarchies. For example, `Person` may have subclasses such as `Professor`, `Student`, and `TeachingAssistant`. Classes may form a partial order, rather than a tree (i.e., multiple inheritance is supported). Non-token attributes of a class are inherited by more specialized ones and can be refined. Inheritance in Telos is strict rather than default.

Interacting with Telos Knowledge Bases

A few examples of Telos are now given. The example application considered is the development of an information system to support organizing international scientific conferences. The knowledge to be represented in this case is about entities such as papers, authors, conferences etc.

The original definition of Telos in [7] defines the operations TELL, UNTELL, RETELL and ASK for interacting with a Telos knowledge base. These operations can be used to add new knowledge, discard existing knowledge, update existing knowledge and query a knowledge base, respectively. Implementations such as ConceptBase have followed the original definition and offer these (and other) operations. The above operations have Telos statements such as the following as their means of interaction with a knowledge base:

```
Individual p133 in Token, Paper with
    author
        firstAuthor: Stanley;
        secondAuthor: LaSalle;
        thirdAuthor: Wong
    title
        called: "The language Telos"
end
```

The above statement introduces an individual with name `p133`. The `in` clause specifies the classes of which `p133` is an instance (in this case, the predefined class `Token` and the application class `Paper`). The `with` clause introduces `p133`'s attributes. The first attribute of `p133` has label `firstAuthor` and is an instance of an attribute class which has source `Paper` and label `author` (the latter is denoted by the *attribute category* `author`). Before introducing individual paper `p133`, one might have defined the class of all papers using the following statement:

```
Individual Paper in SimpleClass with
    attribute
        author: Person;
        referee: Person;
        title: String;
        pages: Integer
end
```

A class definition prescribes the attributes that can be associated with its instances: `p133` can have `author`, `referee`, `title` and `page` attributes as seen previously, because it is an instance of class `Paper` that has these *attribute classes*. Moreover,

```
       [p133,firstAuthor,Stanley]
```

is an instance of attribute class

```
       [Paper,author,Person]
```

in exactly the same sense that `p133` is an instance of
`Paper`.

Once `Paper` has been defined, one can introduce
specializations such as `InvitedPaper` using the `isA`
clause of class definitions:

```
Individual AcceptedPaper in
SimpleClass isA Paper with
   attribute
      session: ConfProgrammeSession
   end
```

`AcceptedPaper` inherits all attributes from `Paper`
and adds a `session` attribute, to indicate the
programme session during which the accepted paper
will be presented.

Metaclasses

Metaclasses are a very powerful concept for modeling
power and extensibility in Telos. It is the metaclass
mechanism combined with its other features that
makes Telos a powerful modeling language (one might
wonder about this, since Telos offers only very simple
primitives). From a modeling point of view, one can use
Telos metaclasses in the following situations:

- To define concrete attributes of classes e.g., cardi-
  nality of a class. This is exactly the same to what a
  simple class does for its instances (tokens).
- To group together semantically similar classes of a
  domain in a generic way. For example, in the conf-
  erence organization example, the classes `Paper`,
  `Announcement`, `Letter`, `Memo` could be grou-
  ped under the metaclass `DocumentClass`.
- To define concepts that are built-in in other frame-
  works e.g., necessary attributes, single-valued attri-
  butes etc.
- To do other forms of meta-level logical reasoning
  (again, for language expressibility).

The conference organization example is now revisited
and defined:

```
DocumentClass in MetaClass with
   attribute
      source: AgentClass;
      content: SimpleClass;
```

```
      destination: AgentClass;
      cardinality: Integer
   end
```

The class `Paper` can now be defined as follows:

```
Paper in DocumentClass with
   source
      author: Person;
   content
      title: String;
      abstract: String
   cardinality
      how_many: 120
   end
```

Note that attribute categories such as `source` intro-
duced in metaclass `DocumentClass` are then used to
define attributes for the instance class `Paper` (this
mechanism is the same along the instantiation
hierarchy).

Integrity Constraints and Deductive Rules

Telos borrows the notions of integrity constraints
and deductive rules from logic-based formalisms such
as deductive databases. *Integrity constraints* are formu-
las that express conditions that knowledge bases
should satisfy. They are used to express rich language
or application semantics that cannot possibly be
expressed only by the structural framework of Telos.
Deductive rules are formulas that can be used to derive
new knowledge. Integrity constraints and deductive
rules in Telos are expressed in appropriately defined
assertional languages that are subsets of first-order
logic [4,7].

Integrity constraints and rules are defined as attri-
butes of Telos classes that are instances of the built-in
object `Class`. For example, the following Telos state-
ment defines well-understood constraints and rules
regarding employees, their managers and their respec-
tive salaries.

```
Class Employee with
  rule
    BossRule: $ forall e/Employee
    m/Manager(exists d/Department
    (e dept d) and (d head m))
    ==> (e boss m) $
  constraint
    SalaryBound: $ forall e/Employee
    b/Manager x,y/Integer(e boss b)
```

```
      and (e salary x) and (b salary y)
      ==> x <= y $
  end
```

Language Extensibility Through Metaclasses and Integrity Constraints

In Telos, one can use integrity constraints together with the metaclass mechanism to define concepts that are built-in in other representational frameworks. For example, in many object-oriented models one can constrain an attribute to be single-valued using some built-in construct of the model. In Telos, one can do this by using only the primitive mechanisms of the language as follows. First, one defines the class Single: (The syntax of this statement is from the original paper of Telos [7] (O-Telos allows one to specify the same thing in a slightly more complex way).)

```
Class Single
  components [Class, single, Class]
  in AttributeClass, MetaClass with
  integrityConstraint
    : $ forall u/Single
        p,q/Proposition(p in u) and
        (q in u) and from(p)=from(q)
        ==> p=q $
  end
```

Then, one uses attribute class Single in the definition of class Paper:

```
Individual Paper in SimpleClass with
    attribute
      author: Person;
      referee: Person
  single
      title: String;
      pages: Integer
  end
```

Now in every instance of Paper, a title attribute is constrained to be single-valued due to the integrity constrain and the instantiation relationships introduced by the above Telos statements.

Query Languages for Telos

The papers [4,7,11] give various query languages for Telos knowledge bases ranging from ones based purely on first-order logic [7] to ones exploiting the structurally object-oriented features of the language as well [4,11].

The paper [8] presents work on a knowledge base management system based on Telos.

Temporal Knowledge in Telos

The original paper of Telos [7] presents a powerful framework for representing and reasoning about temporal knowledge. In Telos, the history of an application domain is modeled by augmenting propositions with a *history time* i.e., an interval representing the time during which these facts are true in the application domain. Historical knowledge in Telos is allowed to be incomplete and a modification of Allen's interval algebra [1] is used to capture the relevant knowledge.

A knowledge base records essentially the *beliefs* of the system, which may be distinct from the actual state of the world at that time. So, for example, the title of a paper might have been changed in March, but the knowledge base is only told of it in May. Or one may make a correction to some previously told fact. Just like it represents the full history of an application domain, Telos also records the full history of its beliefs. For this reason, Telos represents *belief times*; these are intervals associated with every proposition in the knowledge base, which commence at the time when the operation responsible for the creation of the corresponding proposition was committed.

For efficiency reasons, implementations of Telos such as ConceptBase [4] have restricted the kinds of temporal knowledge that can be represented.

Telos and RDF

Telos is probably the pre-Web knowledge representation language most closely related to the Resource Description Framework (RDF) and the RDF Vocabulary Description Language or RDF Schema proposed by the W3C for representing knowledge about Web resources (see e.g., http://www.w3.org/TR/rdf-primer/). This relationship has been exploited by the prominent RDF query language RQL defined by ICS-FORTH [6] but also in the O-Telos-RDF proposal [9].

Key Applications

Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle [7]. The strengths of the language made it the choice of many prominent research projects in Europe and North America including DAIDA [5], ITHACA [2] and others.

URL to Code

The most mature implementation of Telos is the ConceptBase system available at http://conceptbase.cc/.

Cross-references

► Meta Model
► Object Data Models
► RDF Schema
► Semantic Data Models

Recommended Reading

1. Allen J. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Constantopoulos P., Jarke M., Mylopoulos J., and Vassiliou Y. The software information base: A server for reuse. VLDB J., 4(1):1–43, 1995.
3. Greenspan S.J., Mylopoulos J., and Borgida A. On formal requirements modeling languages: RML revisited. In Proc. 16th Int. Conf. on Software Eng., 1994, pp. 135–147.
4. Jarke M., Gallersdörfer R., Jeusfeld M.A., and Staudt M. ConceptBase – A deductive object base for meta data management. J. Intell. Inf. Syst., 4(2):167–192, 1995.
5. Jarke M., Mylopoulos J., Schmidt J.W., and Vassiliou Y. DAIDA: An environment for evolving information systems. ACM Trans. Inf. Syst., 10(1):1–50, 1992.
6. Karvounarakis G., Alexaki S., Christophides V., Plexousakis D., and Scholl M. RQL: A declarative query language for RDF. In Proc. 11th Int. World Wide Web Conference, 2002.
7. Mylopoulos J., Borgida A., Jarke M., and Koubarakis M. Telos: A language for representing knowledge about information systems. ACM Trans. Inf. Syst., 8(4):325–362, 1990.
8. Mylopoulos J., Chaudhri V.K., Plexousakis D., Shrufi A., and Topaloglou T. Building knowledge base management systems. VLDB J., 5(4):238–263, 1996.
9. Nejdl W., Dhraief H., and Wolpers M. O-Telos-RDF: A resource description format with enhanced meta-modeling functionalities based on O-Telos. In Proc. Workshop on Knowledge Markup and Semantic Annotation at the 1st Int. Conf. on Knowledge Capture, 2001.
10. Plexousakis D. Semantical and ontological consideration in Telos: A language for knowledge representation. Comput. Intell., 9:41–72, 1993.
11. Staudt M., Nissen H.W., and Jeusfeld M.A. Query by class, rule and concept. Appl. Intell., 4(2):133–156, 1994.

Temporal Access Control

Yue Zhang, James B. D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Time-based access control

Definition

Temporal access control refers to access control service that restricts granting of authorization based on time. The authorization may be given to a subject for a particular interval or duration of time or based on the temporal characteristics of the objects being accessed. Such a need arises from the fact that a subject's need to access a resource and the sensitivity (and hence the protection requirement) of the objects being accessed may change with time.

Historical Background

Work related to temporal access control has only a brief history and goes back to early 1990s. In many real-world situations, access to information and resources may have to be restricted based on time as the subject and object characteristics may change and so can the need for the subject to access the object. For example, in a hospital, the head of the hospital may need to grant the permissions related to a part-time doctor only during certain time intervals. Similarly, an external auditor may need to be given access to sensitive company data for a specific duration only.

Bertino et al.'s *Temporal Authorization Model* (TAM) is the first known access control model to support the time-based access control requirements in a discretionary access control (DAC) model [2]. TAM associates a periodicity constraint with each authorization indicating the valid time instants of the authorization. TAM also defines three derivation rules that allow an authorization to be derived based on another authorization. The TAM model, however, is limited to specifying the temporal interval for an entire authorization and does not consider the temporal characteristics of data/objects [1]. For example, there may be a need for be an authorization such as "a subject *s* is allowed to read object *o* one month after it has been created/written." Furthermore, the states of an object can change with time and access to such different object-states may need to be carefully specified. Atluri et al. propose a Temporal and Derived Data Authorization Model (TDAM) for a Web information portal [1]. An information portal mainly aims to provide access to data from different sources and hence the temporal characteristics of such data need to be properly captured in an access control policy [1]. TDAM uses a logic formula to capture time-related conditions to specify authorization rules and address the consistency issues.

Another significant work related to temporal access control can be seen within the context of the Role Based Access Control (RBAC) model. Bertino et al. proposed the Temporal Role Based Access Control Model (TRBAC) model by extending the existing RBAC model [3]. The TRBAC model supports the periodicity/interval constraints on the role enabling/disabling and uses triggers to define dependencies that may exist among basic role related events such as "*enable role*" and "*disable role*."

One limitation of the TRBAC model is that it only supports specification of a set of temporal intervals on the role enabling/disabling events. For example, the user-role and the role-permission assignments are not time-constrained in TRBAC, neither are the role activations. Joshi et al. proposed a General Temporal RBAC (GTRBAC) model to allow specification of more fine-grained temporal access control policies [5]. The GTRBAC model allows the interval and duration constraints on user-role assignment, role-permission assignment, and role enabling events. It also defines the duration and cardinality constraints on role activation. GTRBAC uses constraint enabling events to trigger the duration constraints and uses run-time requests to allow dynamic changes in the authorization states by administrators and users. GTRBAC uses triggers, first introduced in the TRBAC model, to support the dependencies among events. The other features of the GTRBAC include the temporal hybrid hierarchy and Separation of Duty (SoD) constraints.

More recently, work on access control approaches based on the location context and the integration of temporal and location based access control has emerged. Such work includes the GEO-RBAC model, the spatial-temporal access control model [4], the spatial-temporal RBAC model [7], the LRBAC model [6], and the location and time-based RBAC (LoT-RBAC) model.

Foundations

In this section, the existing time-based access control models mentioned above are reviewed in more detail, namely TAM, TDAM, the TRBAC model, and the GTRBAC model. The major features of each model are discussed below.

TAM: Temporal Authorization Model

TAM models the temporal context as a periodicity constraint. A periodicity constraint is of the form $<[begin, end], P>$; here, P represents a recurring set of intervals and the entire expression indicates every time instant in P between "begin" and "end." For example, "[1/1/1994, ∞], *Monday*" indicates every *Monday* starting 1/1/1994. The model uses the symbol ∞ in place of P to indicate all time instants. The temporal authorization in TAM associates such a periodicity constraint with a normal discretionary authorization. The authorization is valid at any time instant specified in the periodicity constraint. At any given time instant, if there are two authorization rules that try to grant and deny an operation on the same object, a conflict is said to occur. In such a case, the model uses the "*denials-take-precedence*" principle to favor negative authorizations.

It is possible that several authorizations have temporal dependencies among them. For example, suppose user u_1 grants a permission p to u_2 (authorization A_1), and u_2 further grants p to u_3 (authorization A_2). It is easy to see that A_2 can only be valid after A_1 becomes valid. Therefore, a requirement such as "u_2 is allowed to grant p to u_3 whenever he/she acquires p from u_1," can be specified as a derivation rule "A_2 WHENEVER A_1." TAM uses three such derivation rules to capture various relationships among different authorizations.

The derivation rules are of the form "[*begin, end*], *P*, $A <op>$, **A**", where, A is an authorization, **A** is a boolean expression on authorizations, and $<op>$ is one of the following: WHENEVER, ASLONGAS, UPON. **A** is true at time instant t, if **A** evaluates to *true* by substituting each authorization in it by *true* if it is valid at t, and by *false* if not valid. For example, **A** = $\neg$ (A_1 and A_2) is *true* at time t if either or both of A_1 and A_2 is *false* at time t. ([*begin, end*], *P*, A WHENEVER **A**) specifies that one can derive A for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ (here, $\Pi(P)$ is the set of all time instants in P) for which **A** is valid. ([*begin, end*], *P*, A ASLONGAS **A**) specifies that one can derive A for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ such that **A** is valid for each time instant in $\Pi(P)$ that is greater than or equal to t_b and lesser than or equal to t_e. ([*begin, end*], *P*, A UPON **A**) specifies that A holds *true* for each instants in $\Pi(P) \cap \{[t_b, t_e]\}$ if there exists an instant $t' \varepsilon \Pi(P)$ that is greater than or equal to t_b and lesser than or equal to t_e such that **A** is valid at time t' in. The difference between WHENEVER and ASLONGAS is that the former only evaluates the authorization state for a time instant while the latter evaluates the history of the authorization states in a given time interval. The

difference between ASLONGAS and UPON is that the former evaluates the authorization state for the entire time interval in the history while the latter only evaluates the authorization states at a time instant in the history.

Given a set of initial authorizations and rules, the derived authorizations may depend on the order in which the rules are evaluated. This is due to the existence of both positive and negative authorizations and the *denials-take-precedence* rule. To analyze this problem, the authors define the unique set of valid authorizations using the notion of critical set and propose a Critical Set Detection (CSD) algorithm to verify that a given set is critical [2].

TDAM: Temporal and Derived data Authorization Model
The TDAM model, as mentioned earlier, focuses on the temporal characteristics of data/objects. This is achieved by associating a formula τ instead of a simple temporal interval to each authorization. At any time instant, if τ is evaluated to be *true*, then the corresponding authorization is valid. By carefully designing the formula τ, the model can support specification of a fine-grained temporal access control policy. It is possible to use the temporal context of a single data item by making it a variable in τ. For example, assume a company maintains a database including the current and future predicted price for some goods; now, let each price of a data item d be associated with a time interval $[t_b, t_e]$ to indicate the temporal interval the specified price is predicted for. It is also possible to restrict a subject to read the current price only by specifying τ as "$t_b \le t \le t_e$" where t is the current time.

Similarly, the temporal dependency can also be indicated by τ. For example, consider the policy "a subject s is allowed to read object o one month after it has been created/written." Such a requirement can be specified by including τ as "$t \ge t_w + 1$ month" where t_w indicates the time when the object o is created/written and t is the current time.

TRBAC: Temporal Role Based Access Control Model
The TRBAC model allows the specification of temporal constraints to specify when a role can be enabled or disabled, and triggers to capture possible temporal dependencies among role events. The TRBAC model also uses periodic time expression instead of simple time interval to represent the periodicity constraint. In RBAC, the users can acquire the permissions only by activating enabled roles within a session; the model

associates the periodic time expressions with role enabling/disabling events to define periodicity constraints.

Triggers constitute an important part of the TRBAC model, and they are used to capture temporal dependencies among RBAC authorization states. A trigger simply specifies that if some events occur and/or some role status predicates are evaluated to true then another event can occur, with a possible time delay. For example, assume that a hospital requires that when a part-time doctor is on duty a part-time nurse can also be allowed to log on to help the part-time doctor. Here, the trigger "enable *part-time doctor* → enable *part-time nurse*" can be used to capture such a requirement.

Bertino et al. introduces a soft notion of *safety* to formally characterize the efficiency and practicality of the model. In essence, because of triggers and other periodicity constraints, ambiguous semantics can be generated. They propose an efficient graph based analysis technique to identify such ambiguous policies so that a unique execution model can be guaranteed by eliminating any ambiguous specification. When such a unique execution model is ensured the policy base is said to be *safe*. Conflicts could also occur because of the opposing *enable* and *disable* role events. The TRBAC model uses *denial-takes-precedence* in conjunction with *priority* to resolve such conflicts.

GTRBAC: Generalized Temporal Role Based Access Control Model
Joshi et al. have proposed the GTRBAC model by extending the TRBAC model to incorporate more comprehensive set of periodicity and duration constraints and time-based activation constraints. The GTRBAC model introduces the separate notion of *role enabling* and *role activation*, and introduces role states. An *enabled* role indicates that a valid user can activate it, whereas a *disabled* role indicates that the role is not available for use. A role in *active* state indicates that at least one user has activated it. Such a distinction makes the semantics and implementation of any RBAC compatible system much clearer. Besides events related to user-role assignment, permission-role assignment and role enabling, the GTRBAC model also includes role activation/deactivation events. The model associates the temporal constraints with every possible event in the system. In particular, it uses the periodicity constraint to constrain the validity of role

enabling events, user-role assignment events as well as the permission-role assignment events. As the role activation events are initiated by the users at their discretion, GTRBAC does not associate temporal constraints with activation events. Besides the periodic temporal constraints, GTRBAC also supports duration constraints. A duration constraint specifies how long should an event be valid for once it occurs. For example, one duration constraint could specify that once a role r has been enabled it should be in the enabled state for 2 hours. Note that the duration constraint has a non-deterministic start time and requires some other actions to initiate the start of the duration. For example, consider a duration constraint (2 hours, enable r). Here, when the "enable r" event occurs because of a trigger, the event becomes valid for 2 hours because of this constraint. At times, one may need to also enable duration or activation constraints – GTRBAC uses constraint enabling events to facilitate that.

The GTRBAC model also supports the temporal and cardinality constraints on role activation. Cardinality constraints have often been mentioned in the literature but have not been addressed much in the existing models. A cardinality constraint simply limits the number of activations (applies for assignments as well) within a given period of time. For example, GTRBAC supports limiting the total number of activations of a role or the maximum concurrent number of activations of a role in a given interval or duration. Furthermore, GTRBAC allows the activation constraint to be applied to all the activations of a role (per-role constraint) or applied to each activation of a role by a particular user (per-user constraint). The model uses the trigger framework introduced in the TRBAC model. In addition to these, the GTRBAC model also extends work on role hierarchy and constraints, and introduces hybrid hierarchy and time-based SoD constraints.

The GTRBAC model uses three conflict types (Type-1, Type-2, and Type-3) to categorize the different types of conflicting situations that may arise and provides a resolution technique using a combination of the following approaches: (i) *priority-based*, (ii) *denial-takes precedence*, and (iii) *more specific constraint takes precedence.*

Suroop et al. have recently proposed the LoTRBAC model by extending the GTRBAC model to address both time and location for security of mobile applications.

Key Applications

Temporal access control models are suitable for the applications where temporal constraints and temporal dependencies among authorizations are important protection requirements. One example is workflow systems that often have timing constraints on tasks and their dependencies. In particular, TAM is suitable for the system implementing the discretionary access control policies. TDAM is suitable for access control for data dissemination systems such as a web information portal where data have different states at different times. TRBAC and GTRBAC are suitable for large scale systems that have very fine-grained time-based access requirements.

Future Directions

With the development of networking and mobile technologies, context based access control is becoming very crucial. Time is very crucial context information, as is location. Hence, the work on temporal access control have provided a basis for looking at the intricacies of the context parameters that need to be used to restrict authorization decisions. In addition to context-based access, content-based access control is also becoming significant issues because of the growing need to dynamically identify content and make authorization decisions based on its semantics. Again, the work on temporal access control has provided a basis for capture the dynamic feature of the object content. Authorization models that capture context and content parameters using temporal access control are being pursued to develop more fine-grained access control models.

Cross-references
▶ Role Based Access Control

Recommended Reading
1. Atluri V. and Gal A. An authorization model for temporal and derived data: securing information portals. ACM Trans. Inf. Syst. Secur., 5(1):62–94, 2002.
2. Bertino E., Bettini C., Ferrari E., and Samarati P. An access control model supporting periodicity constraints and temporal reasoning. ACM Trans. Database Syst., 23(3): 231–285, 1998.
3. Bertino E., Bonatti P.A., and Ferrari E. TRBAC: a temporal role-based access control model. ACM Trans. Inf. Syst. Secur., 4(3):191–233, 2001.
4. Fu S. and Xu C.-Z. A coordinated spatio-temporal access control model for mobile computing in coalition environments. In Proc.

19th IEEE Int. Parallel and Distributed Processing Sym. – Workshop 17, vol. 18, 2005, p.289.2.

5. Joshi J.B.D., Bertino E., Latif U., and Ghafoor A. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng., 17(1):4–23, 2005.

6. Ray I., Kumar M., and Yu L. LRBAC: a location-aware role-based access control model. In Proc. 2nd Int. Conf. on Information Systems Security, 2006, pp. 147–161.

7. Ray I. and Toahchoodee M. A spatio-temporal role-based access control model. In Proc. 21st Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2007, pp. 420–431.

Temporal Aggregation

JOHANN GAMPER[1], MICHAEL BÖHLEN[1], CHRISTIAN S. JENSEN[2]
[1]Free University of Bozen-Bolzano, Bolzano, Italy
[2]Aalborg University, Aalborg, Denmark

Definition

In database management, aggregation denotes the process of consolidating or summarizing a database instance; this is typically done by creating so-called aggregation groups of elements in the argument database instance and then applying an aggregate function to each group, thus obtaining an aggregate value for each group that is then associated with each element in the group. In a relational database context, the instances are relations and the elements are tuples. Aggregation groups are then typically formed by partitioning the tuples based on the values of one or more attributes so that tuples with identical values for these attributes are assigned to the same group. An aggregate function, e.g., *sum*, *avg*, or *min*, is then applied to another attribute to obtain a single value for each group that is assigned to each tuple in the group as a value of a new attribute. Relational projection is used for eliminating detail from aggregation results.

In temporal relational aggregation, the arguments are temporal relations, and the tuples can also be grouped according to their timestamp values. In temporal grouping, groups of values from the time domain are formed. Then an argument tuple is assigned to each group that overlaps with the tuple's timestamp, this way obtaining groups of tuples. When aggregate functions are applied to the groups of tuples, a temporal relation results. Different kinds of temporal groupings are possible: instantaneous temporal aggregation where the time line is partitioned into time instants/points; moving-window (or cumulative) temporal aggregation where additionally a time period is placed around a time instant to determine the aggregation groups; and span aggregation where the time line is partitioned into user-defined time periods.

Historical Background

Aggregate functions assist with the summarization of large volumes of data, and they were introduced in early relational database management systems such as System R and INGRES. During the intensive research activities in temporal databases in the 1980s, aggregates were incorporated in temporal query languages, e.g., the Time Relational model [1], TSQL [8], TQuel [9], and a proposal by Tansel [10]. The earliest proposal aimed at the efficient processing of (instantaneous) temporal aggregates is due to Tuma [12]. Following Tuma's pioneering work, research concentrated on the development of efficient main-memory algorithms for the evaluation of instantaneous temporal aggregates as the most important form of temporal aggregation [6,7].

With the diffusion of data warehouses and OLAP, disk-based index structures for the incremental computation and maintenance of temporal aggregates were investigated by Yang and Widom [14] and extended by Zhang et al. [15] to include non-temporal range predicates. The high memory requirements of the latter approach were addressed by Tao et al. [11], and approximate solutions for temporal aggregation were proposed. More recently, Vega Lopez et al. [13] formalized temporal aggregation in a uniform framework that enables the analysis and comparison of the different forms of temporal aggregation based on various mechanisms for defining aggregation groups. In a similar vein, Böhlen et al. [3] develop a new framework that generalizes existing forms of temporal aggregation by decoupling the partitioning of the time line from the specification of the aggregation groups.

It has been observed that expressing queries on temporal databases is often difficult with SQL, in particular for aggregation. As a result, temporal query languages often include support for temporal aggregation. A recent paper [2] studies the support for temporal aggregation in different types of temporal extensions to SQL. A subset of the temporal aggregates considered in the entry are also found in non-relational query languages, e.g., τXQuery [5].

Foundations

A discrete time domain consisting of a totally ordered set of time instants/points is assumed together with an interval-based, valid-time data model, i.e., an interval timestamp is assigned to each tuple that captures the time when the corresponding fact is true in the modeled reality. As a running example, the temporal relation *CheckOut* in Fig. 1 is used, which records rentals of video tapes, e.g., customer C101 rents tape T1234 from time 1 to 3 at cost 4.

Defining Temporal Aggregation

Various forms of temporal aggregation that differ in how the temporal grouping is accomplished have been studied. In *instantaneous temporal aggregation (ITA)* the time line is partitioned into time instants and an aggregation group is associated with each time instant t that contains all tuples with a timestamp that intersects with t. Then the aggregate functions are evaluated on each group, producing a single aggregate value at each time t. Finally, identical aggregate results for consecutive time instants are coalesced into so-called constant intervals that are maximal intervals over which all result values remain constant. In some approaches, the aggregate results in the same constant interval must also have the same lineage, meaning that they are produced from the same set of argument tuples. The following query, Q_1, and its result in Fig. 2a illustrate ITA: *What is the number of tapes that have been checked out?* Without the lineage requirement, the result tuples $(1,[17,18])$ and $(1,[19,20])$ would have been coalesced into $(1,[17,20])$. While, conceptually, the time line is partitioned into time instants, which yields the most detailed result, the result tuples are consolidated so that only one tuple is reported for each constant interval. A main drawback is that the result relation is typically larger than the argument relation and can be up to twice the size of the argument relation.
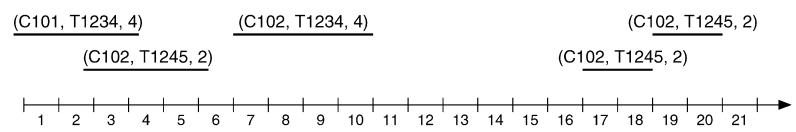
With *moving-window temporal aggregation (MWTA)* (first introduced in TSQL [8] and later also termed *cumulative temporal aggregation* [9,14]), a time window is used to determine the aggregation groups. For each time instant t, an aggregation group is defined as the set of argument tuples that hold in the interval $[t{-}w,\ t]$, where $w \geq 0$ is called a window offset. In some work [13], a pair of offsets w and w' is used, yielding a window $[t{-}w,\ t{+}w']$ for determining the aggregation groups. After computing the aggregate functions for each aggregation group, coalescing is applied similarly to how it is done for ITA to obtain result tuples over maximal time intervals. The following query, Q_2, and its result in Fig. 2b illustrate MWTA: *What is the number of tapes that have been checked out in the last three days?* To answer this query, a window is moved along the time line, computing at each time point an aggregate value over the set of tuples that are valid at some point during the last three days. While both ITA and MWTA partition the time line into time instants, the important difference is in how the aggregation groups for each time instant are defined.

Next, for *span temporal aggregation (STA)*, the time line is first partitioned into predefined intervals that are defined independently of the argument relation. For each such interval, an aggregation group is then given as the set of all argument tuples that overlap the interval. A result tuple is produced for each interval by evaluating an aggregate function over the corresponding aggregation group. The following query, Q_3, and its result in Fig. 2c illustrate STA: *What is the weekly number of tapes that have been checked out?* The time span is here defined as a period of seven days. Unlike in ITA and MWTA, in STA the timestamps of the result tuples are specified by the application and are independent of the argument data. Most approaches consider only regular time spans expressed in terms of granularities, e.g., years, months, and days.

The *multi-dimensional temporal aggregation (MDTA)* [3] extends existing approaches to temporal aggregation, by decoupling the definition of result groups and aggregation groups. A result group specifies the part of a result tuple that is independent of the actual aggregation (corresponds to the group by

CustID	TapeNum	Cost	T
C101	T1234	4	[1,3]
C102	T1245	2	[3,5]
C102	T1234	4	[7,10]
C102	T1245	2	[17,18]
C102	T1245	2	[19,20]

Temporal Aggregation. Figure 1. Tabular representation and graphical representation of temporal relation *CheckOut*.

Cnt	T
1	[1,2]
2	[3,3]
1	[4,5]
0	[6,6]
1	[7,10]
0	[11,16]
1	[17,18]
1	[19,20]

a Q_1: ITA

Cnt	T
1	[1,2]
2	[3,5]
1	[6,6]
2	[7,7]
1	[8,12]
0	[13,16]
1	[17,18]
2	[19,20]
1	[21,22]

b Q_2: MWTA

Cnt	T
3	[1,7]
1	[8,14]
2	[15,21]

c Q_3: STA

CntE	CntC	T
0	0	[1,7]
2	1	[8,14]
1	0	[15,21]
0	2	[15,21]

d Q_4: MDTA

Temporal Aggregation. Figure 2. Results of different forms of temporal aggregation.

attributes in SQL). Each result group has an associated aggregation group, namely the set of tuples from which the aggregated value is computed. In general, the grouping attributes of the tuples in an aggregation group might differ from the grouping attributes of the result group. For the specification of the result groups, two different semantics are supported: constant-interval semantics that covers ITA and MWTA and fixed-interval semantics that covers STA. The fixed-interval semantics supports the partitioning of the time line into arbitrary, possibly overlapping time intervals. The following query, Q_4, and its result in Fig. 2d illustrate some of the new features of MDTA: *For each week, list the number of expensive and the number of cheap checkouts during the preceding week?* (expensive being defined as a cost equal or greater than 4 and cheap as a cost equal or smaller than 2). The result groups are composed of a single temporal attribute that partitions the time line, the tuples in the associated aggregation groups do not have to overlap the timestamp of the result group, and two aggregates over different aggregation groups are computed for each result group.

Temporal Aggregation Processing Techniques

The efficient computation of temporal aggregation poses new challenges, most importantly the computation of the time intervals of the result tuples that depend on the argument tuples and thus are not known in advance.
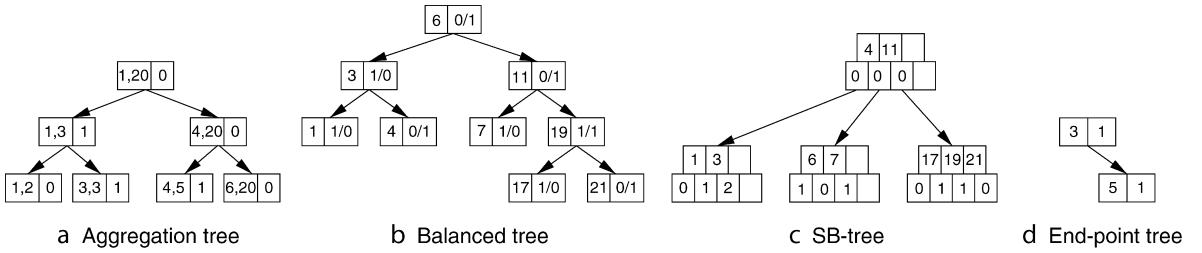
Two Scans The earliest proposal for computing ITA was presented by Tuma [12] and requires two scans of the argument relation – one for computing the constant intervals and one for computing the aggregate values over these intervals. The algorithm has a worst case running time of $O(mn)$ for m result tuples and n argument tuples.

Following Tuma's pioneering work, research concentrated on algorithms that construct main-memory data structures that allow to perform both steps at once, thus requiring only one scan of the argument relation.

Aggregation Tree The *aggregation tree* algorithm for ITA by Kline and Snodgrass [6] incrementally constructs a tree structure in main memory while scanning the argument relation. The tree stores a hierarchy of intervals and partial aggregation results. The intervals at the leaf nodes encode the constant intervals. Accumulating the partial results in a depth-first traversal of the tree yields the result tuples in chronological order. Figure 3a shows the tree for Query Q_1 after scanning the first two argument tuples. The path from the root to the leaf with time interval [3,3] yields the result tuple (2,[3,3]). The algorithm is constrained by the size of the available main memory, and it has a worst case time complexity of $O(n^2)$ for n argument tuples since the tree is not balanced.

An improvement, although with the same worst case complexity, is the k-ordered aggregation tree [6], which requires the argument tuples to be chronologically ordered to some degree. This allows to reduce the memory requirements by garbage collecting old nodes that will not be affected by any future tuples. Gao et al. [4] describe a number of parallel temporal aggregation algorithms that are all based on the aggregation tree.

Balanced Tree Moon et al. [7] propose the *balanced tree* algorithm for the main memory evaluation of ITA queries involving *sum*, *count*, and *avg*. As the argument tuples are scanned, their start and end times are stored in a balanced tree together with two values for each aggregate function being computed, namely the partial

Temporal Aggregation. Figure 3. Different forms of tree structures for temporal aggregation.

aggregate result over all tuples that start and end here, respectively. An in-order traversal of the tree combines these values to compute the result relation. Whenever a node, v, is visited, a result tuple is produced over the interval that is formed by the time point of the previously visited node and the time point immediately preceding v. Figure 3b shows the balanced tree for Query Q_1. The aggregate value of the result tuple $(2, [3,3])$ is determined as $1 + (1 - 0) = 2$. Although the balanced tree requires less memory than the aggregation tree, it is constrained by the amount of available memory. For the *min* and *max* functions a merge-sort like algorithm is proposed. Both algorithms have $O(n \log n)$ time complexity for n argument tuples. To overcome the memory limitation, a bucket algorithm is proposed, which partitiones the argument relation along the time line and keeps long-lived tuples in a meta-array. Aggregation is then performed on each bucket in isolation.

SB-Tree Yang and Widom [14] propose a disk-based index structure, the *SB-tree*, together with algorithms for the incremental computation and maintenance of ITA and MWTA queries. It combines features from the segment tree and the B-tree and stores a hierarchy of time intervals associated with partially computed aggregates. To find the value of an aggregate at a time instant t, the tree is traversed from the root to the leaf that contains t and the partial aggregate values associated with the time intervals that contain t are combined. Figure 3c shows the SB-tree for Query Q_1. The value at time 8 results from adding 0 and 1 (associated with [10,11] and [7,11], respectively). The time complexity of answering an ITA query at a single time point is $O(h)$, where h is the height of the tree, and $O(h + r)$ for retrieving the result over a time interval, where r is the number of leaves that intersect with the given time interval. The same paper extends the basic SB-tree to

compute MWTA. For a fixed window offset w, the timestamps of the argument tuples are extended by w to account for the tuples' contributions to the results at later time points. For arbitrary window offsets, a pair of SB-trees is required.

MVSB-Tree With the SB-tree, aggregate queries are always applied to an entire argument relation. The *multi-version SB-tree* (MVSB-tree) by Zhang et al. [15] tackles this problem and supports temporal aggregation coupled with non-temporal range predicates that select the tuples over which an aggregate is computed. The MVSB-tree is logically a sequence of SB-trees, one for each timestamp. The main drawbacks of this approach are: the tree might be larger than the argument relation, the range restriction is limited to a single non-timestamp attribute, and the temporal evolution of the aggregate values cannot be computed. Tao et al. [11] present two approximate solutions that address the high memory requirements of the MVSB-tree. They use an MVB-tree and a combination of B- and R-trees, respectively. These achieve linear space complexity in the size of the argument relation and logarithmic query time complexity.

MDTA Böhlen et al. [3] provide two memory-based algorithms for the evaluation of MDTA queries. The algorithm for fixed-interval semantics keeps in a *group table* the result groups that are extended with an additional column for each aggregate being computed. As the argument relation is scanned, all aggregate values to which a tuple contributes are updated. The group table contains then the result relation. The memory requirements only depend on the size of the result relation. With an index on the group table, the average runtime is $n \log m$ for n argument tuples and m result groups, the worst case being $O(nm)$ when each argument tuple contributes to each result tuple.

The algorithm for constant-interval semantics processes the argument tuples in chronological order and computes the result tuples as time proceeds. An *endpoint tree* maintains partial aggregate results that are computed over all argument tuples that are currently valid, and they are indexed by the tuples' end points. Figure 3d shows the endpoint tree for Query Q_1 after processing the first two argument tuples. When the third argument tuple is read, the result tuples $(2,[3,3])$ and $(1,[4,5])$ are generated by accumulating all partial aggregate values; the nodes are then removed from the tree. The size of the tree is determined by the maximal number of overlapping tuples, n_o. The average time complexity of the algorithm is nn_o. The worst-case complexity is $O(n^2)$, when the start and end points of all argument tuples are different and all tuples overlap.

Key Applications

Temporal aggregation is used widely in different data-intensive applications, which become more and more important with the increasing availability of huge volumes of data in many application domains, e.g., medical, environmental, scientific, or financial applications. Prominent examples of specific applications include data warehousing and stream processing. Time variance is one of four salient characteristics of a data warehouse, and there is general consensus that a data warehouse is likely to contain several years of time-referenced data. Temporal aggregation is a key operation for the analysis of such data repositories. Similarly, data streams are inherently temporal, and the computation of aggregation functions is by far the most important operation on such data. Many of the ideas, methods, and technologies from temporal aggregation have been and will be adopted for stream processing.

Future Directions

Future research work is possible in various directions. First, it may be of interest to study new forms of temporal aggregation. For example, a temporal aggregation operator that combines the best features of ITA and STA may be attractive. This operator should follow a data-driven approach that approximates the precision of ITA while allowing to limit the size of the result. Second, it is relevant to study efficient evaluation algorithms for more complex aggregate functions

beyond the five standard functions for which most research has been done so far. Third, the results obtained so far can be adapted for and applied in related fields, including spatio-temporal databases where uncertainty is inherent as well as data streaming applications.

Cross-references

▶ Bi-Temporal Indexing
▶ Query Processing (in relational databases)
▶ Temporal Coalescing
▶ Temporal Data Mining
▶ Temporal Database
▶ Temporal Query Languages
▶ Temporal Query Processing

Recommended Reading

1. Ben-Zvi J. The Time Relational Model. Ph.D. thesis, Computer Science Department, UCLA, 1982.
2. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data? In Proc. 13th Int. Symp. on Temporal Representation and Reasoning, 2006, pp. 121–136.
3. Böhlen M.H., Gamper J., and Jensen C.S. Multi-dimensional aggregation for temporal data. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 257–275.
4. Gao D., Gendrano J.A.G., Moon B., Snodgrass R.T., Park M., Huang B.C., and Rodrigue J.M. Main memory-based algorithms for efficient parallel aggregation for temporal databases. Distrib. Parallel Dat., 16(2):123–163, 2004.
5. Gao D. and Snodgrass R.T. Temporal slicing in the evaluation of XML queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 632–643.
6. Kline N. and Snodgrass R.T. Computing temporal aggregates. In Int. Conf. on Data Engineering, 1995, pp. 222–231.
7. Moon B., Vega Lopez I.F., and Immanuel V. Efficient algorithms for large-scale temporal aggregation. IEEE Trans. Knowl. Data Eng., 15(3):744–759, 2003.
8. Navathe S.B. and Ahmed R. A temporal relational model and a query language. Inf. Sci., 49(1–3):147–175, 1989.
9. Snodgrass R.T., Gomez S., and McKenzie L.E. Aggregates in the temporal query language TQuel. IEEE Trans. Knowl. Data Eng., 5(5):826–842, 1993.
10. Tansel A.U. A statistical interface to historical relational databases. In Proc. Int. Conf. on Data Engineering, 1987, pp. 538–546.
11. Tao Y., Papadias D., and Faloutsos C. Approximate temporal aggregation. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 190–201.
12. Tuma P.A. Implementing Historical Aggregates in TempIS. M.Sc. thesis, Wayne State University, 1992.
13. Vega Lopez I.F., Snodgrass R.T., and Moon B. Spatiotemporal aggregate computation: a survey. IEEE Trans. Knowl. Data Eng., 17(2):271–286, 2005.

14. Yang J. and Widom J. Incremental computation and maintenance of temporal aggregates. VLDB J., 12(3):262–283, 2003.

15. Zhang D., Markowetz A., Tsotras V., Gunopulos D., and Seeger B. Efficient computation of temporal aggregates with range predicates. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 237–245.

Temporal Algebras

Abdullah Uz Tansel
Baruch College – CUNY New York, NY, USA

Synonyms

Historical algebras; Valid-time algebras; Transaction-Time algebras; Bitemporal algebras

Definition

Temporal algebra is a generic term for an algebra defined for a data model that organizes temporal data. A temporal data model may support Valid-time (the time over which a data value is valid), Transaction-time (time when a data value is recorded in the database), or both (Bitemporal). So an algebra can be defined for each case, a Valid-time relational algebra, a Transaction-time relational algebra, or a Bitemporal relational algebra, respectively. Temporal algebras include the temporal versions of relational algebra operations in addition to new operations for manipulating temporal data like Timeslice, Rollback, Temporal Coalesce, temporal restructuring operations, and others. For a Temporal algebra, it is desirable to be closed (common algebras are closed), a consistent extension of the relational algebra and to reduce to relational algebra when only current data are considered.

Historical Background

Temporal algebraic languages first appeared as extensions to the relational algebra in early 1980s, mostly Valid–time or Transaction-time algebras, followed by Bitemporal algebras. These extensions differed according to their operations and how temporal data are represented. McKenzie and Snodgrass surveyed temporal algebras and identified desirable criteria that are needed in a temporal algebra [10]. However, some of these criteria are conflicting.

Foundations

Temporal Algebra Basics

Let T represent the time domain which has a linear order under "$\leq$". A time point (instant) is any element of T. A period is a consecutive sequence of time points. A temporal element is a set of disjoint maximal periods [6], and a temporal set is any set of time points. Any of these time constructs can be used to timestamp data values. A Bitemporal atom, $<Valid\text{-}Time$, $Transaction\text{-}time$, $Data>$ asserts that $data$ are valid during $Valid\text{-}time$ and is recorded during $Transaction\text{-}time$. Either Valid-time or Transaction-time may be omitted and the result is a Valid-time Atom, or a Transaction-time atom, respectively.

A temporal relational algebra is closely related to how the temporal data (temporal atoms) are represented, i.e., the type of timestamps used, where they are attached (relations, tuples, or attribute values), and whether temporal atoms are kept atomic or broken into their components. In other words, time specification may be explicit or implicit. This in turn determines possible evaluation (semantics) of temporal algebra expressions. There are two commonly adopted approaches: (i) Snapshot (Point or Sequenced [1]) evaluation that manipulates the snapshot relation at each time point, like Temporal Logic [6,11,17]; (ii) Traditional (Nonsequenced [1]) evaluation that manipulates the entire temporal relation much like the traditional relational algebra. It is also possible to mix these approaches. The syntax and the operations of a temporal algebra are designed to accommodate a desired type of evaluation. Moreover, specifying temporal algebra operations at an attribute level, instead of tuples keeps the tuple structure intact after the operation is executed, so preserving the rest of a temporal relation that is beyond the scope of operation applied.

Let Q(A, B, C), R(A, D) and S(A, D) be temporal relations in attribute timestamping, whose attribute values are sets of temporal atoms except attribute A which has constant values. It is possibly a temporal grouping identifier [4] (or a temporal key). For the sake of simplicity, attributes B, C, and D are assumed to have one temporal atom in each tuple, i.e., they are already flattened. Q_t stands for the snapshot of temporal relation Q at time t. Temporal Algebras generally include temporal equivalents of traditional relational algebra operations. The five basic Temporal Algebra

operations, $\cup^t$, $-^t$, π^t, σ^t, and $\times^t$ in snapshot evaluation are [2,4,5]:

- $R \cup^t S_t$ is $R_t \cup S_t$ for all t in T
- $R -^t S_t$ is $R_t - S_t$ for all t in T
- $\pi^t_{A1,A2,...,An} (R)$ is $\pi_{A1,A2,...,An} (R_t)$ for all t in T
- $\sigma^t_F (R)$ is $\sigma_F(R_t)$ for all t in T; Formula F includes traditional predicates and temporal predicates like *Before*, *After*, *Overlaps*, etc
- $R \times^t Q$ is $R_t \times Q_t$ for all t in T

An important issue in Snapshot Evaluation is the homogeneity of the temporal relations. A temporal relation is homogenous if each tuple is defined on the same time, i.e., in a tuple, all the attributes have the same time reference [6]. If attributes in a tuple have different time references then a snapshot may have null values leading to complications. Thus, a Temporal Algebra may be homogenous, depending on the relations scheme on which it is defined.

In Temporal Algebras that use traditional evaluation, $\cup^t$, $-^t$, π^t, σ^t, and $\times^t$ may be defined exactly the same as the relational algebra operations or they have temporal semantics incorporated in their definitions. The temporal set operations may specially be defined by considering the overlapping timestamps in tuples. Two temporal tuples are value equivalent if their value components are the same, but their timestamps may be different. Let {(a1,<[2/07,11/07), d1>)} and {(a1, <[6/07,8/07), d1>)} be tuples in R and S, respectively. These two tuples are value equivalent. In case of R U^t S, value equivalent tuples are combined into one if their timestamps overlap. Considering the former tuples the result is {(a1,<[2/07,11/07), d1>)}. In case of R –^{t} S, the common portion of the timestamps for the value equivalent tuples is removed from the tuples of R. For the above tuples the result is {a1, <[2/07,6/07), d1>), (a1, <[8/07,11/07), d1>)}. Existence of value equivalent tuples makes query specification more complex but, query evaluation is less costly. On the other hand, eliminating them is also more costly. Temporal Coalescing operation combines value equivalent tuples into one tuple [2]. Temporal algebras may include aggregates.

The definition of Temporal Projection (π^t) is straightforward. However, it may generate value equivalent tuples much like the traditional projection operation creates duplicate tuples. Moreover, the projection operation may also be used to discard the time of a relation if it is explicitly specified. In the case of implicit time specification, it needs to be converted to an explicit specification before applying the projection operation. The formula F in the Selection operation ($\sigma^t_F(Q)$) may include time points, the end points of periods, and periods in temporal elements, or temporal predicates like *Before*, *After*, *Overlaps*, etc. It is possible to simulate the temporal predicates by conditions referring to time points or end points of periods.

Other temporal algebra operations such as Temporal Set Intersection or Temporal Join are similarly defined. There are different versions of Temporal Join. Intersection join is computed over the common time of operand relations (see the entry on temporal joins). For instance, if {(a1, <[1/07, 5/07), b1>, <[1/07, 4/07), c1>)} is a tuple in Q, the natural join (Q ⋈ R) contains the tuple {(a1, <[1/07, 5/07), b1>, <[1/07, 4/07), c1>, <[2/07,11/07), d1>)}. If this were an intersection natural join, times of the attributes in this tuple would be restricted to their common time period [2/07, 4/07). It is also possible to define temporal outer joins [11].

Temporal algebra operations are defined independent of time granularities. However, if operand relations are defined on different time granularities, a granularity conversion is required as part of processing the operation.

Algebras for Tuple Timestamping

In tuple timestamping relations are augmented with one column to represent time points, periods or temporal elements, or two columns to represent periods. Relation Q is represented as Q1(A, B, From, To) and Q2(A, C, From, To) where From and To are the end points of periods. Similarly, R(A, D, From, To) and S(A, D, From, To) correspond to the relations R and S, respectively. The tuple of Q given above is represented as the following tuples: (a1, b1, 1/07, 5/07) in Q1 and (a1, c1, 1/07, 4/07) in Q2. For accessing time points within a period snapshot evaluation may be used [6,17] or in case of Traditional Evaluation, attributes representing the end points of periods may be specified in operations. Another path followed is to define temporal expansion and contraction operations [9]. A period is expanded to all the time points included in it by temporal expansion and temporal contraction does the opposite, converts a sequence of time points to a period. Relation instances indexed by time points are used to define a temporal algebra by Clifford, Croker, and Tuzhilin [17].

Algebras for Attribute Timestamping

Timestamps are attached to attributes and N1NF relations are used and the entire history of an object is represented as a set of temporal atoms in one tuple. These temporal relations are called temporally grouped in contrast to temporally ungrouped relations that are based on tuple timestamping [17]. Naturally, temporally grouped algebras are more expressive than temporally ungrouped algebras and the former is more complex than the latter [17]. A temporal algebra that is based on snapshot evaluation and allows set theoretic operations on temporal elements is given in [6]. For the algebra expression e, the construct [[e]] returns the time over which e's result is defined [6] and it can further be used in algebraic expressions. An algebra, based on time points and lifespans, that uses snapshot evaluation is proposed in [3,5]. The nest and unnest operations for the transformations between 1NF and N1NF relations and operations which form and break temporal atoms are included in a temporal algebra [5,12,13]. N1NF Temporal relations and their algebras may or may not be homogonous [6,12,13].

Valid-time and Transaction-time Algebras

Most of the algebras mentioned above are Valid-time Relational Algebras. A Valid-time Relational Algebra includes additionally a Slice operation ($\varsigma\tau\Theta$) that is a redundant, but very useful operation. Let R be a Valid-time Relation and t be a time point (period, temporal element, or temporal set). Then, $\varsigma\tau\Theta_t$ (R) cuts a slice from R, the values that are valid over time t and returns them as a relation [1,5,12,13]. Common usage of the Slice operation is to express the "when" predicate in natural languages. Slice may also be incorporated into the selection operation or the specification of a relation to restrict the time of a temporal relation by a temporal element, i.e., R[t] [6]. Slice may be applied at an attribute level to synchronize time of one attribute by another attribute [5,12,13]. For instance, $\varsigma\tau\Theta_{B,C}$ (Q) restricts the time of attribute B by the time of attribute C. Applying the Slice operation on all the attributes by the same time specification returns a snapshot at the specified time.

A Transaction-time relational algebra includes a Rollback operation (τ), another form of Slice for rolling back to the values recorded at a designated time. Let R be a Transaction-time relation and t be a time point (period, temporal element, or temporal set). Then, τ_t (R) cuts a slice from R, the values that were recorded over time t and returns them as a relation [1,5,8]. Note the duality between the Valid-time relational algebra and the Transaction-time relational algebra; each has the same set of operations and an appropriate version of the slice operation.

Bitemporal Relational Algebras

Bitemporal algebra operations are more complicated than temporal algebras that support one time dimension only [1,8,15]. A Bitemporal algebra includes both forms of the Slice operation in addition to other algebraic operations. A Bitemporal query has a context that may or may not imply a Rollback operation [15]. However, once a Rollback operation is applied on a Bitemporal relation, Valid-time algebra operations can be applied on the result. It is also possible to apply Bitemporal Algebra operations on a bitemporal relation before applying a Rollback operation. In this case, the entire temporal relation is the context of the algebraic operations. In coalescing Bitemporal tuples, starting with Valid-time or Transaction-time may result in different coalesced tuples [8]. For the data maintenance queries, Valid-time needs to be coalesced within the Transaction-time.

Key Applications

Use of temporal algebra includes query language design [14], temporal relational completeness [16,17], and query optimization [14]. Some Relational algebra identities directly apply to temporal relational algebra whereas other identities do not hold due to the composite representation or semantics of temporal data [10]. Naturally, identities for the new temporal algebra operations and their interaction with the operations borrowed from the relational algebra need to be explored as well [5,7].

Cross-references

► Bitemporal Interval
► Bitemporal Relation
► Nonsequenced Semantics
► Relational Algebra
► Relational Model
► Sequenced Semantics
► Snapshot Equivalence
► Temporal Aggregates
► Temporal Coalescing
► Temporal Conceptual Models
► Temporal Data Models

Recommended Reading

1. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, 2000.
2. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 180–191.
3. Clifford J. and Croker A. The historical relational data model (HRDM) and algebra based on lifespans. In Proc. 3th Int. Conf. on Data Engineering, 1987, pp. 528–537.
4. Clifford J., Croker A., and Tuzhilin A. On completeness of historical data models. ACM Trans. Database Syst., 19 (1):64–116, 1993.
5. Clifford J. and Tansel A.U. On an algebra for historical relational databases: two views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 247–265.
6. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, 1988.
7. Gadia S.K. and Nair S.S. Algebraic identities and query optimization in a parametric model for relational temporal databases. IEEE Trans. Knowl. Data Eng., 10(5):793–807, 1998.
8. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, 1994.
9. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Syst., 13(3):289–296, 1988.
10. McKenzie E. and Snodgrass R.T. Evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, 1991.
11. Soo M.D., Jensen C., and Snodgrass R.T. *1*An algebra for TSQL2. In TSQL2 Temporal Query Language, R.T. (ed.). R.T. Snodgrass (ed.). Kluwer Academic, Norwell, MA, 1995, pp. 505–546.
12. Tansel A.U. Adding time dimension to relational model and extending relational algebra. Inf. Syst., 11(4):343–355, 1986.
13. Tansel A.U. Temporal relational data model. IEEE Trans. Knowl. Database Eng., 9(3):464–479, 1997.
14. Tansel A.U., Arkun M.E., and Ozsoyoglu G. Time-by-example query language for historical databases. IEEE Trans. Softw. Eng., 15(4):464–478, 1989.
15. Tansel A.U. and Eren-Atay C. Nested bitemporal relational algebra. In Proc. 21st Int. Symp. on Computer and Information Sciences, 2006, pp. 622–633.
16. Tansel A.U. and Tin E. Expressive power of temporal relational query languages. IEEE Trans. Knowl. Data Eng., 9(1):120–134, 1997.
17. Tuzhilin A. and Clifford J. A temporal relational algebra as basis for temporal relational completeness. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 13–23.

Temporal Assignment

Temporal Association Mining

Temporal Coalescing

Michael Böhlen
Free University of Bozen-Bolzano, Bolzano, Italy

Definition

Temporal coalescing is a unary operator applicable to temporal databases that is similar to duplicate elimination in conventional databases. Temporal coalescing merges value-equivalent tuples, i.e., tuples with overlapping or adjacent timestamps and matching explicit attribute values. Tuples in a temporal relation that agree on the explicit attribute values and that have adjacent or overlapping timestamps are candidates for temporal coalescing. The result of operators may change if a relation is coalesced before applying the operator. For instance, an operator that counts the number of tuples in a relation or an operator that selects all tuples with a timestamp spanning at least 3 months are sensitive to temporal coalescing.

Historical Background

Early temporal relational models implicitly assumed that the relations were coalesced. Ben Zvi's Time Relational Model [13, Chap. 8], Clifford and Croker's Historical

Relational Data Model (HRDM) [13, Chap. 1], Navathe's Temporal Relational Model (TRM) [13, Chap. 4], and the data models defined by Gadia [13, pp. 28–66], Sadeghi [9] and Tansel [13, Chap. 7] all have this property. The term *coalesced* was coined by Snodgrass in his description of the data model underlying TQuel, which also requires temporal coalescing [10]. Later data models, such as those associated with HSQL [13, Chap. 5] and TSQL2 [11], explicitly required coalesced relations. The query languages associated with these data models generally did not include explicit constructs for temporal coalescing. HSQL is the exception; it includes a COALESCE ON clause within the select statement, and a COALESCED optional modifier immediately following SELECT [13, Chap. 5]. Some query languages that do not require coalesced relations provide constructs to explicitly specify temporal coalescing; VT-SQL [8] and ATSQL [2] are examples.

Navathe and Ahmed defined the first temporal coalescing algebraic operator; they called this COMPRESS [13, Chap. 4]. Sarda defined an operator called COALESCE [13, Chap. 5], Lorentzos' FOLD operator includes temporal coalescing [13, Chap. 3], Leung's second variant of a temporal select join operator TSJ$_2$ [13, Chap. 14] can be used to effect temporal coalescing, and TSQL2's representational algebra also included a coalesce operator [11].

In terms of performance and expressiveness Leung and Pirahesh provided a mapping of the coalesce operation into *recursive* SQL [6, p. 329]. Lorentzos and Johnson provided a translation of his FOLD operator into Quel [7, p. 295]. Böhlen et al. [3] show how to express temporal coalescing in terms of standard SQL and compare different implementations. SQL-based solutions to coalescing have also been proposed by Snodgrass [12] and Zhou et al. [14].

Foundations

Temporal databases support the recording and retrieval of time-varying information [13] and associate with each tuple in a temporal relation one or more timestamps that denote some *time periods*. The discussion assumes that each tuple is associated with a *valid time* attribute VT. This attribute is called the timestamp of the tuple. The timestamps are half open time periods: the start point is included but the end point is not. The non-timestamp attributes are referred to as the explicit attributes.

In a temporal database, tuples are uncoalesced when they have identical attribute values and their timestamps are either adjacent in time ("meet" in Allen's taxonomy [1]) or have some time in common. Consider the relations in Fig. 1. The relation records bonus payments that have been given to employees. Ron received two 2K bonuses: one for his performance from January 1981 to April 1981 and another one for his performance from May 1981 to September 1981. Pam received a 3K bonus for her performance from April 1981 to May 1981. Bonus1 is uncoalesced since the tuples for Ron have adjacent timestamps and can be coalesced. Bonus2 is coalesced. Coalescing Bonus1 yields Bonus2.

As with duplicate elimination in nontemporal databases, the result of some operators in temporal databases changes if the argument relation is coalesced before applying the operator [11]. For instance an operator that counts the number of tuples in a relation or an operator that selects all tuples with a timestamp spanning at least 3 months are sensitive to temporal coalescing.

In general, two tuples in a *valid time* relation are candidates for temporal coalescing if they have identical explicit attribute values (see *value equivalence* [10]) and have adjacent or overlapping timestamps. Such tuples can arise in many ways. For example, a projection of a coalesced temporal relation may produce an uncoalesced result, much as duplicate tuples may be produced by a duplicate preserving projection on a duplicate-free nontemporal relation. In addition, update and insertion operations may not enforce temporal coalescing, possibly due to efficiency concerns.

Bonus1

Name	Amount	VT
Ron	2K	[1981/01−1981/06)
Ron	2K	[1981/01−1981/05)
Pam	3K	[1981/05−1981/10)

Bonus2

Name	Amount	VT
Ron	2K	[1981/01−1981/10)
Pam	3K	[1981/04−1981/06)

Temporal Coalescing. Figure 1. Uncoalesced (Bonus1) and coalesced (Bonus2) valid time relations.

Thus, whether a relation is coalesced or not makes a semantic difference. In general, it is not possible to switch between a coalesced and an uncoalesced representation without changing the semantics of programs. Moreover, as frequently used database operations (projection, union, insertion, and update) may lead to potentially uncoalesced relations and because many (but not all) real world queries require coalesced relations, a fast implementation is imperative.

Temporal coalescing is potentially more expensive than duplicate elimination, which relies on an equality predicate over the attributes. Temporal coalescing also requires detecting if the timestamps of tuples overlap, which is an inequality predicate over the timestamp attribute. Most conventional DBMSs handle inequality predicates poorly; the typical strategy is to resort to exhaustive comparison when confronted with such predicates [5], yielding quadratic complexity (or worse) for this operation.

Implementing Temporal Coalescing

Temporal coalescing does not add expressive power to SQL. Assuming that time is linear, i.e., totally ordered, it is possible to compute a coalesced relation instance with a single SQL statement (see also [4, p. 291]). The basic idea is to use a join to determine the first (f) and last (l) time period of a sequence of *value equivalent* tuples with adjacent or overlapping timestamps as illustrated in Fig. 2.

The SQL code assumes that the *time period* is represented by start (S) and end (E) point, respectively. Besides start and end point there is an explicit attribute *c*. This yields a relation with schema $R(S, E, c)$. Two subqueries are used to ensure that there are no temporal gaps (for example between l and f' is a temporal gap) and that the sequence is maximal (there is no tuple with a time period that starts before the start point of f and that temporally overlaps with f; there is no tuple with a time period that ends after the end point of l and that temporally overlaps with l), respectively.

```
SELECT DISTINCT f.S, l.E, f.c
FROM r AS f, r AS l
WHERE f.S < l.E
AND f.c = l.c
AND NOT EXISTS (SELECT *
                FROM r AS m
                WHERE m.c = f.c
                AND f.S < m.S AND
                m.S < l.E
AND NOT EXISTS  (SELECT *
                FROM r AS a1
                WHERE a1.c = f.c
                AND a1.S < m.S AND m.S
                <= a1.E))
AND NOT EXISTS (SELECT *
                FROM r AS a2
                WHERE a2.c = f.c
                AND (a2.S < f.S AND f.S
                <= a2.E OR
                    a2.S <= l.E AND l.E
                    < a2.E))
```
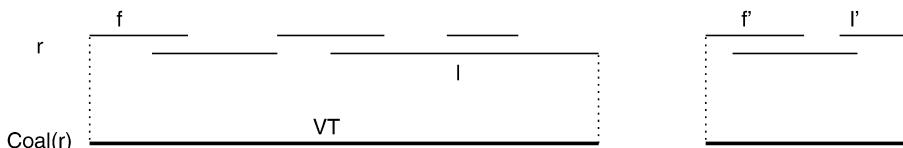
The above SQL statement effectively coalesces a relation. However, current database systems cannot evaluate this statement efficiently. It is possible to exploit the fact that only the maximal time periods are relevant. Rather than inserting a new tuple (and retaining the old ones) it is possible to update one of the tuples that was used to derive the new one. This approach can be implemented by iterating an update statement. The statement is repeated until the relation does not change anymore, i.e., until the fixpoint with respect to temporal coalescing is reached.

```
repeat
   UPDATE r l
   SET (l.E) =
     (SELECT MAX(h.E)
      FROM r h
      WHERE l.c = h.c
      AND l.S < h.S AND l.E >= h.S AND l.
      E < h.E)
```


Temporal Coalescing. Figure 2. Illustration of temporal coalescing.

```
    WHERE EXISTS (
        SELECT *
        FROM r h
        WHERE l.c = h.c
        AND l.S < h.S AND l.E >= h.S AND
        l.E < h.E)
```
until fixpoint(r)

One means to further improve the performance is to use the DBMS as an enhanced storage manager and to develop main memory algorithms on top of it. Essentially, this means to load the relation into main memory, coalesce it manually, and then store it back in the database. If tuples are fetched ordered primarily by explicit attribute values and secondarily by start points it is possible to coalesce a relation with just a single tuple in main memory. The core of the C code of the temporal coalescing algorithm is displayed below. It uses ODBC to access the database.

```
SQLAllocEnv(&henv);
SQLAllocConnect(henv, &hdbc);
SQLConnect(hdbc,"Ora10g",
SQL_NTS,"scott",SQL_NTS,"tiger",
SQL_NTS;
SQLAllocHandle(SQL_HANDLE_STMT,
hdbc, &hstmt1)
SQLAllocHandle(SQL_HANDLE_STMT,
hdbc, &hstmt2)

/* initialize buffer curr_tpl (a one
tuple buffer) */
SQLExecDirect(hstmt1, "SELECT S, E, c
FROM r ORDER BY c, S", SQL_NTS))
curr_tpl.S = next_tpl.S;
curr_tpl.E = next_tpl.E;
curr_tpl.c = next_tpl.c;

/* open a cursor to store tuples back
in the DB */
SQLPrepare(hstmt2,    "INSERT    INTO
r_coal VALUES (?,?,?)", SQL_NTS)

/* main memory temporal coalescing */
while     (SQLFetch(hstmt1)     !=
SQL_NO_DATA){ /* fetch all tuples */
   if (curr_tpl.c == next_tpl.c &&
   next_tpl.S <= curr_tpl.E){
      /* value-equivalent and
      overlapping */
```

```
      if (next_tpl.E > curr_tpl.E)
         curr_tpl.E = next_tpl.E;
   } else{
   /* not  value-equivalent  or  non-
   overlapping */
   SQLExecute(hstmt2)  /*  store  back
   current tuple */
   curr_tpl.S = next_tpl.S;
   curr_tpl.E = next_tpl.E;
   curr_tpl.c = next_tpl.c;
   }
}
SQLExecute(hstmt2) /* store back cur-
rent tuple */
```

Key Applications

Temporal coalescing defines a normal form for temporal relations and is a crucial and frequently used operator for applications that do not want to distinguish between snapshot equivalent relations. Applications that allow to distinguish between snapshot equivalent relations have temporal coalescing as an explicit operator similar to duplicate elimination in existing database systems.

Cross-references

▶ Snapshot Equivalence
▶ Temporal Database
▶ Temporal Data Model
▶ Time Domain
▶ Time Interval
▶ Time Period
▶ Valid Time

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 16(11):832–843, 1983.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):48, 2000.
3. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases. 1996, pp. 180–191.
4. Celko J. SQL for Smarties: Advanced SQL Programming. Morgan Kaufmann, 1995.
5. Leung C. and Muntz R. Query Processing for Temporal Databases. In Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 200–208.
6. Leung T.Y.C. and Pirahesh H. Querying Historical Data in IBM DB2 C/S DBMS Using Recursive SQL. In J. Clifford, A. Tuzhilin (eds.). Recent Advances in Temporal Databases, Springer, 1995.

T

T

7. Lorentzos N. and Johnson R. Extending relational algebra to manipulate temporal data. Inf. Syst., 15(3), 1988.

8. Lorentzos N.A. and Mitsopoulos Y.G. Sql extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.

9. Sadeghi R., Samson W.B., and Deen S.M. HQL – A Historical Query Language. Technical report, Dundee College of Technology, Dundee, Scotland, September 1987.

10. Snodgrass R.T. The temporal query language TQuel. ACM Trans. Database Syst., 12(2):247–298, June 1987.

11. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer Academic, Boston, 1995.

12. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 2000.

13. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. 1Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Redwood City, California, 1993.

14. Zhou X., Wang F., and Zaniolo C. Efficient Temporal Coalescing Query Support in Relational Database Systems. In Proc. 17th Int. Conf. Database and Expert Syst. Appl., 2006, pp. 676–686.

Temporal Compatibility

Michael H. Böhlen[1], Christian S. Jensen[2], Richard T. Snodgrass[3]

[1]Free University of Bozen-Bolzano, Bozen-Bolzano, Italy

[2]Aalborg University, Aalborg, Denmark

[3]University of Arizona, Tucson, AZ, USA

Definition

Temporal compatibility captures properties of temporal languages with respect to the nontemporal languages that they extend. Temporal compatibility, when satisfied, ensures a smooth migration of legacy applications from a non-temporal system to a temporal system. Temporal compatibility dictates the semantics of legacy statements and constrains the semantics of temporal extensions to these statements, as well as the language design.

Historical Background

Since the very early days of temporal database research, the compatibility with legacy languages and systems has been considered, but the first comprehensive investigation was reported by Bair et al. [2]. Compatibility issues are common for work done in the context of systems and commercial languages, such as SQL or Quel. Theoretical or logic-based approaches usually do not explore compatibility notions since they tend to strictly separate temporal from nontemporal structures.

Foundations

Motivation

Most data management applications manage time-referenced, or temporal, data. However, these applications typically run on top of relational or object-relational database management systems (DBMSs), such as DB2, Oracle, SQL Server, and MySQL, that offer only little built-in support for temporal data management. Organizations that manage temporal data may benefit from doing so using a DBMS with built-in temporal support. Indeed, it has been shown that using a temporal DBSM in place of a non-temporal DBMS may reduce the number of lines of query language code by a factor of three, with the conceptual complexity of application development decreasing even further [13].

Then, what hinders an organization from adopting a temporal DBMS? A key observation is that an organization is likely to already have a portfolio of data management applications that run against a non-temporal DBMS. In fact, the organization is likely to have made very large investments in its legacy systems, and it depends on the functioning of these systems for the day-to-day operation of its business.

It should be as easy as possible for the organization to migrate to a temporal DBMS. It would be attractive if all existing applications would simply work without modification on the temporal DBMS. This would help protect the organization's investment in its legacy applications. The opposite, that of having to rewrite all legacy applications, is a daunting proposition.

However, this type of compatibility is only the first step. The next step is to make sure that legacy applications can coexist with new applications that actually exploit the enhanced temporal support of the new DBMS. These applications may query and modify the same (legacy) tables. It should thus be possible to add a new temporal dimension to existing tables, without this affecting the legacy applications that use these tables.

Next, the organization maintains a large investment in the skill set of its IT staff. In particular, the staff is skilled at using the legacy query language, typically SQL. The new, temporal query language should leverage this investment, by making it easy for the

application programmers to write temporal queries against temporal relations.

Next four specific compatibility properties that aim to facilitate the migration from a non-temporal DBMS to a temporal DBMS are considered.

Upward Compatibility

The property of upward compatibility states that all language statements expressible in the underlying non-temporal query language must evaluate to the same result in the temporal query language, when evaluated on non-temporal data.

Figure 1 illustrates this property. In the figure, a conventional table is denoted with a rectangle. The current state of this table is the rectangle in the upper-right corner. Whenever a modification is made to this table, the previous state is discarded; hence, at any time, only the current state is available. The discarded prior states are denoted with dashed rectangles; the right-pointing arrows denote the modifications that took the table from one state to the next.

When a query q is applied to the current state of a table, a resulting table is computed, shown as the rectangle in the bottom right corner. While this figure only concerns queries over single tables, the extension to queries over multiple tables is clear.

As an example, consider a hypothetical temporal extension of the conventional query language SQL [9]. Upward compatibility states that (i) all instances of tables in SQL are instances of tables in this extension, (ii) all SQL modifications to tables result in the same tables when the modifications are evaluated according to the semantics of the extension, and (iii) all SQL queries result in the same tables when the queries are evaluated according to the extension.

By requiring that a temporal extension to SQL is a strict superset (i.e., only *adding* constructs and semantics), it is relatively easy to ensure that the extension is upward compatible with SQL. TOSQL [1], TSQL [10], HSQL [11], IXSQL [7], TempSQL [5], and TSQL2 [12] were designed to satisfy upward compatibility.

While upward compatibility is essential in ensuring a smooth transition to a new temporal DBMS, it does not address all aspects of migration. It only ensures the operation of existing legacy applications and does not address the coexistence of these with new applications that exploit the improved temporal support of the DBMS.

Temporal Upward Compatibility

The property of temporal upward compatibility (TUC) addresses the coexistence of legacy and new applications. Assume an existing or new application needs support for the temporal dimension of the data in one or more of the existing tables that record only the current state. This is best achieved by changing the snapshot table to become a temporal table. It is undesirable to be forced to change the application code that accesses the snapshot table when that table is made temporal. TUC states that conventional queries on temporal data yield the same results as do the same queries on a conventional database formed by taking a timeslice at "now." TUC applies also to modifications, views, assertions, and constraints [2].

Temporal upward compatibility is illustrated in Fig. 2. When temporal support is added to a table, the history is preserved and modifications over time are retained. In the figure, the rightmost dashed state was the current state when the table was made temporal. All subsequent modifications, denoted again by

Temporal Compatibility. Figure 1. Upward compatible queries.

Temporal Compatibility. Figure 2. Temporal upward compatibility.

arrows, result in states that are retained, and thus are represented by solid rectangles. Temporal upward compatibility ensures that the states will have identical contents to those states resulting from modifications of the snapshot table. The query q is a conventional SQL query. Due to temporal upward compatibility, the semantics of this query must not change if it is applied to a temporal table. Hence, the query only applies to the current state, and a snapshot table results.

Most temporal languages were not designed with TUC in mind, and TOSQL [1], TSQL [10], HSQL [11], IXSQL [7], and TSQL2 [12] do not satisfy TUC. The same holds for temporal logics [4]. TempSQL [5] introduces a concept of different types of users, classical and system user. TempSQL satisfies TUC for classical users. ATSQL [3] was designed to satisfy TUC.

Snapshot Reducibility

This third property states that for each conventional query, there is a corresponding temporal query that, when applied to a temporal relation, yields the same result as the original snapshot query when applied separately to every snapshot state of the temporal relation.

Graphically, snapshot reducibility implies that for all conventional query expressions q in the snapshot model, there must exist a temporal query q^t in the temporal model so that for all db^t and for all c, the commutativity diagram shown in Fig. 3 holds.

This property requires that each query q (or operator) in the snapshot model has a counterpart q^t in the temporal model that is snapshot reducible with respect to the original query q. Observe that q^t being snapshot reducible with respect to q poses no syntactical restrictions on q^t. It is thus possible for q^t to be quite different from q, and q^t might be very involved. This is undesirable: the temporal model should be a straightforward extension of the snapshot model.

Most languages satisfy snapshot reducibility, but only because corresponding non-temporal and temporal statements do not have to be syntactically similar. This allows the languages to formulate for each nontemporal statement a snapshot reducible temporal statement, possibly a very different and complex statement.
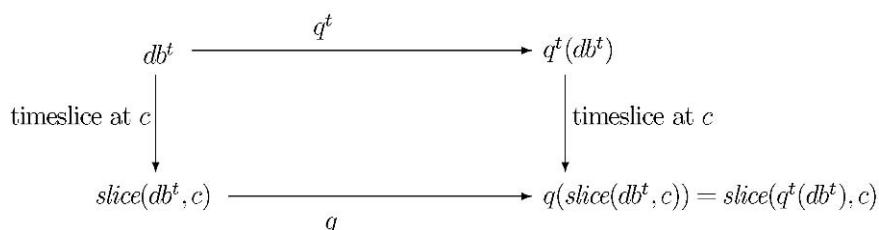
Sequenced Semantics

This property addresses the shortcoming of snapshot reducibility: it requires that q^t and q be syntactically identical, modulo an added string.
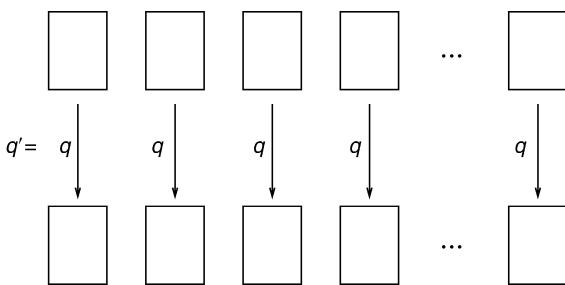
Figure 4 illustrates this property. This figure depicts a temporal query, q', that, when applied to a temporal table (the sequence of values across the top of the figure), results in a temporal table, which is the sequence of values across the bottom.

The goal is to ensure that an application programmer who is familiar with the conventional query language is able to easily formulate temporal generalizations of conventional queries using the temporal query language. This is achieved if a query q can be made temporal by simply adding a string to it. The syntactical similarity requirement of sequenced semantics makes this possible. Specifically, the meaning of q' is precisely that of applying the analogous *non-temporal* query q on each value of the argument table (which must be temporal), producing a state of the result table for each such application.

Most temporal languages do not offer sequenced semantics. As an exception, ATSQL [3] prepends the modifier SEQUENCED, together with the time dimension (valid time or transaction time, or both), to non-temporal statements to obtain their snapshot reducible generalizations. Temporal Logic [4] satisfies sequenced semantics as well: the original nontemporal statement yields sequenced semantics when evaluated over a corresponding temporal relation.

Temporal Compatibility. Figure 3. Snapshot reducibility.

Temporal Compatibility. Figure 4. Sequenced semantics.

Key Applications

Temporal compatibility properties such as those covered here are important for the adoption of temporal database technology in practice. The properties are important because temporal technology is likely to most often be applied in settings where substantial investments have already been made in database management staff and applications. The properties aim at facilitating the introduction of temporal database technology in such settings.

Properties such as these are easily as crucial for the successful adoption of temporal database technology as is highly sophisticated support for the querying of time-referenced data.

Given the very significant decrease in code size and complexity for temporal applications that temporal database technology offers, it is hoped that other DBMS vendors will take Oracle's lead and incorporate support for temporal databases into their products.

Future Directions

Further studies of compatibility properties are in order. For example, note that temporal upward compatibility addresses the case where existing tables are snapshot tables that record only the current state. However, in many cases, the existing tables may already record temporal data using a variety of ad-hoc formats. The challenge is then how to migrate such tables to real temporal tables while maintaining compatibilities.

Next, it is felt that much could be learned from conducting actual case studies of the migration of real-world legacy applications to a temporal DBMS.

Cross-references

▶ Period-Stamped Temporal Models
▶ Point-Stamped Temporal Models
▶ Sequenced Semantics
▶ Snapshot Equivalence
▶ Temporal Database
▶ Temporal Data Models
▶ Temporal Element
▶ Temporal Query Languages
▶ Time Domain
▶ Time Interval
▶ Time Period
▶ Valid Time

Recommended Reading

1. Ariav G. A temporally oriented data model. ACM Trans. Database Syst., 11(4):499–527, December 1986.
2. Bair J., Böhlen M., Jensen C.S., and Snodgrass R.T. Notions of upward compatibility of temporal query languages. Bus. Inform. (Wirtschafts Informatik), 39(1):25–34, February 1997.
3. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, December 2000.
4. Chomicki J., Toman D., and Böhlen M.H. Querying ATSQL databases with temporal logic. ACM Trans. Database Syst., 26(2):145–178, June 2001.
5. Gadia S.K. and Nair S.S. Temporal databases: A prelude to parametric data. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 28–66.
6. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December1994.
7. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.
8. McKenzie E. and Snodgrass R.T. An evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, December 1991.
9. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, USA, 1993.
10. Navathe S. and Ahmed R. Temporal extensions to the relational model and SQL. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 92–109.
11. Sarda N. HSQL: A historical query language. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 110–140.
12. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer, Boston, USA, 1995.
13. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, USA, July 1999.

T

Temporal Conceptual Models

Vijay Khatri
Indiana University, Bloomington, IN, USA

Definition

A *conceptual model* provides a notation and formalism that can be used to construct a high-level, implementation-independent description of selected aspects of the "real world," termed a miniworld. This process is called conceptual modeling, and the resulting description is referred to as a *conceptual schema*. Conceptual modeling is an important part of systems analysis and design. A *temporal conceptual model* provides a notation and formalism with built-in support for capturing temporal aspects of a miniworld during conceptual design.

Historical Background

Temporal applications need to represent data semantics not only related to "what" is important for the application, but also related to "when" it is important. The history of temporal conceptual models can be viewed in terms of two generations. The *first generation temporal conceptual models*, e.g., [2,14], provide support for only *user-defined time*; see Fig. 1a for an example. In contrast to the first generation, the *second generation temporal conceptual models*, e.g., [5,9,16], provide varying degree of support for temporal aspects; see Fig. 1b for an example. Because the first generation temporal conceptual models provide support for representation of only user-defined time, they may be thought of as "almost" time-agnostic conceptual models; on the other hand, second generation temporal conceptual models that support temporal semantics, e.g., event, state, valid time, transaction time, may be construed as time-aware conceptual models.

Foundations
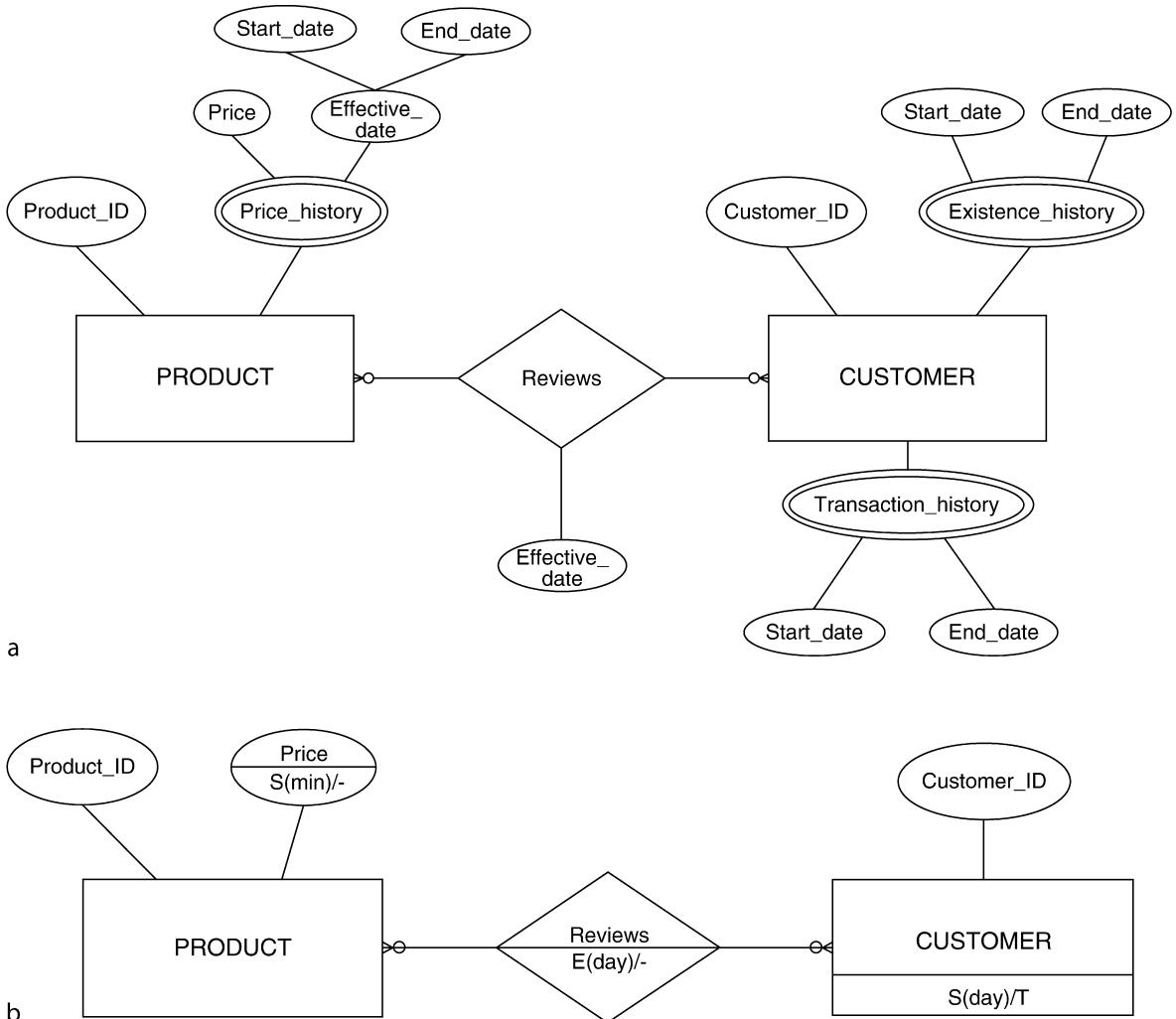
To highlight core concepts developed in the research related to temporal conceptual models, a framework of linguists is adopted that studies symbols with respect to three dimensions: syntactics, semantics and pragmatics [13]. The *syntactics* dimension includes formal relation between symbols, the *semantics* dimension involves the study of symbols in relation to the designatum (i.e., what the sign refers to) and the *pragmatics* dimension includes the relation between symbols and the interpreter.

In the following, a motivating example is employed to differentiate between first and second generation temporal conceptual models. Core concepts related to temporal conceptual models are described using syntactics, semantics and pragmatics.

Motivating Example

Figure 1a (first generation temporal conceptual model) provides an example of an ER schema that requires preserving the history of "prices" (of PRODUCT). Additionally, there is another entity type, CUSTOMER, whose existence needs to be modeled; the existence history needs to take into account both when the customer exists in the real world (Existence_history) and when the customer was added to the database (Transaction_history). Further, a CUSTOMER "reviews" PRODUCTs and the Effective_date on which the PRODUCT was reviewed needs to be captured as well. Because the first generation conceptual models do not provide a mechanism to represent temporal concepts, e.g., valid time, transaction time, event and state, these are all represented using only user-defined time. For example, the schema cannot differentiate Existence_history from Transaction_history, which are both represented simply as multi-valued attributes (double-lined ellipse). Additionally, the database analyst needs to make ad-hoc decisions related to granularity of a user-defined attribute such as Transaction_history. Start_date during implementation. As a result of the lack of a mechanism for directly mapping the miniworld to its representation, database designers are left to discover, design, and implement the temporal concepts in an ad-hoc manner.

The second generation temporal conceptual schema, referred to as the ST USM (geoSpatio-Temporal Unifying Semantic Model) schema [9] shown in Fig. 1b employs a textual string to represent temporal semantics. For example, "Price" is associated with valid time, which is represented as state ("S") with granularity of "min"(ute); further, the transaction time related to price is not relevant ("–"). The temporal semantics associated with "Price" are therefore represented by a textual string of valid time state ("S"), followed by a slash ("/"), followed by the specification of transaction time ("–"): "S (min)/–". Because both the existence (or valid) time and transaction time need to be recorded for the entity type, CUSTOMER, the annotation string for CUSTOMER is specified as "S(day)/T". Note that the granularity of transaction time is not specified

Temporal Conceptual Models. Figure 1. Examples of the two generations of temporal conceptual models.

because it is system-defined. A CUSTOMER "reviews" a PRODUCT at a certain point in time (event, E), captured to the granularity of day ("E (day)/–").

In summary, while the first generation temporal conceptual models provide a mechanism to represent only user-defined time, the second generation temporal conceptual models provide a mechanism to represent temporal data semantics. The readers are referred to Gregersen and Jensen [6] for a survey on various temporal conceptual models of the second generation.

Syntactics
Prior research has employed both graphical (see, for example, the TimeER Model [6]) and textual (see, for example, the Temporal Entity Relationship Model,

TERM [11]) syntax to represent the temporal data semantics.

While some of graphical temporal conceptual models have changed the semantics of the constructs of conventional conceptual models (see, for example, [2]), others have proposed a new formalism for representing temporal aspects. The Temporal EER (TEER) Model [3] gave *new* meaning to *extant* ER modeling constructs such as the entity type, the attribute and the relationship; for example, each entity of an entity type is associated with a temporal element that represents the lifespan of the entity, i.e., the semantics of an entity type in a conventional conceptual model was changed. On the other hand, most of the graphical temporal conceptual models propose *new* constructs that represent the temporal aspects.

Prior research has employed two ways to graphically represent temporal aspects using new constructs; they are referred to as augmented (see, for example, the TimeER Model [6] and ST USM [9]) and standalone (see, for example, the Relationships, Attributes, Keys and Entities (RAKE) Model [4]). The *augmented approaches* construe second generation conceptual schemas as "constrained" first generation schemas. For example, ST USM employs a "shorthand" for temporal semantics that is represented as annotations (see, for example, Fig. 1b); however, the semantics of a second generation schema (ST USM schema) can be "unwrapped" using a first generation schema (USM schema) and a set of constraints. The readers are referred to [9] for examples of and procedure for "unwrapping" of the semantics of an annotated schema. In contrast, the *standalone approaches* suggest *new* constructs for representing the temporal aspects. The *augmented approaches* provide a mechanism for capturing temporal data semantics at the second level of abstraction; such approaches *deliberately* defer elicitation of the temporal data semantics ("when") from the first level of abstraction that focuses on "what" is important for the application. In contrast, the *standalone approaches* provide a single level of abstraction for representing both "what" and "when."

Having outlined different syntax adopted by various conceptual models, the temporal semantics that need to be captured in a temporal conceptual model are described next.

Semantics

Based on [8], definitions of different temporal aspects that need to be represented in a temporal conceptual model are outlined below.

An *event* occurs at a point in time, that is, it has no duration (for example, a special promotion for a product is scheduled on Christmas Eve this year (2007–12–24)), while a *state* has duration (for example, a certain price for a product is valid from 5:07 P.M. on 2005–11–11 to 5:46 P.M. on 2007–1–11).

Facts can interact with time in two orthogonal ways, resulting in transaction time and valid time. *Transaction time* links a fact to the time that it becomes current in the database, and implies the storage of versions of data. The data semantics of transaction time associated with a fact require that the fact can exist in certain time periods in the past until *now* (state). *Valid time* is used to record the time at which a particular fact is true in the real world and implies the storage of histories related to facts. The data semantics of valid time associated with a fact imply that the fact can exist at certain points in time (events) or in certain time periods (states), in the past, the present, or the future.
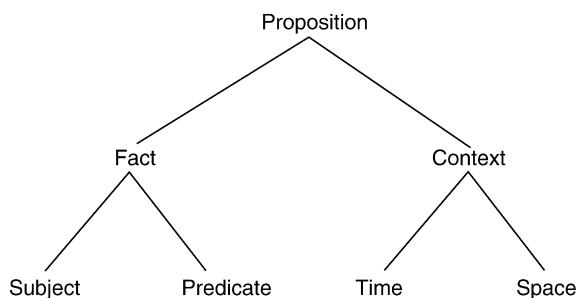
Granularities, which are intrinsic to temporal data, provide a mechanism to hide details that are not known or not pertinent for an application. Day, minute, and second are examples of temporal granularities related to the Gregorian calendar. The price history for a manufacturing application may, for example, be associated with a temporal granularity of "day," while the representation of price history for a stock market application may require a temporal granularity of "minute" or even "second."

Pragmatics

Prior research suggests that "effective exchange of information between people and machines is easier if the data structures that are used to organize the information in the machine correspond in a natural way to the conceptual structures people use to organize the same information" [12]. Three criteria play a role in how an "interpreter" (users) interacts with "symbols" (conceptual schema): (i) "internal" representation; (ii) snapshot reducibility; (iii) upward compatibility. While snapshot reducibility and upward compatibility may be rooted in syntactics and semantics, they affect the pragmatic goal, comprehension.

Internal Representation All human knowledge is stored as abstract conceptual propositions. Based on propositions, Anderson and Bower's [1] Human Associative Model (HAM) represents information in the long-term memory as shown in Fig. 2. A *proposition*

Temporal Conceptual Models. Figure 2. Internal representation of human knowledge. (Adapted from HAM Model [1].)

is an assertion about the real world that is composed of a *fact* and *context* (associated with the fact). A *subject* and *predicate* correspond with a topic and a comment about the topic. For some applications, the context in which the fact is true can be the key to reasoning about the miniworld. This context in turn, is composed of *time* and *location* associated with the fact. Note that the "context" element is orthogonal to the "fact" element and specifies the temporal reality for which the fact is true. (This entry does not cover spatial aspects; see [9] for details on a spatio-temporal conceptual model.) An augmented approach that segregates "what" from "when" corresponds with the way humans naturally organize temporal information and should, thus, support comprehension of the schema.

Snapshot Reducibility Snapshot reducibility implies a "natural" generalization of the syntax and semantics of extant conventional conceptual models, e.g., the ER Model [2], for incorporating the temporal extension. Snapshot reducibility ensures that the semantics of a temporal model are understandable "in terms of" the semantics of the conventional conceptual model. Here, the overall objective is to help ensure minimum additional investment in a database analyst training.

For example, in a "conventional" conceptual model a *key attribute* uniquely identifies an entity (at a point in time). A *temporal key* implies uniqueness at *each point in time*. As may be evident, the semantics of a temporal key here are implied by the semantics of a key in a "conventional" conceptual model.

Upward Compatibility Upward compatibility refers to the ability to render a conventional conceptual schema temporal without impacting or negating that legacy schema, thus, protecting investments in the existing schemas. It also implies that both the legacy schemas and the temporal schemas can co-exist. Upward compatibility requires that the syntax and

Temporal Conceptual Models. Table 1. Summary of a sample of first and second generation temporal conceptual models

		Syntactics	Semantics			Pragmatics		
	Syntax	User-defined time	Valid time and transaction time	Event and state	Granularity	Consideration of internal representation	Upward compatibility	Snapshot reducibility
First Generation								
ER Model [2]	• "What": Graphical • "When": NA	Yes	No	No	No	NA	NA	NA
Second Generation								
TERM [11]	• "What": Textual • "When": Textual	Yes	Valid time only	Both	Yes	No	No	No
TERC+ [16]	• "What": Graphical • "When": Graphical	Yes	Valid time only	Both	No	No	Yes	Yes
TimeER [5]	• "What": Graphical • "When": Textual	Yes	Both	Both	No	No	Yes	Yes
ST USM [9]	• "What": Graphical • "When": Textual	Yes	Both	Both	Yes	Yes	Yes	Yes

semantics of the traditional conceptual model remain unaltered. An augmented approach that extends conventional conceptual models would ensure upward compatibility.

Summary

Because one of the important roles of conceptual modeling is to support user-database analyst interaction, the linguistics-based framework of evaluation is broader: it not only includes syntactics and semantics but also includes cognitive aspects in conceptual design (pragmatics). Table 1 summarizes the evaluation of a first generation and a few second generation temporal conceptual models.

Key Applications

There are several applications of this research, both for researchers and practitioners. (i) A temporal conceptual model can help support elicitation and representation of temporal data semantics during conceptual design. (ii) A temporal conceptual schema can, thus, be the basis for the logical schema and the database. (iii) A temporal conceptual modeling approach can be used as the basis for developing a design-support environment. Such a design support environment can be integrated with tools such as ERWin. (http://www.ca.com/us/products/product.aspx?id=260)

Future Directions

Future research should explore how the temporal schema can be used as the canonical model for information integration of distributed temporal databases. A temporal conceptual model should also be extended to incorporate schema versioning.

While an initial user study has been conducted [10], future research should further evaluate temporal conceptual modeling using, e.g., protocol analysis. Studies that address *how* problem solving occurs focus on "opening up the black box" that lies between problem-solving inputs and outputs; that is, such studies investigate what happens during individual problem solving (*isomorphic approach*) rather than simply observing the effects of certain stimuli averaged over a number of cases, as in traditional studies (*paramorphic approach*) [7]. The most common approach to opening up the black box is to examine the characteristics of the problem-solving process using protocol analysis.

Experimental Results

To evaluate the augmented temporal conceptual design approach, a user experiment [10] views conceptual schema comprehension in terms of matching the *external problem representation* (i.e., conceptual schema) with *internal task representation*, based on the theory of HAM and the theory of cognitive fit [15]. The study suggests that the similarity between annotated schemas (external representation) and the HAM model of internal memory results in cognitive fit, thus, facilitating comprehension of the schemas.

Cross-references

▶ Now in Temporal Databases
▶ Schema Versioning
▶ Sequenced Semantics
▶ Supporting Transaction Time Databases
▶ Temporal Data Models
▶ Temporal Granularity

Recommended Reading

1. Anderson J.R. and Bower G.H. Human Associative Memory. Washington, D.C.: V. H. Winston & Sons, 1973.
2. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
3. Elmasri R., Wuu G., and Kouramajian V. A temporal model and query language for EER databases. In Temporal Databases: Theory, Design and Implementation, A. Tansel (ed.). Benjamin/Cummings, Menlo Park, CA, 1993, pp. 212–229.
4. Ferg S. Modeling the time dimension in an entity-relationship diagram. In Proc. 4th Int. Conf. on Entity-Relationship Approach, 1985, pp. 280–286.
5. Gregersen H. and Jensen C. Conceptual Modeling of Time-Varying Information. TIMECENTER Technical Report TR-35, September 10, 1998.
6. Gregersen H. and Jensen C.S. Temporal entity-relationship models-A survey. IEEE Trans. Knowl. Data Eng., 11(3):464–497, 1999.
7. Hoffman P.J. The paramorphic representation of clinical judgment. Psychol. Bull., 57(2):116–131, 1960.
8. Jensen C.S., Dyreson C.E., Bohlen M., Clifford J., Elmasri R., Gadia S.K., Grandi F., Hayes P., Jajodia S., Kafer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peresi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R., and Wiederhold G. A consensus glossary of temporal database concepts – February 1998 Version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). Springer Berlin, 1998.
9. Khatri V., Ram S., and Snodgrass R.T. Augmenting a conceptual model with geospatiotemporal annotations. IEEE Trans. Knowl. Data Eng., 16(11):1324–1338, 2004.
10. Khatri V., Vessey I., Ram S., and Ramesh V. Cognitive fit between conceptual schemas and internal problem representations: the

case of geospatio-temporal conceptual schema comprehension. IEEE Trans. Profession. Commun., 49(2):109–127, 2006.

11. Klopprogge M.R. TERM: an approach to include the time dimension in the entity relationship model. In Proc. 2nd Int. Conf. on Entity-Relationship Approach, 1981, pp. 473–508.

12. Moens M. and Steedman M. Temporal ontology and temporal reference. Comput. Linguist., 14(2):15–28, 1988.

13. Morris C.W. Foundations of the theory of signs. In Int. Encyclopedia of Unified Science, vol. 1, 2nd edn. University of Chicago Press, 1955.

14. Ram S. Intelligent database design using the unifying semantic model. Inform. Manage., 29(4):191–206, 1995.

15. Vessey I. Cognitive fit: a theory-based analysis of graphs vs. tables literature, Decision Sci., 22(2):219–240, 1991.

16. Zimanyi E., Parent C., Spaccapietra S., and Pirotte A. TERC+: a temporal conceptual model. In Proc. Int. Symp. Digital Media Information Base, 1997.

Temporal Constraints

PETER REVESZ
University of Nebraska-Lincoln, Lincoln, NE, USA

Definition

Temporal Constraints describe relationships among variables that refer somehow to time. A set of temporal constraints can be stored in a temporal database, which is queried by temporal queries during problem solving. For example, a set of temporal constraints may form some requirements, all of which must be satisfied during some scheduling problem.

Most interesting temporal constraints derive from references to time in natural language. Such references typically compare two *time points*, two *sets of time points*, or two *time intervals*. The literature on temporal constraints and this entry focuses on the study of these types of comparative or *binary* constraints.

Historical Background

The seminal work on temporal intervals is by Allen [1]. Difference Bounded Matrices (see the Section on Scientific Fundamentals) were introduced by Dill [3]. A graph representation of difference constraints and efficient constraint satisfaction problem-based solutions for consistency of difference constraints were presented by Dechter et al. [2]. A graph representation of gap-order constraints and an efficient algebra on them is presented by Revesz [11]. A graph representation of set order constraints and algebra on them is described in [12]. Addition constraints are considered in [12].

Temporal Constraints. Table 1. Consistency for conjunctions of temporal constraints

Case	Temporal constraints	Integers	Rationals
1.	After, Before, Equal, Inequal, AoE, BoE	$O(v + e)$	$O(v + e)$
2.	Addition	$O(ve)$	$O(v + e)$
3.	Inequal, Difference	NP-complete	$O(v^3)$

The complexity of deciding the consistency of conjunctions of integer addition constraints in Table 1 is from [9].

Periodicity constraint within query languages are considered by Kabanza et al. [4] and Toman and Chomicki [15]. Constraint databases [12,8] were introduced by Kanellakis et al. [5] with a general framework for constraints that includes temporal constraints. Indefinite temporal constraint databases were introduced by Koubarakis [6]. Linear cardinality constraints on sets were considered by Kuncak et al. [7] and Revesz [13].

Deciding the consistency of conjunctions of rational (or real) difference and inequality constraints was proven tractable by Koubarakis, but the $O(v^3)$ complexity result in Table 1 is from Péron and Halbwachs [10]. The NP-completeness result in Table 1 follows from [14].

Foundations

Temporal constraints on time points express temporal relationships between two time points, which are called also *time instances*. More precisely, let x and y be integer or rational variables or constants representing time points, and let b be an integer constant. Then some common temporal constraints on time points include the following:

$$
\begin{aligned}
&After: & x > y \\
&Before: & x < y \\
&Equal: & x = y \\
&Inequal: & x \neq y \\
&After\ or\ Equal\ (AoE): & x \geq y \\
&Before\ or\ Equal\ (BoE): & x \leq y \\
&After\ by\ at\ least\ b\ (Gap-Order): \\
&\qquad x - y \geq b\ \ \text{where}\ \ b \geq 0 \\
&Difference: & x - y \geq b \\
&Potential: & x - y \leq b \\
&Addition: & \pm x \pm y \geq b
\end{aligned}
$$

In the above table, the first six constraints are called *qualitative* constraints, and the last four constraints are called *metric* constraints because they involve a measure b of time units. For example, "a copyright form needs to be filled out before publication" can be expressed as $t_{copyright} < t_{publication}$, where $t_{copyright}$ is the time point when the copyright form is filled out and $t_{publication}$ is the time point when the paper is printed. This constraint could be just one of the requirements in a complex scheduling problem, for example, the process of publishing in a book a collection of research papers. Another temporal constraint may express that "the publication must be at least 30 days after the time of submission," which can be expressed by the constraint $t_{publication} - t_{submission} \geq 30$.

Temporal constraints on sets of time points express temporal relationships between two sets of time points. The domain of a set X is usually assumed to be the finite and infinite subsets of the integers. Common temporal constraints between sets of time points include the following:

$$
\begin{aligned}
&Equal: &&X = Y \\
&Inequal: &&X \neq Y \\
&Contains \ (Set \ Order): &&X \subseteq Y \\
&Disjoint: &&X \cap Y = \emptyset \\
&Overlap \ with \ b \ elements: && \\
&&&|X \cap Y| = b
\end{aligned}
$$

where X and Y are set variables or constants, $\emptyset$ is the empty set, b is an integer constant, and $||$ is the *cardinality* operator. For example, "The Database Systems class and the Geographic Information Systems class cannot be at the same time" can be expressed as $T_{Database} \cap T_{GIS} = \emptyset$, where $T_{Database}$ is the set of time points (measured in hour units) the Database System class meets, and T_{GIS} is the set of time points the Geographic Information Systems class meets.

Temporal constraints on time intervals express temporal relationships between two time intervals. These types of temporal constraints are known as *Allen's Relations* [Allen's Relations] because they were studied first by J. F. Allen [1].

Other types of temporal constraints: The various other types of temporal constraints can be grouped as follows:

- *n-ary temporal constraints.* These generalize the binary temporal constraints to n number of variables that refer to time. While *temporal databases* typically use binary constraints, *constraint databases* [5] use n-ary constraints [Constraint Databases], for example linear and polynomial constraints on n time point variables.

- *Temporal periodicity constraints.* Periodicity constraints [4,15] occur in natural language in phrases such as "every Monday." These constraints are discussed separately in [Temporal Periodicity Constraints].

- *Indefinite temporal constraints.* The nature of a temporal constraint can be two types: definite and indefinite. Definite constraints describe events precisely, while indefinite constraints describe events less precisely leaving several possibilities. For example, "Ed had fever between 1 P.M. and 9 P.M. but at no other times" is a definite constraint because it relays each time instance whether Ed had a fever or not. On the other hand, "Ed had fever for some time during 1 P.M. and 9 P.M." is an indefinite constraint because it allows the possibility that Ed had fever at 5 P.M. and at no other times, or another possibility that he had fever between 1 P.M. and 4 P.M. and at no other times. Hence it does not relay whether Ed had fever at 5 P.M. Conjunctions of temporal constraints can be represented in a number of ways. Consider a scheduling problem where one needs to schedule the events $e_1,...,e_6$. Suppose that there are some scheduling requirements of the form "some (second) event occurs after another (first) event by at least b days." For example, each row of the following table, which is a temporal database relation, represents one such scheduling constraint.

Scheduling_Requirements

Second_Event	First_Event	After_By
0	e_1	5
0	e_2	2
e_1	e_3	−2
e_1	e_5	−9
e_2	e_1	−6
e_3	e_2	3
e_3	e_4	−3
e_4	e_3	−5
e_5	e_4	3
e_5	e_6	3
e_6	0	1

Many queries are easier to evaluate on some alternative representation of the above temporal database relation. Some alternative representations that may be used within a temporal database system are given below.

Conjunctions of Constraints

Let $x_1,...,x_6$ represent, respectively, the times when events $e_1,...,e_6$ occur. Then the *Scheduling_Requirements* relation can be represented also by the following conjunction of difference constraints:

$$0 - x_1 \geq 5,$$
$$0 - x_2 \geq 2,$$
$$x_1 - x_3 \geq -2, x_1 - x_5 \geq -9, x_2 - x_1 \geq -6, x_3 - x_2 \geq 3, x_3 - x_4 \geq -3, x_4 - x_3 \geq -5, x_5 - x_4 \geq 3,$$
$$x_5 - x_6 \geq 3,$$
$$x_6 - 0 \geq 1$$

Labeled Directed Graphs

In general, the graph contains $n + 1$ vertices representing all the n variables and 0. For each difference constraint of the form $x_i - x_j \geq b$, the graph contains also a directed edge from the vertex representing x_j to the vertex representing x_i. The directed edge is labeled by b.

Difference Bound Matrices

Conjunctions of difference constraints can be represented also by *difference bound matrices (DBMs)* of size $(n + 1) \times (n + 1)$, where n is the number of variables. For each difference constraint of the form $x_i - x_j \geq b$, the DBM contains the value b in its (j, i)th entry. The default value is $-\infty$. For example, above set of difference constraints can be represented by the following DBM:

	0	x_1	x_2	x_3	x_4	x_5	x_6
0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1
x_1	5	$-\infty$	-6	$-\infty$	$-\infty$	$-\infty$	$-\infty$
x_2	2	$-\infty$	$-\infty$	3	$-\infty$	$-\infty$	$-\infty$
x_3	$-\infty$	-2	$-\infty$	$-\infty$	-5	$-\infty$	$-\infty$
x_4	$-\infty$	$-\infty$	$-\infty$	-3	$-\infty$	3	$-\infty$
x_5	$-\infty$	-9	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
x_6	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	3	$-\infty$

A major question about temporal constraint formulas is whether they are *consistent* or *satisfiable*. A formula is consistent or satisfiable if and only if it has at least one substitution for the variables that makes the formula true. Otherwise, it is called *inconsistent* or *unsatisfiable*. For example, if the conjunction of difference constraints that describe the *Scheduling_Requirements* table is inconsistent, then there is no schedule of the events $e_1,...,e_6$ such that all the requirements are satisfied.

Table 1 summarizes some computational complexity results, in terms of v the number of vertices and e the number of edges in the graph representation.

Most complexity results translate deciding the consistency to classical problems on graphs with efficient and well-known solutions. Decher et al. [2] provides a translation to constraint satisfaction problems, which use efficient search heuristics for large sets of constraints. Many operations on DBMs can be defined. These operators include *conjunction* or *merge* of DBMs, *variable elimination* or *projection* of a variable from a DBM, testing *implication* of a DBM by a disjunction of DBMs, and *transitive closure* of a DBM [12].

Temporal Constraints on Time Intervals.

In general, deciding the consistency of a conjunction of temporal constraints on intervals is NP-complete. Many computational complexity results for temporal constraints on time intervals follow from the complexity results for temporal constraint on time points. In particular, any conjunction of pointisable temporal constraints on time intervals can be translated to a conjunction of temporal constraints on time points. After the translation, the consistency can be tested as before.

Key Applications

Temporal constraints are used in scheduling, planning, and temporal database querying [Temporal Databases Queries]. Temporal database queries usually take the form of SQL or Datalog combined with temporal constraints. Examples include Datalog with gap-order constraints [11] and Datalog with periodicity constraints [15].

Future Directions

There are still many open problems on the use of temporal constraints in temporal database query languages. An important problem is finding efficient indexing methods for conjunctions of temporal

constraints. The combination of temporal constraints with spatial constraints is an interesting area within spatiotemporal databases [Spatiotemporal Databases] and constraint databases [12].

Cross-references
► Database Query Languages
► Indexing
► Temporal Dependencies
► Temporal Integrity Constraints
► Temporal Periodicity

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Dechter R., Meiri I., and Pearl J. Temporal constraint networks. Artif. Intell., 49(1-3):61–95, 1991.
3. Dill D.L. Timing assumptions and verification of finite-state concurrent systems. In Proc. Automatic Verification Methods for Finite State Systems, 1989, pp. 197–212.
4. Kabanza F., Stevenne J.-M., and Wolper P. Handling infinite temporal data. J. Comput. Syst. Sci., 51(1):1–25, 1995.
5. Kanellakis P.C., Kuper G.M., and Revesz P. Constraint query languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
6. Koubarakis M. The complexity of query evaluation in indefinite temporal constraint databases. Theor. Comput. Sci., 171(1-2):25–60, 1997.
7. Kuncak V., Nguyen H.H., and Rinard M.C. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In Proc. 20th Int. Conf. on Automated Deduction, 2005, pp. 260–277.
8. Kuper G.M., Libkin L., and Paredaens J. (eds.). Constraint Databases. Springer, Berlin Heidelberg New York, 2000.
9. Lahiri S.K. and Musuvathi M. An efficient decision procedure for UTVPI constraints. In Proc. 5th Int. Workshop on Frontiers of Combining Systems, 2005, pp. 168–183.
10. Péron M. and Halbwachs N. An abstract domain extending difference-bound matrices with disequality constraints. In Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation, 2007, pp. 268–282.
11. Revesz P. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. Theor. Comput. Sci., 116 (1):117–49, 1993.
12. Revesz P. Introduction to Constraint Databases. Springer, Berlin Heidelberg New York, 2002.
13. Revesz P. Quantifier-elimination for the first-order theory of Boolean algebras with linear cardinality constraints. In Proc. 8th Conf. on Advances in Databases and Information Systems, 2004, pp. 1–21.
14. Rosenkrantz D.J. and Hunt H.B. Processing conjunctive predicates and queries. In Proc. 6th Int. Conf. on Very Data Bases, 1980, pp. 64–72.
15. Toman D. and Chomicki J. Datalog with integer periodicity constraints. J. Logic Program., 35(3):263–290, 1998.

Temporal Data Mining

Nikos Mamoulis
University of Hong Kong, Hong Kong, China

Synonyms
Time series data mining; Sequence data mining; Temporal association mining

Definition
Temporal data mining refers to the extraction of implicit, non-trivial, and potentially useful abstract information from large collections of temporal data. Temporal data are sequences of a primary data type, most commonly numerical or categorical values and sometimes multivariate or composite information. Examples of temporal data are regular time series (e.g., stock ticks, EEG), event sequences (e.g., sensor readings, packet traces, medical records, weblog data), and temporal databases (e.g., relations with time-stamped tuples, databases with versioning). The common factor of all these sequence types is the total ordering of their elements. They differ on the type of primary information, the regularity of the elements in the sequence, and on whether there is explicit temporal information associated to each element (e.g., time-stamps). There are several mining tasks that can be applied on temporal data, most of which directly extend from the corresponding mining tasks on general data types. These tasks include classification and regression (i.e., generation of predictive data models), clustering (i.e., generation of descriptive data models), temporal association analysis between events (i.e., causality relationships), and extraction of temporal patterns (local descriptive models for temporal data).

Historical Background
Analysis of time series data has been an old problem in statistics [4], the main application being forecasting for different applications (stock market, weather, etc.) Classic statistical models for this purpose include autoregression and hidden Markov models. The term temporal data mining came along as early as the birth of data mining in the beginning of the 1990s. Soon after association rules mining in large databases [1] has been established as a core research problem, several researchers became interested in association analysis in

long sequences and large temporal databases (see [12] for a survey). One big challenge in temporal data mining is the large volume of the data, which make traditional autoregression analysis techniques inapplicable. Another challenge is the nature of the data which is not limited to numerical-valued time series, but includes sequences of discrete, categorical, and composite values (e.g., sets). This introduces new, interesting types of patterns, like causality relationships between events in time, partial periodic patterns, and calendric patterns.

Foundations

Classic data mining tasks, like classification, clustering, and association analysis can naturally be applied on large collections of temporal data. Special to temporal databases, are the extraction of patterns that are frequent during specific temporal intervals and the identification of temporal relationships between values or events in large sequences. In the following, the above problems are discussed in more detail.

Classification and Clustering

Classification of time series is often performed by nearest neighbor (NN) classifiers [13]. Given a time series $\vec{s}$ of unknown label and a database $\mathcal{D}$ of labeled samples, such a classifier (i) searches in $\mathcal{D}$ for the k most similar time series to $\vec{s}$ and (ii) gives $\vec{s}$ the most popular label in the set of k returned time series. This process involves two challenges: definition of an appropriate *similarity* function to be used by the NN classifier and scalability of classification. The dissimilarity (distance) between two time series is typically quantified by their Euclidean distance or the dynamic time warping (DTW) distance. Like classification, clustering of time series can be performed by applying an off-the-shelf clustering algorithm [7] (e.g., k-means), after defining an appropriate distance (i.e., dissimilarity) function.

For sequences of categorical data, Hidden Markov Models (HMM) can be used to capture the behavior of the data. HMM can be used for classification as follows. For each class label, a probabilistic state transition model that captures the probabilities of seeing one symbol (state) after the current one can be built. Then a sequence is given the label determined by the HMM that describes its behavior best.

Prediction

For continuous-valued sequences, like time series, regression is an alternative to classification. Regression does not use a fixed set of class labels to describe each sequence, but models sequences as functions, which are more appropriate for predicting the values in the future. Autoregression is a special type of regression, where future values are predicted as a linear combination of recent previous values, assuming that the series exhibits a periodic behavior. Formally, an autoregressive model of order p for a time series $\vec{s} = \{s_1, s_2,...\}$ can be described as follows:

$$s_i = e_i + \sum_{j=1}^{p} \phi_j s_{i-j},$$

where $\phi_j (1 \leq j \leq p)$ are the parameters of autoregression, and e_i is an error term. The error terms are assumed to be independent identically-distributed random variables (i.i.d.) that follow a zero-mean normal distribution. The main trend of a time series is commonly described by a *moving average* function, which is a smoothed abstraction of the same length. Formally, the moving average of order q for a time series $\vec{s} = \{s_1, s_2,...\}$ can be described as follows:

$$MA(\vec{s})_i = e_i + \sum_{j=1}^{q} \psi_j e_{i-j},$$

where $\psi_j (1 \leq j \leq q)$ are the parameters of the model. By combining the above two concepts, a time series $\vec{s}$ can be described by an autoregressive moving average (ARMA) model:

$$s_i = e_i + \sum_{j=1}^{p} \phi_j s_{i-j} + \sum_{j=1}^{q} \psi_j e_{i-j},$$

Autoregressive integrated moving average (ARIMA) is a more generalized model, obtained by integrating an ARMA model. In long time series, periodic behaviors tend to be local, so a common practice is to segment the series into pieces with constant behavior and generate an autoregression model at each piece.

Association Analysis and Extraction of Sequence Patterns

Agrawal and Srikant [3] proposed one of the first methods for association analysis in timestamped transactional databases. A transactional database records timestamped customer transactions (e.g., sets of books

bought at a time) in a store (e.g., bookstore) and the objective of the analysis is to discover causality relationships between sets of items bought by customers. An example of such a *sequential* pattern (taken from the paper) is "5% of customers bought 'Foundation,' then 'Foundation and Empire,' and then 'Second Foundation,' " which can be represented by {(Foundation), (Foundation and Empire),(Second Foundation)}. In general, sequential patterns are total orders of *sets of items* bought in the same transaction. For example, {(Foundation,Life),(Second Foundation)} models the buying of "Foundation" and "Life" at a single transaction followed by "Second Foundation" at another transaction. The patterns can be extracted by dividing the database that records the transaction history of the bookstore into groups, one per customer, and then treat each group as an ordered sequence. For example, the transactional database shown in Fig. 1a is transformed to the grouped table of Fig. 1b.

The algorithm for extracting sequential patterns from the transformed database is reminiscent to the Apriori algorithm for frequent itemsets in transactional databases [2]. It takes as input a minimum support threshold *min-sup* and operates in multiple passes. In the first pass, the items that have been bought by at least *min-sup* of the customers are put to a frequent items set L_1. Then, orderings of pairs of items in L_1 form a candidate set C_2 of level-2 sequential patterns, the supports of which are counted during the second pass of the transformed database and the frequent ones form L_2. A sequence adds to the support of a pattern if the pattern is contained in it. For example, the sequence {(A, C),(B, E),(F)} of customer C2 in Fig. 1 adds to the support of pattern {(A),(F)}. In general, after L_k has been formed, the algorithm generates and counts candidate patterns of $k + 1$ items. These candidates are generated by joining pairs (s_1, s_2) of frequent k-sequences, such that the subsequence obtained by dropping the first item of s_1 is identical to the one obtained by dropping the last item of s_2. For example, {(A, B),(C)} and {(B), (C, D)} generate {(A, B), (C, D)}. Candidates resulting from the join phase are pruned if they have a subsequence that is not frequent.

Agrawal and Srikant also considered adding constraints when counting the supports of sequential patterns. For example, if "Foundation and Empire" is bought 3 years after "Foundation," these two books may be considered unrelated. In addition, they considered relaxing the rigid definition of a transaction by unifying transactions of the same customer that took place close in time. For example, if a customer buys a new book minutes after her previous transaction, this book should be included in the previous transaction (i.e., the customer may have forgotten to include it in her basket before). Parallel to Agrawal and Srikant, Mannila et al. [9] studied the extraction of frequent causality patterns (called episodes) in long event sequences. The main differences of this work are (i) the input is a single very long sequence of events (e.g., a stream of sensor indications), (ii) patterns are instantiated by temporal sliding windows along this stream of events, and (iii) patterns can contain sequential modules (e.g., A after B) or parallel modules (e.g., A and B in any order). An example of such an episode is "C first, then A and B in any order, then D". To compute frequent episodes Mannila et al. [9] proposed adaptations of the classic Apriori technique [2]. A more efficient technique for mining sequential patterns was later proposed by Zaki [14].

Han et al. [6] studied the problem of mining partial periodic patterns in long event sequences. In many

Transaction-ID	Time	Customer-ID	Itemset
101	10	C1	A, B, C
102	11	C2	A, C
103	12	C3	B
104	15	C2	B, E
105	18	C2	F
106	20	C1	E
107	25	C3	F

a   Original database

Customer-ID	Time	Itemset
C1	10	A, B, C
C1	20	E
C2	11	A, C
C2	15	B, E
C2	18	F
C3	12	B
C3	25	F

b   Input of mining algorithm

Temporal Data Mining. Figure 1. Transformation of a timestamped transactional database.

applications, the associations between events follow a periodic behavior. For instance, the actions of people follow habitual patterns on a daily basis (i.e., "wake-up," then "have breakfast," then "go to work," etc.). Given a long event sequence (e.g., the actions of a person over a year) and a time period (e.g., 24 h), the objective is to identify patterns of events that have high support over all the periodic time intervals (e.g., days). For this purpose, all subsequences corresponding to the activities of each periodic interval can be extracted from the long sequence, and a sequential pattern mining algorithm [3] can be applied. Based on this idea, an efficient technique for periodic pattern mining, which is facilitated by the use of a sophisticated prefix tree data structure, was proposed by Han et al. [6]. In some applications, the time period every when the patterns are repeated is unknown and has to be discovered from the data. Towards this direction, Cao et al. [5] present a data structure that automatically identifies the periodicity and discovers the patterns at only a small number of passes over the data sequence.

Temporal, Cyclic, and Calendric Association Rules

An association rule in a transactional database may not be strong (according to specific support and confidence thresholds) in the whole database, but only when considering the transactions in a specified time interval (e.g., during the winter of 2005). An association rule bound to a time interval, where it is strong, is termed temporal association rule [12]. Identification of such a rule can be performed by starting from short time intervals and progressively extending them to the maximum possible length where the rule remains strong.

Özden et al. [10] noticed that association rules in transactional databases (e.g., people who buy turkey they also buy pumpkins) may hold only in particular temporal intervals (e.g., during the last week of November every year). These are termed *cyclic* association rules, because they are valid periodically, at a specific subinterval of a cycle (e.g., year). Such rules can be discovered by identifying the periodic intervals of fixed granularity (e.g., week of the year), which support the associations.

Cyclic rules are assumed to be supported at *exact* intervals (e.g., the last day of January), and at *every* cycle (e.g., every year). In practice, a rule may be supported with some mismatch threshold (e.g., the last weekday of January) and only at the majority of cycles (e.g., 80% of the cycles). Accordingly, the

"cyclic" rule concept was extended by Ramaswamy et al. [11] to the more flexible *calendric* association rule. A calendar is defined by a set of time intervals (e.g., the last 3 days of January, every year). For a calendric rule to be strong, it should have enough support and confidence in at least *min-sup%* of the time units included in the calendar. An algerbra for defining calendars and a method for discovering calendric association rules referring to them can be found in Ref. [11].

Li et al. [8] proposed a more generalized framework for calendric association rules. Instead of searching based on a predetermined calendar, they automatically identify the rules and their supporting calendars, taken from a hierarchy of calendar concepts. The hierarchy is expressed by a relation of temporal generalizations of varying granularity, e.g., *R(year, month, day)*. A possible calendric pattern is expressed by setting to each attribute, either a specific value of its domain, or a wildcard value "∗." For example, pattern (∗, *Jan*, 30) means the 30th of January each year, while (2005, ∗, 30) means the the 30th day of each month in year 2005. By defining containment relationships between such patterns (e.g., (∗, ∗, 30) contains all the intervals of (2005, ∗, 30)) and observing that itemset supports for them can be computed constructively (e.g., the support of an itemset in (∗, ∗, 30) can be computed using its support in all $(y, ∗, 30)$ for any year y), Li et al. [8] systematically explore the space of all calendric patterns using the Apriori principle to prune space (e.g., an itemset is not frequent in (∗, ∗, 30) if it is infrequent in all $(y, ∗, 30)$ for every year y).

Key Applications

Weather Forecasting

Temporal causality relationships between events can assist the prediction of weather phenomena. In fact, such patterns have been used for this purpose since the ancient years (e.g., "if swallows fly low, it is going to rain soon").

Market Basket Analysis

Extension of classic association analysis to consider temporal information finds application in market analysis. Examples include, temporal relationships between

products that are purchased within the same period by customers ("5% of customers bought 'Foundation,' then 'Foundation and Empire' ") and calendric association rules (e.g., turkey is bought together with pumpkin during the last week of November, every year).

Stock Market Prediction

Time-series classification and regression is often used by financial analysts to predict the future behavior of stocks. The structure of the time series can be compared with external factors (such as pieces of news) to derive more complex associations that result in better accuracy in prediction.

Web Data Mining

The World Wide Web can be viewed as a huge graph where nodes correspond to web pages (or web sites) and edges correspond to links between them. Users navigate through the web defining sequences of page visits, which are tracked in weblogs. By analyzing these sequences one can identify frequent sequential patterns between web pages or even classify users based on their behavior (sequences of sites they visit and sequences of data they download).

Cross-references

▶ Association rules

▶ Spatial and Spatio-Temporal Data Models and Languages

▶ Temporal Periodicity

▶ Time Series Query

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A.N. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules in large databases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
4. Box G.E.P. and Jenkins G. Time Series Analysis, Forecasting and Control. Holden-Day, 1990.
5. Cao H., Cheung D.W., and Mamoulis N. Discovering partial periodic patterns in discrete data sequences. In Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conf., 2004, pp. 653–658.
6. Han J., Dong G., and Yin Y. Efficient mining of partial periodic patterns in time series database. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 106–115.
7. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
8. Li Y., Ning P., Wang X.S., and Jajodia S. Discovering calendar-based temporal association rules. Data Knowl. Eng., 44(2):193–218, 2003.
9. Mannila H., Toivonen H., and Verkamo A.I. Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov., 1(3):259–289, 1997.
10. Özden B., Ramaswamy S., and Silberschatz A. Cyclic association rules. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 412–421.
11. Ramaswamy S., Mahajan S., and Silberschatz A. On the discovery of interesting patterns in association rules. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 368–379.
12. Roddick J.F. and Spiliopoulou M. A survey of temporal knowledge discovery paradigms and methods. IEEE Trans. Knowl. Data Eng., 14(4):750–767, 2002.
13. Wei L. and Keogh E.J. Semi-supervised time series classification. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 748–753.
14. Zaki M.J. Spade: an efficient algorithm for mining frequent sequences. Mach. Learn., 42(1/2):31–60, 2001.

Temporal Data Models

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Valid-time data model; Transaction-time data model; Bitemporal data model; Historical data model

Definition

A "data model" consists of two components, namely a set of objects and a language for querying those objects [4]. In a temporal data model the objects vary over time, and the operations in some sense "know" about time. Focus has been on the design of data models where the time references capture valid time, or transaction time, or a combination of both (for bitemporal data).

Historical Background

Almost all real-world databases contain time-referenced data. Few interesting databases are entirely stagnant, and when the modeled reality changes, the database must be updated. Usually at least the start time of currently valid data are captured, though most databases also retain previous data.

Two decades of research into temporal databases have unequivocally shown that a *time-referencing table*, containing certain kinds of time-valued columns that capture one or more temporal aspects of data recorded in other columns, is completely different from this table, without the time-valued columns. Effectively designing, querying, and modifying time-referencing tables requires a different set of approaches and techniques. It is possible to handle such data within standard data models, generally at the expense of high data redundancy, awkward modeling, and unfriendly query languages. An alternative is a data model, probably an extension of an extant, non-temporal data model, that explicitly incorporates time, making it easier to express queries, modifications, and integrity constraints.

As an example, consider a primary key of the following relation, which records the current positions of employees, identified by their social security numbers: `EMP(EmpID, POSITION)`. The primary key is obviously `EmpID`. Now add `STARTTIME` and `STOP-TIME` attributes. While a primary key of (`EmpID`, `STARTTIME`) seems to work, such a primary key will not prevent overlapping periods, which would allow an employee to have two positions at a point of time, which is problematic. Stating the primary key constraint properly requires a complex assertion containing a dozen lines of code with multiple sub-queries [3]. Referential integrity is even more challenging.

The last two decades have seen the introduction of a great many temporal data models, not just in the context of relational data, as in the above example, but also with object-oriented, logic-based, and semi-structured data.

Foundations

Levels of Abstraction
Temporal data models exist at three abstraction levels: the *conceptual level*, in which the data models are generally extensions of the Entity-Relationship Model, the *logical level*, in which the data models are generally extensions of the relational data model or of an object-oriented data model, and, infrequently, the *physical level*, in which the data model details how the data are to be stored. In terms of prevalence, models at the logical level are by far the most numerous. However, it has been shown that several of the models that were originally proposed as logical data models are actually

equivalent to the BCDM logical model, and should more properly be viewed as physical data models [2]. This entry is restricted to logical models, focusing on the objects that are subject to querying rather than the query languages. First examined is the association of time with data, as this is at the core of temporal data management.

Temporal Aspects of Data
A database models and records information about a part of reality, termed the modeled reality. Aspects of the modeled reality are represented in the database by a variety of structures, termed database entities. In general, times are associated with database entities. The term "fact" is used for any (logical) statement that can meaningfully be assigned a truth value, i.e., true or false.

The facts recorded by database entities are of fundamental interest, and a fundamental temporal aspect may be associated with these: the valid time of a fact is the times when the fact is true in the modeled reality. While all facts have a valid time by definition, the valid time of a fact may not necessarily be recorded in the database. For example, the valid time may not be known, or recording it may not be relevant. Valid time may be used for the capture of more application-specific temporal aspects. Briefly, an application-specific aspect of a fact may be captured as the valid time of another, related fact.

Next, the transaction time of a database entity is the time when the entity is current in the database. Like valid time, this is an important temporal aspect. Transaction time is the basis for supporting accountability and "traceability" requirements. Note that transaction time, unlike valid time, may be associated with any database entity, not only with facts. As for valid time, the transaction-time aspect of a database entity may or may not be captured in the database. The transaction-time aspect of a database entity has a duration: from insertion to deletion. As a consequence of the semantics of transaction time, deleting an entity does not physically remove the entity from the database; rather, the entity remains in the database, but ceases to be part of the database's current state.

Observe that the transaction time of a database fact, say *f*, is the valid time of the related fact, "*f* is current in the database." This would indicate that supporting transaction time as a separate aspect is redundant. However, both valid and transaction time

are aspects of the content of all databases, and recording both of these is essential in many applications. In addition, transaction time, due to its special semantics, is particularly well-behaved and may be supplied automatically by the DBMS. Specifically, the transaction times of a fact stored in the database is bounded by the time the database was created at one end of the time line and by the current time at the other end.

The above discussion suggests why temporal data models generally offer built-in support for one or both of valid and transaction time.

Representation of Time

The valid and transaction time values of database entities are drawn from some appropriate time domain. There is no single answer to how to perceive time in reality and how to represent time in a database, and different time domains may be distinguished with respect to several orthogonal characteristics. First, the time domain may or may not stretch infinitely into the past and future. Second, time may be perceived as discrete, dense, or continuous. Some feel that time is really continuous; others contend that time is discrete and that continuity is just a convenient abstraction that makes it easier to reason mathematically about certain discrete phenomena. In databases, a finite and discrete time domain is typically assumed, e.g., in the SQL standards. Third, a variety of different structures have been imposed on time. Most often, time is assumed to be totally ordered.

Much research has been conducted on the semantics and representation of time, from quite theoretical topics, such as temporal logic and infinite periodic time sequences, to more applied questions such as how to represent time values in minimal space. Substantial research has been conducted that concerns the use of different time granularities and calendars in general, as well as the issues surrounding the support for indeterminate time values. Also, there is a significant body of research on time data types, e.g., time instants, time intervals (or "periods"), and temporal elements.

Data Model Objects

The management of temporal aspects has been achieved by building time into the data model objects. Here, the relational model is assumed, with a focus on valid time. One approach is to timestamp tuples with time instants, or points. Then a fact is represented by one tuple for each time point during which the fact is valid. An example instance for the EMP relation example is shown in Fig. 1.

A distinguishing feature of this approach is that (syntactically) different relations have different information content. Next, timestamps are atomic values that can be easily compared. Assuming a totally ordered time domain, the standard set of comparison predicates, $=$, $\neq$, $<$, $>$, $\leq$, and $\geq$, is sufficient to conveniently compare timestamps. The conceptual simplicity of time points comes at a cost, though. The model offers little support for capturing, e.g., that employee 2 was assigned to Sales during two contiguous periods [4,5] and [6,7], instead of during a single contiguous period [4,7].

It is important to note that the point model is not meant for physical representation, as for all but the most trivial time domains, the space needed when using the point model is prohibitive. The combination of conceptual simplicity and low computational complexity has made the point model popular for theoretical studies.

Another type of data model uses time periods as timestamps. This type of model associates each fact with a period that captures the valid time of the fact. Multiple tuples are needed if a fact is valid over disjoint periods. Figure 2 illustrates the approach.

The notion of snapshot equivalence, which reflects a point-based view of data, establishes a correspondence between the point-based and period-based models. Imagine that the last two tuples in the relation in Fig. 2 were replaced with the single tuple (2, Sales, [4,7]) to obtain a new relation. The resulting two relations are different, but snapshot equivalent.

EmpID	Position	T
1	Sales	3
1	Sales	4
1	Engineering	5
1	Engineering	6
2	Sales	4
2	Sales	5
2	Sales	6
2	Sales	7

Temporal Data Models. Figure 1. Point model.

Specifically, the new relation is a coalesced version of the original relation.

In some data models, the relations are taken to contain the exact same information. These models adopt a point-based view and are only period-based in the weak sense that they use time periods as convenient representations of (convex) sets of time points. It then also makes sense for such models to require that their relation instances be *coalesced*. This requirement ensures that relation instances that are syntactically different are also semantically different, and vice versa. In such models, the relation in Fig. 2 is not allowed.

In an inherently period-based model, periods carry meaning beyond denoting a set of points. In some situations, it may make a difference whether an employee holds a position for two short, but consecutive time periods versus for one long time period. Period-based models do not enforce coalescing and capture this distinction naturally.

Next, a frequently mentioned shortcoming of periods is that they are not closed under all set operations, e.g., subtraction. This has led to the proposal that temporal elements be used as timestamps instead. These are finite unions of periods.

With *temporal elements*, the same two semantics as for periods are possible, although models that use temporal elements seem to prefer the point-based semantics. Figure 3a and Figure 3b uses temporal elements to capture the example assuming the period-based semantics and point-based semantics,

respectively. (As [4,5] ∪ [6,7] = [4,7], this latter period could have been used instead of [4,5] ∪ [6,7].)

Note that the instance in Fig. 3b exemplifies the instances used by the point-based bitemporal conceptual data model (BCDM) when restricted to valid time. This model has been used for TSQL2. The BCDM timestamps facts with values that are sets of time points. This is equivalent to temporal elements because the BCDM adopts a discrete and bounded time domain.

Because value-equivalent tuples are not allowed (this corresponds to the enforcement of coalesced relations as discussed earlier), the full history of a fact is contained in exactly one tuple, and one tuple contains the full history of exactly one fact. In addition, relation instances that are syntactically different have different information content, and vice versa. This design decision reflects the point-based underpinnings of the BCDM.

With temporal elements, the full history of a fact is contained in a single tuple, but the information in a relation that pertains to some real-world object may still be spread across several tuples. To capture all information about a real-world object in a single tuple, attribute value timestamping has been introduced. This is illustrated in Fig. 4, which displays the sample instance using a typical attribute-value timestamped data model.

The instance records information about employees and thus holds one tuple for each employee, with a tuple containing all information about an employee. An obvious consequence is that the information about a position cannot be contained in a single tuple. Another observation is that a single tuple may record multiple facts. In the example, the first tuple records two facts: the position type for employee 1 for the two positions, Sales and Engineering.

It should also be noted that different groupings into tuples are possible for this attribute-value timestamping model. Figure 5 groups the relation instance in Fig. 4 on the POSITION attribute, indicating that it is now the positions, not the employees, that are the objects in focus.

EmpID	Position		T
1	Sales		[3, 4]
1	Engineering		[5, 6]
2	Sales		[4, 5]
2	Sales		[6, 7]

Temporal Data Models. Figure 2. Period (or interval) model.

EmpIS	Position		T
1	Sales		[3, 4]
1	Engineering		[5, 6]
2	Sales		[4, 5]
2	Sales		[6, 7]

a   Period view

EmpID	Position		T
1	Sales		[3, 4]
1	Engineering		[5, 6]
2	Sales		[4, 5] ∪ [6, 7]

b   Point view

Temporal Data Models. Figure 3. Temporal element model.

EmpID		Position	
[3, 6]	1	[3, 4]	Sales
		[5, 6]	Engineering
[4, 7]	2	[4, 7]	Sales

Temporal Data Models. Figure 4. Attribute-value timestamped model.

EmpID		Position	
[3, 4]	1	[3, 7]	Sales
[4, 7]	2		
[5, 6]	1	[5, 6]	Engineering

Temporal Data Models. Figure 5. Attribute-value timestamped model, grouped on POSITION.

Data models that timestamp attribute values may be temporally grouped. In a temporally grouped model, all aspects of a real-world object may be captured by a single tuple [1].

At first sight, that attribute-value timestamped model given above is temporally grouped. However, with a temporally grouped model, a real-world object is allowed to change the value for its key attribute. In the example, this means that the instance in Fig. 6 should be possible. Now observe that when grouping this instance on EmpID or POSITION, or both, it is not possible to get back to the original instance. Thus, temporally grouped tuples are not well supported. Clifford et al. [1] explore the notion of temporally grouped models in considerable depth.

Query Languages
Having covered the objects that are subject to querying, the last major aspect of a logical data model is the query language associated with the objects. Such languages come in several variants.

Some are intended for internal use inside a temporally enhanced database management system. These are typically algebraic query languages. However, algebraic languages have also been invented for more theoretical purposes. For example, an algebra may be used for defining the semantics of a temporal SQL extension. A key point is that an algebra is much simpler than is such an extension. Little is needed in terms of

EmpID		Position	
[3, 5]	1	[3, 4]	Engineering
[6, 6]	3	[5, 6]	
[4, 7]	2	[4, 7]	Sales

Temporal Data Models. Figure 6. Attribute-value timestamped model, temporally grouped.

language design; only a formal definition of each operator is needed.

Other query languages are targeted at application programmers and are thus typically intended to replace SQL. The vast majority of these are SQL extensions. Finally, languages have been proposed for the purpose of conducting theoretical studies, e.g., of expressive power.

Key Applications
Virtually all databases contain temporal information, and so virtually all database applications would benefit from the availability of data models that provide natural support for such time-varying information.

Future Directions
Rather than come up with a new temporal data model, it now seems better to extend, in an upward-consistent manner, existing non-temporal models to accommodate time-varying data.

Cross-references
► Period-Stamped Temporal Models
► Point-Stamped Temporal Models
► Probabilistic Temporal Databases
► Supporting Transaction Time Databases
► Temporal Access Control
► Temporal Compatibility
► Temporal Concepts in Philosophy
► Temporal Conceptual Models
► Temporal Constraints
► Temporal Database
► Temporal Indeterminacy
► Temporal Logical Models
► Temporal Object-Oriented Databases
► Temporal Query Languages
► Temporal XML
► Transaction Time
► Valid Time

Recommended Reading

1. Clifford J., Croker A., and Tuzhilin A. On completeness of historical relational query languages. ACM Trans. Database Syst., 19(1):64–16, March 1994.
2. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December 1994.
3. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, San Francisco, CA, July 1999.
4. Tsichritzis D.C. and Lochovsky F.H. Data Models. Software Series. Prentice-Hall, 1982.

Temporal Data Warehousing

▶ Data Warehouse Maintenance, Evolution and Versioning

Temporal Database

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Historical database; Time-oriented database

Definition

A temporal database is a collection of time-referenced data. In such a database, the time references capture some temporal aspect of the data; put differently, the data are timestamped. Two temporal aspects are prevalent. The time references may capture either the past and current states of the database, yielding a transaction-time database; they may capture states of the reality being modeled by the data, yielding a valid-time database; or they may capture both aspects of the data, yielding a bitemporal database.

Historical Background

"Time" is a fundamental concept that pervades all aspects of daily life. As one indicator, a recent study by Oxford University Press found that the word "time" is the most commonly used noun in the English language. The nouns "year" and "day" rank third and fifth in the study.

Moving to a database context, the capture and representation of time-varying information go back literally thousands of years. The Egyptians and Syrians carved records into stone walls and pyramids of inventories of grain over many years. But it has been only in the last few decades that the advent of increasingly inexpensive and voluminous digital storage has enabled the computerized management of increasingly large volumes of time-referenced data.

Temporal databases have been the subject of intense study since the early 1980s. A series of bibliographies on temporal databases enumerates the thousands of refereed papers that have been written on the subject (the series started with Boulour's paper [3]; the most recent in the series is by Wu et al. [17]). There have also been more specialized bibliographies [2,9,13].

The first temporal database conference was held in Sofie Antipolis, France in 1987 [1]. Two workshops, the Temporal Database Workshop in Arlington, Texas in 1993 and in Zürich, Switzerland in 1995 [8], were held subsequently. The premier conferences on the topic are the International Symposium on Temporal Representation and Reasoning (*TIME*) (held annually since 1994), the International Symposium on Spatial and Temporal Databases (*SSTD*) (held biannually), and the Spatio-Temporal Database Management (*STDBM*) series (three up through 2006).

A seminal book collecting important results to date in this field appeared in 1993 [16]. Several surveys have been written on the topic [4–7, 10–12, 14, 15]. Temporal databases were covered in detail in an advanced database textbook [18].

Foundations

Time impacts all aspects of database technology, including database design (at the conceptual, logical, and physical levels) and the technologies utilized by a database management system, such as query and modification languages, indexing techniques and data structures query optimization and query evaluation techniques, and transaction processing.

The entries related to temporal databases go into more detail about these aspects. The following provides an organization on those entries (which are indicated in *italics*).

General Concepts

- Philosophers have thought hard about time (*temporal concepts in philosophy*).
- Two general temporal aspects of data attract special attention: *valid time* and *transaction time.*
- The *time domain* can be differentiated along several aspects: its structure, e.g., linear or branching; discrete versus continuous; bounded or infinite.
- Just as multiple versions of data may be stored, independently, the schemas can be versioned (*schema versioning*).
- The concept of "now" is important (*now in temporal databases*).

Temporal Data Models

- *Temporal conceptual models* generally extend an existing conceptual model, such as one of the variants of the Entity-Relationship model.
- *Temporal logical models* generally extend the relational model or an object-oriented model (*temporal object-oriented models*) or XML (*temporal XML*).
- Data can be associated with time in several ways: with time points (*point-stamped temporal models*) or time periods (*period-stamped temporal models*); these may capture valid and/or transaction time; and the associations of the data with the time values may carry probabilities (*temporal probabilistic models*).
- The time values associated with the data are characterized by their *temporal granularity*, and they may possess *temporal indeterminacy* and *temporal periodicity.*
- Data models incorporate *temporal constraints*, *temporal integrity constraints*, and *temporal dependencies.*

Temporal Query Languages

- Most *temporal query languages* are based on the relational algebra or calculus. Not surprisingly, much attention has been given to the design of user-level temporal query languages, notably *SQL-based temporal query languages*. For such languages, different notions of *temporal compatibility* have been an important design consideration.
- *Qualitative temporal reasoning* and *temporal logic in database query languages* provide expressive query facilities.

- *Temporal vacuuming* provides a way to control the growth of an otherwise append-only transaction-time database.
- *TSQL2* and its successor SQL/Temporal provided a way for many in the temporal database community to coordinate their efforts in temporal query language design and implementation.
- *Temporal query processing* involves disparate architectures, from *temporal strata* outside the conventional DBMS to adding native temporal support within a DBMS.
- *Supporting transaction time* in an efficient manner in the context of transactions is challenging and generally requires changes to the kernel of a DBMS.
- *Temporal algebras* extend the conventional relational algebra. Some specific operators (e.g., *temporal aggregation*, *temporal coalescing*, *temporal joins*) have received special attention.
- Temporal storage structures and indexing techniques have also received a great deal of attention (*temporal indexing*).
- *Temporal visual languages* have also been designed that present graphical user interfaces, as contrasted with the textual form of the temporal query languages mentioned previously.

Temporal Applications

- *Temporal access control* uses temporal concepts in database security.
- *Temporal data mining* has recently received a lot of attention.
- *Time series* has also been an active area of research.
- Other applications include temporal constraint satisfaction, support for planning systems, and natural language disambiguation.

The concepts of temporal databases are also making their way into research on data warehousing, OLAP, and data streams.

Key Applications

As storage costs decrease, more databases are retaining historical data. The dual of such a decrease is that the cost of deletion is effectively increasing, as the application then has to explicitly make the decision on what to retain and what to delete, and the DBMS has to revisit the data on disk in order to move it. Some have asserted that it may be simpler to simply disallow deletion (except for purging of unneeded records,

termed *temporal vacuuming*) within a DBMS, rendering all databases by default temporal databases.

Commercial products are starting to include temporal support. Most notably, the Oracle database management system has included temporal support from its 9i version. Lumigent's LogExplorer product provides an analysis tool for Microsoft SQLServer logs, to allow one to view how rows change over time (a nonsequenced transaction-time query) and then to selectively back out and replay changes, on both relational data and the schema (it effectively treats the schema as a transaction-versioned schema). aTempo's Time Navigator is a data replication tool for DB2, Oracle, Microsoft SQL Server, and Sybase that extracts information from a database to build a slice repository, thereby enabling image-based restoration of a past slice; these are transaction time-slice queries. IBM's DataPropagator can use data replication of a DB2 log to create both before and after images of every row modification to create a transaction-time database that can be later queried.

Future Directions

While much progress has been made in all of the above listed areas, temporal database concepts and technologies have yet to achieve the desired levels of simplification and comprehensibility. While many of the subtleties of temporal data and access and storage thereof have been investigated, in many cases quite thoroughly, a synthesis is still needed of these concepts and technologies into forms that are usable by novices.

Given the simplifications of data management afforded by built-in support of time in database management systems, it is hoped that DBMS vendors will continue to enhance the temporal support in their products.

Cross-references

▶ Bi-Temporal Indexing
▶ Now in Temporal Databases
▶ Period-Stamped Temporal Models
▶ Point-Stamped Temporal Models
▶ Probabilistic Temporal Databases
▶ Qualitative Temporal Reasoning
▶ Schema Versioning
▶ SQL-Based Temporal Query Languages
▶ Supporting Transaction Time Databases
▶ Temporal Access Control
▶ Temporal Aggregation
▶ Temporal Algebras
▶ Temporal Coalescing
▶ Temporal Compatibility
▶ Temporal Concepts in Philosophy
▶ Temporal Conceptual Models
▶ Temporal Constraints
▶ Temporal Database
▶ Temporal Data Mining
▶ Temporal Data Models
▶ Temporal Dependencies
▶ Temporal Granularity
▶ Temporal Indeterminacy
▶ Temporal Integrity Constraints
▶ Temporal Joins
▶ Temporal Logic in Database Query Languages
▶ Temporal Logical Models
▶ Temporal Object-Oriented Databases
▶ Temporal Periodicity
▶ Temporal Query Languages
▶ Temporal Query Processing
▶ Temporal Strata
▶ Temporal Vacuuming
▶ Temporal Visual Languages
▶ Temporal XML
▶ Time Domain
▶ Time Series
▶ TSQL2

Recommended Reading

1. Rolland C., Bodart F., and Leonard M. (eds.). In Proc. Conf. on Temporal Aspects in Information Systems. North-Holland/Elsevier, 1987.
2. Al-Tara K.K., Snodgrass R.T., and Soo M.D. A bibliography on spatio- temporal databases. Int. J. Geogr. Inf. Syst., 8(1):95–103, January–February 1994.
3. Boulour A., Anderson T.L., Dekeyser L.J., and Wong H.K.T. The role of time in information processing: a survey. ACM SIGMOD Rec., 12(3):27–50,April 1982.
4. Böhlen M.H., Gamper J., and Jensen C.S. Temporal databases. In Handbook of Database Technology, J. Hammer, M. Schneider (eds.). Computer and Information Science Series. Chapman and Hall, to appear.
5. BöhlenM.H. and Jensen C.S. Temporal data model and query language concepts. In Vol. 4.Encyclopedia of Information Systems, Academic, New York, NY, USA, 2003, pp. 437–453.
6. Chomicki J. Temporal query languages: A survey. In Proc. 1st Int. Conf. on Temporal Logic, 1994, pp. 506–534.

7. Chomicki J. and Toman D. Temporal databases. In Handbook of Time in Artificial Intelligence, M.' Fisher et al. (eds.). Elsevier, Amsterdam, The Netherlands, 2005.

8. Clifford J. and Tuzhilin A. (eds.). In Proc. International Workshop on Temporal Databases: Recent Advances in Temporal Databases, 1995.

9. Grandi F. Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web. ACM SIGMOD Rec., 33(2):84–86, June 2004.

10. Jensen C.S. and Snodgrass R.T. Temporal data management. IEEE Trans. Knowl. Data Eng., 11(1):36–44, January/February 1999.

11. López I.F.V., Snodgrass R.T., and Moon B. Spatiotemporal aggregate computation: A survey. IEEE Trans. Knowl. Data Eng., 17(2):271–286, February 2005.

12. Özsoyoğlu G. and Snodgrass R.T., Temporal and real-time databases: A survey. IEEE Trans. Knowl. Data Eng., 7(4):513–532, August 1995.

13. Roddick J.F. and Spiliopoulou M. A bibliography of temporal, spatial and spatio-temporal data mining research. SIGKDD Explor., 1(1):34–38, January 1999.

14. Snodgrass R.T. Temporal databases: Status and research directions. ACM SIGMOD Rec., 19(4):83–89, December 1990.

15. Snodgrass R.T. Temporal databases. In Proc. Int. Conf. on GIS: From Space to Territory, 1992, pp, 22–61.

16. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). Temporal databases: Theory, design, and implementation. Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, USA, March 1993, pp. 633+xx.

17. Wu Y., Jagodia S., and Wang X.S. Temporal database bibliography update. In Temporal Databases – Research and Practice. D. Etzion, S. Jajodla and S. Sripada (eds.). Springer, Berlin, 1998, pp. 338–367.

18. Zaniolo C., Ceri S., Faloutsos C., Snodgrass R.T., Subrahmanian V.S., and Zicari R. Advanced Database Systems. Morgan Kaufmann, San Francisco, CA, 1997.

Temporal Dependencies

JEF WIJSEN
University of Mons-Hainaut, Mons, Belgium

Definition

Static integrity constraints involve only the current database state. Temporal integrity constraints involve current, past, and future database states; they can be expressed by essentially unrestricted sentences in temporal logic. Certain syntactically restricted classes of temporal constraints have been studied in their own right for considerations of feasibility or practicality;

they are usually called temporal dependencies. Most temporal dependencies proposed in the literature are dynamic versions of static functional dependencies.

Historical Background

Static dependencies (functional, multivalued, join, and other dependencies) have been investigated in depth since the early years of the relational model. Classical problems about dependencies concern logical implication and axiomatization. The study of a particular dependency class is often motivated by its practical importance in databases. This is undeniably the case for the notion of functional dependency (FD), which is fundamental in database design. A dynamic version of functional dependencies was first proposed by Vianu [7]. Since the mid 1990's, several other temporal variants of the notion of FD have been introduced.

Foundations

This section gives an overview of several temporal dependency classes proposed in the literature. With the exception of the notion of dynamic algebraic dependency (DAD) [2], all temporal dependencies here presented can be seen as special cases of the notion of constraint-generating dependency (CGD) [1]. The formalism of CGD, presented near the end, thus allows to compare and contrast different temporal dependency classes.

Functional Dependencies Over Temporal Databases

Since the semantics of temporal versions of FDs will be explained in terms of static FDs, the standard notion of FD is recalled next. All the following definitions are relative to a fixed set $U = \{A_1,...,A_n\}$ of *attributes*.

Definition   A tuple over U is a set $t = \{A_1 : c_1,...,A_n : c_n\}$, *where each c_i is a constant. If $X \subseteq U$, then $t[X]$ denotes the restriction of t to X. A relation over U is a finite set of tuples over U.*

A functional dependency (FD) over U is an expression $X \rightarrow Y$ where $X,Y \subseteq U$. A relation I *over U satisfies the FD $X \rightarrow Y$ if for all tuples $s,t \in I$, if $s[X] = t[X]$, then $s[Y] = t[Y]$.*

When evaluating FDs over temporal relations, one may want to treat timestamp attributes different from other attributes. To illustrate this, consider the temporal relation *EmpInterval* with its obvious meaning.

EmpInterval

Name	Sex	Sal	Project	From	To
Ed	M	10K	Pulse	1	3
Ed	M	10K	Wizard	2	3
Ed	M	12K	Wizard	4	4

An employee has a unique sex and a unique salary, but can work for several projects. The salary, unlike the sex, may change over time. The relation *EmpInterval* is "legal" with respect to these company rules, because for any time point i, the *snapshot* relation $\{t[Name, Sex, Sal, Project] \mid t \in EmpInterval, t(From) \leq i \leq t(To)\}$ satisfies *Name* → *Sex* and *Name* → *Sal*. Note that the relation *EmpInterval* violates the FD *Name* → *Sal*, because the last two tuples agree on *Name* but disagree on *Sal*. However, since these tuples have disjoint periods of validity, they do not go against the company rules.

Hence, the intended meaning of an FD expressed over a temporal relation may be that the FD must be satisfied at every snapshot. To indicate that *Name* → *Sal* has to be evaluated on snapshots, Jensen et al. [6] use the notation *Name* $\overset{T}{\to}$ *Sal*. The FD *Name* → *Sex* needs no change, because the sex of an employee should be unique not only at each snapshot, but also over time.

Chomicki and Toman [3] note that no special syntax is needed if tuples are timestamped by time points. For example, given the following point-stamped temporal relation, one can simply impose the classical FDs *Name* → *Sex* and *Name*, *T* → *Sal*.

EmpPoint

Name	Sex	Sal	Project	T
Ed	M	10K	Pulse	1
Ed	M	10K	Pulse	2
Ed	M	10K	Pulse	3
Ed	M	10K	Wizard	2
Ed	M	10K	Wizard	3
Ed	M	12K	Wizard	4

Vianu's Dynamic Functional Dependency [7]
Consider an employee table with attributes *Name*, *Sex*, *Merit*, and *Sal*, with their obvious meanings. The primary key is *Name*. Assume an annual companywide update of merits and salaries. In the relation shown next, every tuple is followed by its updated version. Old values appear in columns with a caron ($\vee$), new values in columns with a caret ($\wedge$). For example, John Smith's salary increased from 10 to 12K. Such a relation that juxtaposes old and new values is called *action relation*.

⊦ old ⊣				⊦ new ⊣			
Năme	Sĕx	Mĕrit	Săl	Nâme	Sêx	Mêrit	Sâl
John Smith	M	Poor	10K	John Smith	M	Good	12K
An Todd	F	Fair	10K	An Todd	F	Good	12K
Ed Duval	M	Fair	10K	Ed Duval	M	Fair	10K

The company's policy that "*Each new salary is determined solely by new merit and old salary*" can be expressed by the classical FD *Săl*, *Mêrit* → *Sâl* on the above action relation. In particular, since John Smith and An Todd agree on old salary and new merit, they must have the same new salary. Such FDs on action relations were introduced by Vianu [7] and called dynamic functional dependencies.

Definition For each $A_i \in U$, assume that $\check{A}_i$ and $\hat{A}_i$ are new distinct attributes. Define $\check{U} = \{\check{A}_1,...,\check{A}_n\}$ and $\hat{U} = \{\hat{A}_1,...,\hat{A}_n\}$. For $t = \{A_1 : c_1,...,A_n : c_n\}$, define:

$$\check{t} = \{\check{A}_1 : c_1,...,\check{A}_n : c_n\}, \text{ a tuple over } \check{U}; \text{ and}$$
$$\hat{t} = \{\hat{A}_1 : c_1,...,\hat{A}_n : c_n\}, \text{ a tuple over } \hat{U}$$

A *Vianu dynamic functional dependency* (VDFD) over U is an FD $X \to Y$ over $\check{U}\hat{U}$ such that for each $A \in Y$, XA contains at least one attribute from $\check{U}$ and one attribute from $\hat{U}$.

An update *over U is a triple* $\langle I, \mu, J \rangle$, *where I and J are relations over U and* μ *is a bijective mapping from I to J. The update* $\langle I, \mu, J \rangle$ satisfies *the VDFD* $X \to Y$ *if the* action relation $\{\check{t} \cup \hat{s} \mid t \in I, s = \mu(t)\}$ *satisfies the FD* $X \to Y$.

The notion of VDFD directly extends to sequences of database updates. The interaction between dynamic VDFDs and static FDs is studied in [7].

Temporal Extensions of Functional Dependency Proposed by Wijsen [9, 10, 11, 12]

Instead of extending each tuple with its updated version, as is the case for VDFDs, one can take the union of the old relation and the new relation:

Name	Sex	Merit	Sal		
John smith	M	Poor	10K	$\top$	
An todd	F	Fair	10K	old	
Ed duval	M	Fair	10K	$\bot$	$\top$
John smith	M	Good	12K		new
An todd	F	Good	12K		$\bot$

The company's policy that "*Every change in merit (promotion or demotion) gives rise to a salary change*," is expressed by *Name, Sal* $\overset{\circ}{\to}$ *Merit* and means that the FD *Name, Sal* $\to$ *Merit* must be satisfied by the union of the old and the new relation. In particular, since Ed Duval's salary did not change, his merit cannot have changed either. Likewise, "*The sex of an employee cannot change*" is expressed by *Name* $\overset{\circ}{\to}$ *Sex*. The construct $\overset{\circ}{\to}$ naturally generalizes to database histories that involve more than two database states.

Definition A Wijsen dynamic functional dependency (WDFD) over *U* is an expression of the form $X \overset{\circ}{\to} Y$, where *X, Y* $\subseteq$ *U*.

A database history *is a sequence* $\langle I_1, I_2, I_3, ... \rangle$, *where each* I_i *is a relation over U. This history* satisfies $X \overset{\circ}{\to} Y$ *if for every* $i \in \{1,2,...\}$, $I_i \cup I_{i+1}$ satisfies $X \to Y$.

Although I_{i+1} can be thought of as the result of an update performed on I_i, there is no need to model a one-one relationship between the tuples of both relations, as was the case for VDFDs. VDFDs and WDFDs capture different types of constraints, even in the presence of some attribute that serves as a time-invariant tuple-identifier. This difference will be illustrated later on in the discussion of constraint-generating dependencies.

In practice, database histories will be be stored in relations with timestamped tuples. Like FDs, WDFDs can result in predictable (i.e., redundant) values. For example, if the following relation has to satisfy *Name, Sal* $\overset{\circ}{\to}$ *Merit*, then the value for the placeholder † must be equal to "Poor." Wijsen [9] develops temporal variants of 3NF to avoid data redundancy caused by WDFDs.

WDFDs can be naturally generalized as follows: instead of interpreting FDs over unions of successive database states, the syntax of FD is extended with a

Name	City	Merit	Sal	From	To
Ed	Paris	Poor	10K	1	2
Ed	London	†	10K	3	4

binary relation on the time domain, called *time accessibility relation*, that indicates which tuple pairs must satisfy the FD.

Definition *A* time accessibility relation (*TAR*) *is a subset of* $\{(i,j) \mid 1 \le i \le j\}$. *A generalized WDFD over U is an expression* $X \to_\alpha Y$, *where X,Y* $\subseteq$ *U and α is a TAR. This generalized WDFD is satisfied by database history* $\langle I_1, I_2, I_3, ... \rangle$ *if for all* $(i,j) \in \alpha$, $s \in I_i$, $t \in I_j$, *if* $s[X] = t[X]$, *then* $s[Y] = t[Y]$.

If the TAR Next is defined by Next $= \{(1,1),(1,2),$ $(2,2),(2,3),(3,3),(3,4),...\}$, then $X \to_{\text{Next}} Y$ and $X \overset{\circ}{\to} Y$ are equivalent. TARs can also capture the notion of time granularity. For example, MonthTAR can be defined as the TAR containing (i, j) whenever $i \le j$ and time points i, j belong to the same month. Then, $Name \to_{\text{MonthTAR}} Sal$ expresses that the salary of an employee cannot change within a month.

The temporal functional dependencies proposed in [11] extend this formalism with a notion of identity, denoted by λ, similar to object-identity. The identity is time-invariant and allows to relate old and new versions of the same object. For example, *Emp*: $\lambda \to_{\text{Next}} Name$ means that the name of an employee object cannot change from one time to the next.

Trend dependencies [10,12] extend generalized WDFDs in still another way by allowing both equalities and inequalities. They can be seen as a temporal extension of the concept of *order dependency* introduced by Ginsburg and Hull [4]. For example, $(Name,=) \to_{\text{Next}} (Sal, \le)$ expresses that the salary of an employee cannot decrease. Technically, it is satisfied by a database history $\langle I_1, I_2, I_3, ... \rangle$ if for all $(i, j) \in$ Next, if $s \in I_i$ and $t \in I_j$ and $s(Name) = t(Name)$, then $s(Sal) \le t(Sal)$.

Wang et al.'s Temporal Functional Dependency [8]

The dynamic versions of FDs introduced by Vianu and Wijsen can impose constraints on tuples valid at successive time points. Wang et al.'s notion of temporal functional dependency (TFD) concentrates on temporal granularity and compares tuples valid during the same granule of some temporal granularity. The

main idea is captured by the following definitions that are simplified versions of the ones found in [8].

Definition *Assume a linear time domain* $(D, <)$. *Every nonempty subset of D is called a* granule. *Two distinct granules G_1 and G_2 are said to be* non-interleaved *if each point of either granule is smaller than all points of the other granule (i.e., either $\forall d_1 \in G_1 \forall d_2 \in G_2 (d_1 < d_2)$ or $\forall d_1 \in G_1 \forall d_2 \in G_2 (d_2 < d_1)$). A* granularity *is a set $\mathcal{G}$ of pairwise non-interleaved granules.*

Other granularity notions found in the literature often assume that granules are indexed by integers. Such index has been omitted here to simplify the formalism.

Common granularities are Month and Year. If granularity $\mathcal{G}$ is associated with temporal relation I, then all tuples of I must be timestamped by granules of $\mathcal{G}$. For example, all tuples in the following relation are timestamped by months. Practical labels, like Nov-2007, are used to denote granules of time points.

EmpMonth

Name	Sal	Position	T : Month
Ed	10K	Lecturer	Nov-2007
Ed	11K	Lecturer	Dec-2007
Ed	12K	Professor	Jan-2008

Definition *Assume a set $U = \{A_1,...,A_n\}$ of attributes and a timestamp attribute $T \notin U$. A* timestamped tuple with granularity $\mathcal{G}$ (*or simply $\mathcal{G}$-tuple*) *over U is a set* $\{A_1: c_1,...,A_n : c_n, T : G\}$, *where each c_i is a constant and $G \in \mathcal{G}$. If $t = \{A_1 : c_1,...,A_n : c_n, T : G\}$, then define* $t[U] = \{A_1 : c_1,...,A_n : c_n\}$ *and $t(T) = G$. A* timestamped relation with granularity $\mathcal{G}$ (*or simply $\mathcal{G}$-relation*) *over U is a finite set of $\mathcal{G}$-tuples over U.*

The TFD $Name \rightarrow_{Month} Sal$ expresses that the salary of an employee cannot change within a month. Likewise, $Name \rightarrow_{Year} Position$ expresses that the position of an employee cannot change within a year. Both dependencies are satisfied by the relation *EmpMonth* shown above. To check $Name \rightarrow_{Year} Position$, it suffices to verify whether for each year, the FD $Name \rightarrow Position$ is satisfied by the set of tuples whose timestamps fall in that year. For example, for the year 2007, the relation $\{t[Name, Sal, Position] \mid t \in EmpMonth, t(T) \subseteq 2007\}$ must satisfy $Name \rightarrow Position$. This is captured by the following definition.

Definition *A* temporal functional dependency (*TFD*) *over U is an expression $X \rightarrow_{\mathcal{H}} Y$, where $X, Y \subseteq U$ and $\mathcal{H}$ is a granularity. A $\mathcal{G}$-relation I over U satisfies $X \rightarrow_{\mathcal{H}} Y$ if for each granule $H \in \mathcal{H}$, the relation $\{t[U] \mid t \in I, t(T) \subseteq H\}$ satisfies the FD $X \rightarrow Y$.*

Wang et al. extend classical normalization theory to deal with data redundancy caused by TFDs. For example, since positions cannot change within a year, Ed must necessarily occupy the same position in Nov-2007 and Dec-2007. To avoid this redundancy, the information on positions must be moved into a new relation with time granularity Year, as shown above. After this decomposition, Ed's position in 2007 is only stored once.

Name	Sal	T : Month
Ed	10K	Nov-2007
Ed	11K	Dec-2007
Ed	12K	Jan-2008

Name	Position	T : Year
Ed	Lecturer	2007
Ed	Professor	2008

Constraint-Generating Dependencies [1]

Classical dependency theory assumes that each attribute of a relation takes its values in some uninterpreted domain of constants, which means that data values can only be compared for equality and disequality. The notion of *constraint-generating dependency* (CGD) builds upon the observation that, in practice, certain attributes take their values in specific domains, such as the integers or the reals, on which predicates and functions, such as $\leq$ and $+$, are defined. CGDs can thus constrain data values by formulas in the first-order theory of the underlying domain.

A *constraint-generating k-dependency* takes the following form in tuple relational calculus:

$$\forall t_1 \forall t_2 ... \forall t_k ((R_1(t_1) \wedge ... \wedge R_k(t_k) \wedge C[t_1,..., t_k]) \implies C'[t_1,..., t_k])$$

where C and C' are arbitrary constraint formulas relating the values of various attributes in the tuples $t_1,...,t_k$.

CGDs naturally arise in temporal databases: timestamp attributes take their values from a linearly

ordered time domain, possibly equipped with arithmetic. Baudinet et al. [1] specifically mention that the study of CGDs was inspired by the work of Jensen and Snodgrass on temporal specialization and generalization [5]. For example, assume a relation R with two temporal attributes, denoted VT and TT. Every tuple $t \in R$ stores information about some event, where $t(TT)$ is the (transaction) time when the event was recorded in the database, and $t(VT)$ is the (valid) time when the event happened in the real world. The following CGD expresses that every event should be recorded within c time units after its occurrence:

$$\forall t (R(t) \Longrightarrow (t(VT) < t(TT) \wedge t(TT) \leq t(VT) + c)).$$

Given appropriate first-order theories for the underlying domains, CGDs can capture the different temporal dependencies introduced above. Assume a point-stamped temporal relation $Emp(Name, Merit, Sal, VT)$ with its obvious meaning. Assume that $Name$ provides a unique and time-invariant identity for each employee. Then, the VDFD $S\check{a}l, \hat{M}erit \rightarrow \hat{S}al$ can be simulated by the following constraint-generating 4-dependency. In this formula, the tuples s and s' concern the same employee in successive database states (likewise for t and t').

$$\forall s \forall s' \forall t \forall t' \left(\left(\begin{array}{l} Emp(s) \wedge Emp(s') \\ \wedge\, s(Name) = s'(Name) \\ \wedge\, s'(VT) = s(VT) + 1 \\ \wedge\, Emp(t) \wedge Emp(t') \\ \wedge\, t(Name) = t'(Name) \\ \wedge\, t'(VT) = t(VT) + 1 \\ \wedge\, s(Sal) = t(Sal) \\ \wedge\, s'(Merit) = t'(Merit) \\ \wedge\, s(VT) = t(VT) \end{array} \right) \Rightarrow \begin{array}{l} s'(Sal) = \\ t'(Sal)) \end{array} \right)$$

The WDFD $Name, Sal \overset{\circ}{\rightarrow} Merit$ can be expressed as a constraint-generating 2-dependency:

$$\forall t \forall t' \left(\left(\begin{array}{l} Emp(t) \wedge Emp(t') \\ \wedge\, t(Name) = t'(Name) \\ \wedge\, t(Sal) = t'(Sal) \\ \wedge\, (t'(VT) = t(VT) \vee t'(VT) \\ \qquad = t(VT) + 1) \end{array} \right) \Rightarrow \begin{array}{l} t(Merit) = \\ t'(Merit) \end{array} \right)$$

Assume a binary predicate $Month$ on the time domain such that $Month(t_1, t_2)$ is true if t_1 and t_2 are two time instants within the same month. Then, $Name \rightarrow_{Month} Sal$ can be expressed as a constraint-generating 2-dependency:

$$\forall t \forall t' \left(\left(\begin{array}{l} Emp(t) \wedge Emp(t') \\ \wedge\, t(Name) = t'(Name) \\ \wedge\, Month(t(VT), t'(VT)) \end{array} \right) \Rightarrow \begin{array}{l} t(Sal) = \\ t'(Sal) \end{array} \right)$$

Dynamic Algebraic Dependencies [2]

Consider a database schema containing $Emp(Name, Merit, Sal)$ and $WorksFor(Name, Project)$. A company rule states that "*The Pulse project only recruits employees whose merit has been good at some past time.*" The relational algebra expression E_0 shown below gets the workers of the Pulse project; the expression F_0 gets the good workers. In a consistent database history, if the tuple t is in the answer to E_0 on the current database state, then t is in the answer to F_0 on some (strict) past database state.

$$E_0 = \pi_{Name}(\sigma_{Project = \text{"Pulse"}} WorksFor)$$
$$F_0 = \pi_{Name}(\sigma_{Merit = \text{"Good"}} Emp)$$

Definition *A relational algebra query E is defined as usual using the named relational algebra operators $\{\sigma, \pi, \bowtie, \rho, \cup, -\}$. A dynamic algebraic dependency (DAD) is an expression of the form EF, where E and F are relational algebra queries over the same database schema and with the same output schema. A finite database history $\langle I_0, I_1, ..., I_n \rangle$, where $I_0 = \{\}$, satisfies the DAD EF if for each $i \in \{1, ..., n\}$, for each $t \in E(I_i)$, there exists $j \in \{1, ..., i - 1\}$ such that $t \in F(I_j)$.*

The preceding definition uses relational algebra. Nevertheless, every DAD EF can be translated into temporal first-order logic in a straightforward way. Let $Now(\vec{x})$ and $Past(\vec{x})$ be safe relational calculus queries equivalent to E and F respectively, with the same free variables $\vec{x}$. Then the DAD EF is equivalent to the closed formula:

$$\forall \vec{x}(Now(\vec{x}) \Rightarrow \blacklozenge\, Past(\vec{x})).$$

It seems that the expressive power of DADs relative to other dynamic constraints has not been studied in depth. Notice that the simple DAD $\forall x(R(x) \Rightarrow \blacklozenge S(x))$

Name	Sex	Sal	Project	Day	Month	Year
Ed	M	10K	Pulse	29-Aug-2007	Aug-2007	2007
Ed	M	10K	Pulse	30-Aug-2007	Aug-2007	2007
Ed	M	10K	Pulse	31-Aug-2007	Aug-2007	2007
Ed	M	10K	Wizard	30-Aug-2007	Aug-2007	2007
Ed	M	10K	Wizard	31-Aug-2007	Aug-2007	2007
Ed	M	12K	Wizard	1-Sep-2007	Sep-2007	2007

is tuple-generating, in the sense that tuples in R require the existence of past tuples in S. The other temporal dependencies presented in this entry are not tuple-generating.

Bidoit and De Amo [2] study the existence of an operational specification that can yield all and only the database histories that are consistent. The operational specification assumes that all database updates are performed through a fixed set of update methods, called transactions. These transactions are specified in a transaction language that provides syntax for concatenation and repetition of elementary updates (insert a tuple, delete a tuple, erase all tuples).

Key Applications

An important motivation for the study of FDs in database courses is schema design. The notion of FD is a prerequisite for understanding the principles of "good" database design (3NF and BCNF). In the same line, the study of temporal functional dependencies has been motivated by potential applications in temporal database design. It seems, however, that one can go a long way in temporal database design by practicing classical, non-temporal normalization theory. As suggested in [3,8], one can put time-stamp attributes on a par with ordinary attributes, write down classical FDs, and apply a standard 3NF decomposition. For example, assume a database schema {*Name*, *Sex*, *Sal*, *Project*, *Day*, *Month*, *Year*}. The following FDs apply:

$$Name \rightarrow Sex$$
$$Name, Month \rightarrow Sal$$
$$Day \rightarrow Month$$
$$Month \rightarrow Year$$

The latter two FDs capture the relationships that exist between days, months, and years. The standard 3NF synthesis algorithm finds the following

decomposition – the last two components may be omitted for obvious reasons:

$$\{Name, Project, Day\}$$
$$\{Name, Sex\}$$
$$\{Name, Sal, Month\}$$
$$\{Day, Month\}$$
$$\{Month, Year\}$$

This decomposition, resulting from standard normalization, is also "good" from a temporal perspective. In general, the "naive" approach seems to prevent data redundancy in all situations where relationships between granularities can be captured by FDs. It is nevertheless true that FDs cannot capture, for example, the relationship between weeks and months. In particular, *Week* →*Month* does not hold since certain weeks contain days of two months. In situations where FDs fall short in specifying relationships among time granularities, there may be a need to timestamp by new, artificial time granularities in order to avoid data redundancy [8].

Cross-references

▶ Temporal Granularity
▶ Temporal Integrity Constraints

Recommended Reading

1. Baudinet M., Chomicki J., and Wolper P. Constraint-generating dependencies. J. Comput. Syst. Sci., 59(1):94–115, 1999.
2. Bidoit N. and de Amo S. A first step towards implementing dynamic algebraic dependences. Theor. Comput. Sci., 190 (2):115–149, 1998.
3. Chomicki J. and Toman D. Temporal databases. In M. Fisher, D.M. Gabbay, L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier Science, 2005.
4. Ginsburg S. and Hull R. Order dependency in the relational model. Theor. Comput. Sci., 26:149–195, 1983.
5. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 6(6):954–974, 1994.

T

6. Jensen C.S., Snodgrass R.T., and Soo M.D. Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng., 8(4):563–582, 1996.

7. Vianu V. Dynamic functional dependencies and database aging. J. ACM, 34(1):28–59, 1987.

8. Wang X.S., Bettini C., Brodsky A., and Jajodia S. Logical design for temporal databases with multiple granularities. ACM Trans. Database Syst., 22(2):115–170, 1997.

9. Wijsen J. Design of temporal relational databases based on dynamic and temporal functional dependencies. In Temporal Databases. J. Clifford A. Tuzhilin (eds.). Springer, Berlin, 1995, pp. 61–76.

10. Wijsen J. Reasoning about qualitative trends in databases. Inf. Syst., 23(7):463–487, 1998.

11. Wijsen J. Temporal FDs on complex objects. ACM Trans. Database Syst., 24(1):127–176, 1999.

12. Wijsen J. Trends in databases: Reasoning and mining. IEEE Trans. Knowl. Data Eng., 13(3):426–438, 2001.

Temporal Domain

▶ Lifespan
▶ Time Domain

Temporal Element

Christian S. Jensen[1], Richard Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Time period set

Definition

A *temporal element* is a finite union of n-dimensional time intervals. Special cases of temporal elements include *valid-time elements*, *transaction-time elements*, and *bitemporal elements*, which are finite unions of valid-time intervals, transaction-time intervals, and bitemporal intervals, respectively.

Key Points

Assuming an n-dimensional time domain, an interval is the product of n convex subsets drawn from each of the constituent dimensions.

Given a finite, one-dimensional time domain, a temporal element may be defined equivalently as a subset of the time domain. If the time domain is unbounded and thus infinite, some subsets of the time domain are not temporal elements. These subsets cannot be enumerated in finite space. For non-discrete time domains, the same observation applies.

Temporal elements are often used as timestamps. Unlike time periods, they are closed under the set theoretic operations of union, intersection, and complement, which is a very desirable property when formulating temporal database queries.

The term "temporal element" has been used to denote the concept of a valid-time interval. However, "temporal" is generally used as generic modifier, so more specific modifiers are adopted here for specific kinds of temporal elements. The term "time period set" is an early term for a temporal element. The adopted term has been used much more frequently.

Cross-references

▶ Bitemporal Interval
▶ Temporal Database
▶ Time Interval
▶ Time Period
▶ Transaction Time
▶ Time Domain
▶ Temporal Query Languages
▶ Valid Time

Recommended Reading

1. Gadia S.K. Temporal element as a primitive for time in temporal databases and its application in query optimization. In Proc. 13th ACM Annual Conf. on Computer Science, 1986, p. 413.

2. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.

3. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Evolution

▶ History in Temporal Databases

Temporal Expression

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

A *temporal expression* is a syntactic construct used, e.g., in a query that evaluates to a temporal value, i.e., an instant, a time period, a time interval, or a temporal element.

Key Points

Advanced by Gadia [1], a temporal expression is a convenient temporal query language construct.

First, any temporal element is considered a temporal expression. As Gadia uses a discrete and bounded time domain, any subset of the time domain is then a temporal expression. Next, an attribute value of a tuple in Gadia's data model is a function from the time domain to some value domain. Likewise, the attribute values of a tuple are valid during some temporal element. To illustrate, consider an (ungrouped) relation with attributes `Name` and `Position`. An example tuple in this relation is:

$$(\langle[4,17]\ Bill\rangle,$$
$$\langle[4,8]Assistant,$$
$$[9,13]\ Associate, [14,17]Full\rangle)$$

Now let X be an expression that returns a function from the time domain to some value domain, such as an attribute value, a tuple, or a relation. Then the temporal expression $[[X]]$ returns the domain of X. Using the example from above, the temporal expression $[[Position]]$ evaluates to $[4,17]$ and the temporal expression $[[Position <> Associate]]$ evaluates to $[4,8] \cup [14,17]$.

The terms "Boolean expression" and "relational expression" may be used for clearly identifying expressions that evaluate to Boolean values and relations.

Cross-references

- ▶ SQL-Based Temporal Query Languages
- ▶ Temporal Database
- ▶ Temporal Element
- ▶ Time Instant
- ▶ Time Interval
- ▶ Time Period

Recommended Reading

1. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Generalization

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Temporal generalization comes in three guises. Consider a temporal database in which data items are timestamped with valid and transaction time. Temporal generalization occurs when weakening constraints hitherto applied to the timestamps. Used in this sense, temporal generalization is the opposite of temporal specialization.

Next, a temporal relation is generalized when new timestamps are being associated with its tuples. In larger information systems where data items flow between multiple temporal relations, items may accumulate timestamps by keeping their previous timestamps and gaining new timestamps as they are entered into new temporal relations. Thus, a tuple in a particular relation has multiple timestamps: a valid timestamp, a *primary* transaction timestamp, which records when the tuple was stored in this relation, one or more *inherited* transaction timestamps that record when the tuple was stored in previous relations, and one or more additional timestamps that record when the tuple was manipulated elsewhere in the system.

Finally, a more involved notion of temporal generalization occurs when a derived relation inherits the transaction timestamps from the relation(s) it is derived from.

By describing the temporal generalization that occurs in an information system, important semantics are

captured that may be utilized for a variety of purposes. For example, a temporal relation may be queried, with specific restrictions, from a temporal relation that receives tuples with some delay from that relation. Another use is to increase the efficiency of query processing.

Key Points

The first notion of temporal generalization is simply the opposite of temporal specialization.

As an example of the second notion of temporal generalization, consider the following complex yet realistic scenario of a collection of temporal relations maintained by the transportation department of a state government. An *employee relation* is maintained on the workstation of each manager in this department, recording schedules, budgets, and salary levels for the employees under that manager. For the entire department, a single *personnel relation* is maintained on the administrative computer under the data processing group, which also maintains a *financial relation*. The bank, responsible for salary payments, maintains an *accounts relation*. Data items in the form of time-stamped tuples move from the employee relation to the personnel relation and then to the financial relation and ultimately to the accounts relation, accumulating transaction timestamps each time they enter a new database. Each timestamp has a relationship with the other transaction timestamps and with the valid timestamp. These can be stated in the schema and utilized during querying to ensure accurate results.

As an example of the third notion of temporal generalization, consider process control in a manufacturing plant. Values from sensors that capture process characteristics such as pressure and temperature may be stored in temporal relations. This data may subsequently be processed further to derive new data that capture relevant aspects of the process being monitored at a higher level of abstraction. The original data, from which the new data was derived, may be stored together with the new data, to capture the lineage, or provenance, of that data. As a result, the new data inherits timestamps from the original data.

Cross-references

► Data Stream
► Temporal Database
► Temporal Specialization

► Transaction Time
► Valid Time

Recommended Reading

1. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 5(6):954–974, December 1994.

Temporal Granularity

Claudio Bettini[1], X. Sean Wang[2], Sushil Jajodia[3]
[1]Università degli Studi di Milano, Milan, Italy
[2]University of Vermont, Burlington, VT, USA
[3]George Mason University, Fairfax, VA, USA

Synonyms

Time granularity; Temporal type

Definition

In the context of databases, a temporal granularity can be used to specify the temporal qualification of a set of data, similar to its use in the temporal qualification of statements in natural languages. For example, in a relational database, the timestamp associated with an attribute value or a tuple may be interpreted as associating that data with one or more granules of a given temporal granularity (e.g., one or more *days*). As opposed to using instants from a system-specific time domain, the use of user-defined granularities enables both more compact representations and temporal qualifications at different levels of abstraction. Temporal granularities include very common ones like *hours*, *days*, *weeks*, *months*, and *years*, as well as the evolution and specialization of these granularities for specific contexts or applications: *trading days*, *banking days*, *academic semesters*, etc.. Intuitively, a temporal granularity is defined by grouping sets of instants from a time domain into so-called *granules* in a rather flexible way with some mild conditions. For example, the granularity *business days* is defined as the infinite set of granules, each including the time instants composing one working day. A label, for example a date for a day granule, is often used to refer to a particular granule of a granularity. Answering queries in terms of a granularity different from the one used to store

data in a database is not simply a matter of syntactic granularity conversion, but it involves subtle semantics issues.

Historical Background

Temporal granularities have always had a relevant role in the qualification of statements in natural languages, and they still play a major role according to a 2006 study by Oxford University. The study includes words "day," "week," "month," and "year" among the 25 most common nouns in the English language. Temporal granularities have also been used for a long time in computer applications, including personal information management, project management, scheduling, and more. Interestingly, in many situations, their use is limited to a very few common ones, their semantics is often simplified and sometimes confusing, and their management is hard-coded in applications with ad-hoc solutions. The database community seems to be a major driver in formalizing temporal granularities. One of the earliest formalizations was proposed in [5]. At the same time the AI community was investigating formalisms to represent calendar unit systems [9,10]. In the early 1990s, the relevant role played by time granularity and calendars in temporal databases, as well as the need to devise algorithms to manage granular data, became widely recognized by the research community, and some significant progress has been made [4, 11, 13, 15]. Some support for granularities was also included in the design of the temporal query language TSQL2. A comprehensive formal framework for time granularities to be applied in several areas of database research emerged in the mid-1990s, and has been progressively refined in the following years through the investigation of its applications in data mining, temporal database design, query processing, and temporal constraint reasoning [3]. This framework is based on a set-theoretic approach (partly inspired by [5]) and on an algebraic representation, and it includes techniques to compute basic as well as more complex operations on granules and granularities. The basic notions found a large consensus in the database community [1]. The use of logic to specify formal properties and to reason about granularities as defined in the above framework was investigated in [6]. Programming oriented support for integrating multiple calendars was provided in [12]. In the logic community, an independent line of research on representation and reasoning with multiple granularities investigated classical and non-classical logic extensions based on multi-layered time domains, with applications to the specification of real-time reactive systems. This approach is extensively described in [8]. More recently, the use of automata to represent granularities and to perform basic operations on them has been proposed [7]. This work, partly inspired by previously proposed string-based representation of granularities [15], has the benefit of providing compact representation of granularities. Moreover, decision procedures for some basic problems, such as granularity equivalence and minimization, can be applied directly on that representation.

Foundations

What follows is an illustration of the main formal definitions of temporal granularities and their relationships according to the set-theoretic, algebraic approach.

Definitions

A temporal granularity can be intuitively described as a sequence of time granules, each one consisting of a set of time instants. A granule can be composed of a single instant, a set of contiguous instants (time-interval), or even a set of non-contiguous instants. For example, the `September 2008 business-month`, defined as the collection of all the business days in September 2008, can be used as a granule. When used to describe a phenomena or, in general, when used to timestamp a set of data, a granule is perceived as a non-decomposable temporal entity. A formal definition of temporal granularity is the following.

Assume a time domain T as a set of totally ordered time instants. A *granularity* is a mapping G from the integers (the *index set*) to the subsets of the time domain such that:

(1) If $i < j$ and $G(i)$ and $G(j)$ are non-empty, then each element in $G(i)$ is less than all the elements in $G(j)$.
(2) IIf $i < k < j$ and $G(i)$ and $G(j)$ are non-empty, then $G(k)$ is non-empty.

Each non-empty set $G(i)$ in the above definition is called *granule*.

The first condition in the granularity definition states that granules in a granularity do not overlap and that their index order is the same as their time domain

order. The second condition states that the subset of the index set for the granules is contiguous. Based on the above definition, while the time domain can be discrete, dense, or continuous, a granularity defines a countable set of granules; each granule is identified by an integer. The index set can thereby provide an "encoding" of the granularity in a computer. Two granules $G(i)$ and $G(j)$ are *contiguous* if there does not exist $t \in T$ such that $\forall s \in G(i)(s < t)$ and $\forall s \in G(j)(s > t)$. Independently, there may be a "textual representation" of each non-empty granule, termed its *label*, that is used for input and output. This representation is generally a string that is more descriptive than the granule's index. An associated mapping, the *label mapping*, defines for each label a unique corresponding index. This mapping can be quite complex, dealing with different languages and character sets, or can be omitted if integers are used directly to refer to granules. For example, "August 2008" and "September 2008" are two labels each referring to the set of time instants (a granule) corresponding to that month.

A granularity is *bounded* if there exist lower and upper bounds k_1 and k_2 in the index set such that $G(i) = \emptyset$ for all i with $i < k_1$ or $k_2 < i$.

The usual collections days, months, weeks and years are granularities. The granularity describing all years starting from 2000 can be defined as a mapping that takes an arbitrary index i to the subset of the time domain corresponding to the year 2000, $i + 1$ to the one corresponding to the year 2001, and so on, with all indexes less than i mapped to the empty set. The years from 2006 to 2010 can also be represented as a granularity G, with $G(2006)$ identifying the subset of the time domain corresponding to the year 2006, $G(2007)$ to 2007, and so on, with $G(i) = \emptyset$ for each $i < 2006$ and $i > 2010$.

The union of all the granules in a granularity G is called the *image* of G. For example, the image of `business-days-since-2000` is the set of time instants included in each granule representing a business-day, starting from the first one in 2000. The single interval of the time domain starting with the greatest lower bound of the image of a granularity G and ending with the least upper bound ($-\infty$ and $+\infty$ are considered valid lower/upper bounds) is called the *extent* of G. Note that many commonly used granularities (e.g., days, months, years) have their image equal to their extent, since each granule is formed by a set of contiguous elements of the time domain and

each pair of contiguous indexes is mapped to contiguous granules.

Granularity Relationships

In the following, some commonly used relationships between granularities are given.

A granularity G *groups into* a granularity H, denoted $G \trianglelefteq H$, if for each index j there exists a (possibly infinite) subset S of the integers such that $H(j) = \cup_{i \in S} G(i)$. For example, days groups into weeks, but weeks does not group into months.

A granularity G is *finer than* a granularity H, denoted $G \preceq H$, if for each index i, there exists an index j such that $G(i) \subseteq H(j)$. If $G \preceq H$, then H is *coarser than* G ($H \succeq G$).
For example, business-days is finer than weeks, while business-days does not group into weeks; business-days is finer than years, while weeks is not.

A granularity G *groups periodically into* a granularity H if:

1) $G \trianglelefteq H$.
2) There exist n, $m \in \mathbf{Z}^+$, where n is less than the number of non-empty granules of H, such that for all $i \in \mathbf{Z}$, if $H(i) = \cup_{r=0}^{k} G(j_r)$ and $H(i + n) \neq \emptyset$, then $H(i + n) = \cup_{r=0}^{k} G(j_r + m)$.

The *groups periodically into* relationship is a special case of *groups into* characterized by a periodic repetition of the "grouping pattern" of granules of G into granules of H. Its definition may appear complicated, but it is actually quite simple. Since G groups into H, any non-empty granule $H(i)$ is the union of some granules of G; for instance, assume it is the union of the granules $G(a_1)$, $G(a_2)$,...,$G(a_k)$. The periodicity property (condition 2 in the definition) ensures that the nth granule *after* $H(i)$, i.e., $H(i + n)$, if non-empty, is the union of $G(a_1 + m)$, $G(a_2 + m)$,...,$G(a_k + m)$. This results in a periodic "pattern" of the composition of n granules of H in terms of granules of G. The pattern repeats along the time domain by "shifting" each granule of H by m granules of G. Many common granularities are in this kind of relationship. For example, days groups periodically into business-days, with $m = 7$ and $n = 5$, and also groups periodically into weeks, with $m = 7$ and $n = 1$; months groups periodically into years with $m = 12$ and $n = 1$, and days groups periodically into years with $m = 14{,}697$ and $n = 400$. Alternatively, the relationship can also be

described, by saying, for example, years is periodic (or 1-periodic) with respect to months, and years is periodic (or 400-periodic) with respect to days. In general, this relationship guarantees that granularity H can be finitely described in terms of granules of G. More details can be found in [3].

Given a granularity order relationship *g-rel* and a set of granularities, a granularity G in the set is a *bottom granularity* with respect to *g-rel*, if G *g-rel* H for each granularity H in the set.

For example, given the set of all granularities defined over the time domain $(\mathbf{R};\leq)$, and the granularity relationship $\preceq$ (finer than), the granularity corresponding to the *empty* mapping is the bottom granularity with respect to $\preceq$. Given the set of all granularities defined over the time domain $(\mathbf{Z};\leq)$, and the granularity relationship $\unlhd$ (groups into), the granularity mapping each index into the corresponding instant (same integer number as the index) is a bottom granularity with respect to $\unlhd$. An example of a set of granularities without a bottom (with respect to $\preceq$ or $\unlhd$) is {weeks, months}.

Calendars are typically used to describe events or time-related properties over the same span of time using different granularities. For example, the Gregorian calendar comprises the granularities days, months, and years. Considering the notion of bottom granularity, a formal definition of calendar follows.

A *calendar* is a set of granularities that includes a bottom granularity with respect to $\unlhd$ (groups into).

Defining New Granularities through Algebraic Operators

In principle, every granularity in a calendar can be defined in terms of the bottom granularity, possibly specifying the composition of granules through the relationships defined above. Several proposals have appeared in the literature for a set of algebraic operators with the goal of facilitating the definition of new granularities in terms of existing ones. These algebras are evaluated with respect to expressiveness, user friendliness, and ability to compute operations on granularities directly on the algebraic representation. Some of the operations that are useful in applications are inter-granule conversions; for example, to compute which day of the week was the k-th day of a particular year, or which interval of days was r-th week of that year, as well as conversions involving different calendars. In the following, the main operators of one of the most expressive calendar algebras [3] are briefly described. Two operators form the backbone of the algebra:

1. The *grouping* operator systematically combines a few granules of the source granularity into one granule in the target granularity. For example, given granularity days, granularity weeks can be generated by combining 7 granules (corresponding to Monday – Sunday) week = $Group_7$(day) if we assume that day(1) corresponds to Monday, i.e., the first day of a week.

2. The *altering-tick* operator deletes or adds granules from a given granularity (via the help of a second granularity) to form a new one. For example, assume each 30 days are grouped into a granule, forming a granularity 30-day-groups. Then, an extra day can be added for each January, March, and so on, while two days are dropped from February. The February in leap years can be similarly changed to have an extra day, hence properly representing month.

Other auxiliary operators include:

– *Shift* shifts the index forward or backward a few granules (e.g., the granule used to be labeled 1 may be re-labeled 10) in order to provide proper alignment for further operations.
– *Combine* combines all the granules of one granularity that fall into (using finer-than relationship) a second granularity. For example, by combining all the business days in a week, business-week is obtained.
– *Subset* generates a new granularity by selecting an interval of granules from a given granularity. For example, choosing the days between year 2000 and 2010, leads to a granularity that only contains days in these years.
– *Select* generates new granularities by selecting granules from the first operand in terms of their relationship with the granules of the second operand. For example, selecting the first day of each week gives the Mondays granularity.

More details, including other operators, conditions of applicability, as well as comparison with other algebra proposals can be found in [3]. The algebra directly supports the inter-granule and inter-calendar conversions mentioned above. Some more complex

operations on granularities (e.g., temporal constraint propagation in terms of multiple granularities) and the verification of formal properties (e.g., equivalence of algebraic granularity representations) require a conversion in terms of a given bottom granularity. An efficient automatic procedure for this conversion has been devised and implemented [2]. The automaton-based approach may be a valid alternative for the verification of formal properties.

Key Applications

Temporal granularities are currently used in several applications, but in most cases their use is limited to very few standard granularities, supported by system calendar libraries and ad-hoc solutions are used to manipulate data associated with them. This approach often leads to unclear semantics, hidden mistakes, low interoperability, and does not take advantage of user-defined granularities and complex operations. The impact of a formal framework for temporal granularities has been deeply investigated for a number of application areas among which logical design of temporal databases, querying databases in terms of arbitrary granularities, data mining, integrity constraint satisfaction, workflows [3]. For example, when temporal dependencies in terms of granularities can be identified in the data (e.g., "salaries of employees do not change within a fiscal year"), specific techniques have been devised for the logical design of temporal databases that can lead to significant benefits. In query processing, it has been shown how to support the retrieval of data in terms of temporal granularities different from the ones associated to the data stored in the database, provided that assumptions on the data semantics are formalized, possibly as part of the database schema. Time distance constraints in terms of granularities (e.g., *Event2 should occur within two business days after the occurrence of Event1*) have been extensively studied, and algorithms proposed to check for consistency and solutions. Applications include the specification of classes of frequent patterns to be identified in time series, the specification of integrity contraints in databases, and the specification of constraints on activities durations and on temporal distance between specific events in workflows.

Temporal granularities also have several applications in other areas like natural language processing, temporal reasoning in AI, including scheduling and planning, and in computer logic, where they have been mainly considered for program specification and verification.

Future applications may include advanced personal information management (PIM). Time and location-aware devices coupled with advanced calendar applications, supporting user-defined granularities, may offer innovative personalized scheduling and alerting systems.

Experimental Results

Several systems and software packages dealing with temporal granularities have been developed, among which MultiCal, Calendar algebra implementation, GSTP Project, and TauZaman. Information about these systems can be easily found online.

Cross-references

► Temporal Constraints
► Temporal Data Mining
► Temporal Dependencies
► Temporal Periodicity
► Time Domain
► TSQL2
► Time Instant

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X. A glossary of time granularity concepts. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.), 1998, pp. 406–413.
2. Bettini C., Mascetti S., and Wang X. Supporting temporal reasoning by mapping calendar expressions to minimal periodic sets. J. Artif. Intell. Res., 28:299–348, 2007.
3. Bettini C., Wang X.S., and Jajodia S. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, Berlin Heidelberg New York, 2000.
4. Chandra R., Segev A., and Stonebraker M. Implementing calendars and temporal rules in next generation databases. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp. 264–273.
5. Clifford J. and Rao A. A simple, general structure for temporal domains. In Proc. IFIP TC 8/WG 8.1 Working Conf. on Temporal Aspects in Inf. Syst., 1987, pp. 23–30.
6. Combi C., Franceschet M., and Peron A. Representing and reasoning about temporal granularities. J. Logic Comput., 14(1):51–77, 2004.
7. Dal Lago U., Montanari A., and Puppis G. Compact and tractable automaton-based representations of time granularities. Theor. Comput. Sci., 373(1–2):115–141, 2007.
8. Euzenat J. and Montanari A. Time granularity. In M. Fisher, D. Gabbay, L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier, 2005.

9. Ladkin P. The completness of a natural system for reasoning with time intervals. In Proc. 10th Int. Joint Conf. on AI, 1987, pp. 462–467.

10. Leban B., McDonald D., and Forster D. A representation for collections of temporal intervals. In Proc. 5th National Conf. on AI, 1986, pp. 367–371.

11. Lorentzos N.A. DBMS support for nonmetric measurement systems. IEEE Trans. Knowl. Data Eng., 6(6):945–953, 1994.

12. Urgun B., Dyreson C.E., Snodgrass R.T., Miller J.K., Kline N., Soo M.D., and Jensen C.S. Integrating multiple calendars using tau-ZAMAN. Software Pract. Exp., 37(3):267–308, 2007.

13. Wang X., Jajodia S., and Subrahmanian V.S. Temporal modules: an approach toward federated temporal databases. Inf. Sci., 82:103–128, 1995.

14. Weiderhold G., Jajodia S., and Litwin W. Integrating temporal data in a heterogeneous environment. In Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993, pp. 563–579.

15. Wijsen J. A string-based model for infinite granularities. In Spatial and Temporal Granularity: Papers from the AAAI Workshop. AAAI Technical Report WS-00-08, AAAI, 2000, pp. 9–16.

Temporal Homogeneity

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Assume a temporal relation where the attribute values of tuples are (partial) functions from some time domain to value domains. A tuple in such a relation is *temporally homogeneous* if the domains of all its attribute values are identical. A temporal relation is temporally homogeneous if all its tuples are temporally homogeneous. Likewise, a temporal database is temporally homogeneous if all its relations are temporally homogeneous.

In addition to being specific to a type of object (tuple, relation, database), homogeneity is also specific to a time dimension when the time domain is multi-dimensional, as in "temporally homogeneous in the valid-time dimension" or "temporally homogeneous in the transaction-time dimension."

Key Points

The motivation for homogeneity arises from the fact that no timeslices of a homogeneous relation produce null values. Therefore, a homogeneous relational model is the temporal counterpart of the snapshot relational model without nulls. Certain data models assume temporal homogeneity, while other models do not.

A tuple-timestamped temporal relation may be viewed as a specific attribute-value timestamped relation. An attribute value a of a tuple with timestamp t is represented by a function that maps each value in t to a. Thus, models that employ tuple timestamping are necessarily temporally homogeneous.

Cross-references

▶ Lifespan
▶ SQL-Based Temporal Query Languages
▶ Temporal Database
▶ Time Domain
▶ Transaction Time
▶ Valid Time

Recommended Reading

1. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.

2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Indeterminacy

Curtis Dyreson
Utah State University, Logan, UT, USA

Synonyms

Fuzzy time; Imprecise time

Definition

Temporal indeterminacy refers to "don't know when" information, or more precisely, "don't know *exactly* when." The modifier 'temporally indeterminate' indicates that the modified object has an associated time, but that the time is not known precisely. The time when an event happens, when a time interval begins or ends, or even the duration of a period may be indeterminate. For example, the event of a car accident might be "sometime last week," the interval an airplane flight takes may be from "Friday to Saturday," or the duration a graduate student takes to write a dissertation may be "four to fifteen years."

The adjective 'temporal' allows parallel kinds of indeterminacy to be defined, such as spatial indeterminacy. There is a subtle difference between indeterminate and imprecise. In this context, indeterminate is a more general term than imprecise since precision is commonly associated with making measurements. Typically, a precise measurement is preferred to an imprecise one. Imprecise time measurements, however, are just one source of temporally indeterminate information.

Historical Background

Despite the wealth of research on adding incomplete information to databases, there are few efforts that address incomplete temporal information [6]. Much of the previous research in incomplete information databases has concentrated on issues related to null values, the applicability of fuzzy set theory, and the integration of various combinations of probabilistic reasoning, temporal reasoning, and planning.

In the earliest work on temporal indeterminacy, an indeterminate instant was modeled with a set of possible chronons [12]. Dutta next introduced a fuzzy set approach to handle events that can be interpreted to have multiple occurrences [5]. For example the event "Margaret's salary is high" may occur at various times as Margaret's salary fluctuates to reflect promotions and demotions. The meaning of "high" is incomplete, it is not a crisp predicate. In Dutta's model all the possibilities for high are represented in a *generalized* event and the user selects some subset according to his or her interpretation of "high." Generalized bitemporal elements were defined somewhat differently in a later paper by Kouramajian and Elmasri [11]. Bitemporal elements combine transaction time and valid time in the same temporal element, and can include a non-contiguous (i.e., indeterminate) set of noncontiguous possible times. Gadia et al. proposed an interesting model that intertwines support for value and temporal incompleteness [8]. By combining the different kinds of incomplete information, a wide spectrum of attribute values are simultaneously modeled, including values that are completely known, values that are unknown but are known to have occurred, values that are known if they occurred, and values that are unknown even if they occurred. Dyreson and Snodgrass proposed using probabilistic events to model temporal indeterminacy. In their model a time is represented as

a probability distribution [7]. Probabilistic times were also comprehensively addressed by Dekhtyar et al. [4].

Reasoning with incomplete information can be computationally expensive. Koubarakis was the first to focus attention on the issue of cost [9,10]. He showed that by restricting the kinds of constraints allowed in representing the indeterminacy, polynomial time algorithms can be obtained. Koubarakis proposed a temporal data model with global and local inequality constraints on the occurrence time of an event. Another constraint-based temporal reasoner is LaTeR, which has been successfully implemented [2]. LaTeR similarly restricts the kinds of constraints allowed (to conjunctions of linear inequalities), but this class of constraints includes many of the important temporal predicates [3].

Temporal indeterminacy has also been addressed in non-relational contexts, for instance in object-oriented databases [1].

Foundations

There are (at least) three possible sources of indeterminacy in a statement with respect to time: (i) a discrepancy between the *granularity* of the temporal qualification and the occurrence time; (ii) an under-specification of the occurrence time when the granularities of the temporal qualification and the occurrence time coincide; and (iii) relative times.

As a first approximation, a statement is *temporally indeterminate* if the granularity of its reference to time (in the examples, the granularity of days) is coarser than the granularity of the time at which the denoted event(s) occur. Temporal indeterminacy as well as relativity of reference to time is mainly a qualification of a statement rather than of the event it denotes (that is, temporal indeterminacy characterizes the relationship between the granularity of the time reference of a statement and the granularity of an event's occurrence time). It does not depend on the time at which the statement is evaluated. The crucial and critical point is the determination of the time granularity of the event occurrence time.

Generally, a statement whose reference to time has a granularity (e.g., days) which is temporally determinate with respect to every coarser granularity (e.g., months) and temporally indeterminate with respect to every finer granularity (e.g., seconds). But this general rule has exceptions since it does not take into account information about the denoted occurrence

time. In particular, for a *macro-event* there exists a (finest) granularity at which its occurrence time can be specified, but with respect to finer granularities, the event as a whole does not make sense, and must, if possible, be decomposed into a set of components.

But not all cases of temporally indeterminate information involve a discrepancy between the granularity of the reference to time and the granularity of the occurrence time. Consider the sentence: "The shop remained open on a Sunday in April 1990 all day long." 'Days' is the granularity of both the time reference and the occurrence time. Nevertheless, this statement is temporally indeterminate because the precise day in which the shop remained open is unknown (it is known only that it is one of the Sundays in April 1990).

Statements that contain a relative reference to time are also temporally indeterminate, but the reverse does not hold: temporally-indeterminate statements can contain relative as well as absolute references to time. The statements "Jack was killed sometime in 1990" and "Michelle was born yesterday" contain absolute and relative references to time, respectively, but they are both temporally indeterminate.

The following example illustrates how temporal indeterminacy can be represented in a relational database. Consider the employment relation shown in Fig. 1 which is cobbled together from the (somewhat hazy) memories of several employees. Each tuple shows a worker's name, salary, department, and time employed (i.e., the valid time). The first tuple represents Joe's employment in Shoes. It is temporally indeterminate since the exact day when Joe stopped working in Shoes is not known precisely; the ending valid time is recorded as sometime in January 2005 and represented as the indeterminate event "1 Jan 2005~31 Jan 2005." Joe then went to work in Admin, which is also temporally indeterminate. Joe started working in Admin "After leaving Shoes" which is a temporal constraint on an indeterminate

time. The third tuple represents Sue's employment history. She began working in Shoes sometime in the first half of January 2005 ("1 Jan 2005~15 Jan 2005") with a uniform probability for each day in that range (which is more information than is known about Joe's termination in Shoes, the probability is missing from that indeterminate event). Finally, Eve began working in Admin on some Monday in January 2005, but it is not known which Monday.

Querying temporal indeterminacy is more challenging than representing it. The two chief challenges are efficiency and expressiveness. Consider a query to find out who was employed on January 10, 2005 as expressed in a temporal version of SQL (e.g., TSQL2) below.

```
SELECT Name
FROM Employee E
WHERE VALID(E) overlaps "10 Jan 2005"
```

There are two well-defined limits on querying incomplete information: the *definite* and the *possible*. The definite answer includes only information that is known. On a tuple-by-tuple basis, determining which employment tuple definitely overlaps January 10, 2005 is straightforward: *none* definitely do. In contrast, every tuple *possibly* overlaps that day. It is efficient to compute both bounds on a tuple-by-tuple basis, but not very expressive. It would be more expressive to be able to find other answers that lie between the bounds. Probabilistic approaches seek to refine the set of potential answers by reasoning with probabilities. For instance, can it be computed who was *probably* employed (exceeding a probability of 0.5)? The probability that Sue began working before January 10, 2005 is 0.67 (the probability mass is uniformly distributed among all the possibilities). Since the probability mass function for Joe's termination in Shoes is missing, he can not be included in the employees who were probably employed.

Name	Dept.	Sal.	Valid time
Joe	Shoes	40K	[1 Jan 2003 – 1 Jan 2005~31 Jan 2005]
Joe	Admin	100K	[After leaving shoes – now]
Sue	Shoes	50K	[1 Jan 2005~15 Jan 2005 (uniform) – now]
Eve	Admin	90K	[A Monday in January 2005 – now]

Temporal Indeterminacy. Figure 1. A relation with temporal indeterminacy.

Most of the existing approaches have focused on improving the efficiency of computing probabilistic answers; the *usability* of probabilistic approaches has not yet been determined. It might be very difficult for users to interpret and use probabilities (should a user interpret "probably" as exceeding 0.75 or 0.5) so other approaches (e.g., fuzzy set approaches) may be needed to improve usability. A second research issue concerns reasoning with *intra-tuple* constraints. The definite answer given above is inaccurate. Even though it is not known exactly when Joe stopped working in Shoes or started working in Admin, it is known that he was employed in one or the other on January 10, 2005. The intra-tuple constraint in the second tuple represents this knowledge. Though intra-tuple constraints provide greater reasoning power, reasoning with them often has high computational complexity. Research continues in defining classes of constraints that are meaningful and computationally feasible. Finally, temporal indeterminacy has yet to be considered in new kinds of queries (e.g., roll-up in data warehouses and top-k queries) and new temporal query languages (e.g., τXQuery and TOWL).

Key Applications

The most common kinds of temporal indeterminacy are valid-time indeterminacy and user-defined time indeterminacy. Transaction-time indeterminacy is rarer because transaction times are always known exactly. Temporal indeterminacy can occur in logistics, especially in planning scenarios where project completion dates are typically inexact. In some scientific fields it is quite rare to know an exact time, for instance, archeology is replete with probabilistic times generated by radio-carbon dating, tree-ring analyses, and by the layering of artifacts and sediments. Indeterminacy can arise even when precise clocks are employed. In a network of road sensors, an "icy road" has an indeterminate lifetime as the change from non-icy to icy is gradual rather than instantaneous; "icy road" has a fuzzy starting and ending time.

Cross-references

▶ Probabilistic Temporal Databases
▶ Qualitative Temporal Reasoning
▶ Temporal Constraints
▶ Temporal Granularity

Recommended Reading

1. Biazzo V., Giugno R., Lukasiewicz T., and Subrahmanian V.S. Temporal probabilistic object bases. IEEE Trans. Knowl. Data Eng., 15(4):921–939, 2003.
2. Brusoni V., Console L., Terenziani P., and Pernici B. Extending temporal relational databases to deal with imprecise and qualitative temporal information. In Proc. Int. Workshop on Temporal Databases, 1995, pp. 3–22.
3. Brusoni V., Console L., Terenziani P., and Pernici B. Qualitative and quantitative temporal constraints and relational databases: theory, architecture, and applications. IEEE Trans. Knowl. Data Eng., 11(6):948–968, 1999.
4. Dekhtyar A., Ross R., and Subrahmanian V.S. Probabilistic temporal databases, I: algebra. ACM Trans. Database Syst., 26(1):41–95, 2001.
5. Dutta S. Generalized events in temporal databases. In Proc. 5th Int. Conf. on Data Engineering, 1989, pp. 118–126.
6. Dyreson C. A bibliography on uncertainty management in information systems. In Uncertainty Management in Information Systems: From Needs to Solutions. Kluwer Academic, Norwell, MA, 1997, pp. 415–458.
7. Dyreson C. and Snodgrass R.T. Supporting valid-time indeterminacy. ACM Trans. Database Syst., 23(1):1–57, 1998.
8. Gadia S.K., Nair S.S., and Poon Y.-C. Incomplete information in relational temporal databases. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 395–406.
9. Koubarakis M. Representation and querying in temporal databases: The power of temporal constraints. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 327–334.
10. Koubarakis M. The complexity of query evaluation in indefinite temporal constraint databases. Theor. Comput. Sci., 171(1–2):25–60, 1997.
11. Kouramajian V. and Elmasri R. A generalized temporal model. In Proc. Uncertainty in Databases and Deductive Systems Workshop, 1994.
12. Snodgrass R.T. Monitoring Distributed Systems: A Relational Approach. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1982.

Temporal Information Retrieval

▶ Time and Information Retrieval

Temporal Integrity Constraints

JEF WIJSEN
University of Mons-Hainaut, Mons, Belgium

Synonyms

Dynamic integrity constraints

Definition

Temporal integrity constraints are integrity constraints formulated over temporal databases. They can express dynamic properties by referring to data valid at different time points. This is to be contrasted with databases that do not store past or future information: if integrity constraints can only refer to data valid at the current time, they can only express static properties. Languages for expressing temporal integrity constraints extend first-order logic with explicit timestamps or with temporal connectives. An important question is how to check and enforce such temporal integrity constraints efficiently.

Historical Background

The use of first-order temporal logic for expressing temporal integrity constraints dates back to the early 1980s (see for example [2]). Since the late 1980s, progress has been made in the problem of checking temporal integrity [3, 9, 11] without having to store the entire database history. This entry deals with general temporal integrity constraints.

Foundations

Integrity constraints, whether they are temporal or not, are an important component of each database schema. They express properties that, ideally, must be satisfied by the stored data at all times. If a database satisfies all the integrity constraints, it is called *consistent*. Integrity constraints are commonly expressed in a declarative way using logic. Declarative integrity constraints generally do not specify how to keep the database consistent when data are inserted, deleted, and modified. An important task is to develop efficient procedures for checking and enforcing such constraints.

Temporal databases store past, current, and future information by associating time to database facts. An integrity constraint can be called "temporal" if it is expressed over a temporal database. By relating facts valid at different points in time, temporal integrity constraints can put restrictions on how the data can change over time. This is to be contrasted with databases that do not store past or future facts: if integrity constraints can only refer to a single database state, they cannot capture time-varying properties of data.

1. Defining Temporal Integrity

While temporal integrity constraints can in principle be expressed as Boolean queries in whichever temporal query language, it turns out that temporal logic on timepoint-stamped data are the prevailing formalism for defining and studying temporal integrity.

1.1 Temporal, Transition, and Static Constraints Since first-order logic is the lingua franca for expressing non-temporal integrity constraints, it is natural to express temporal integrity constraints in temporal extensions of first-order logic. Such temporalized logics refer to time either through variables with a type of time points, or by temporal modal operators, such as:

Operator	Meaning
$\bullet p$	Property p was true at the previous time instant.
$\blacklozenge p$	Property p was true sometime in the past.
$\circ p$	Property p will be true at the next time instant.
$\diamond p$	Property p will be true sometime in the future.

Satisfaction of such constraints by a temporal database can be checked if the question "Does (did/will) $R(a_1,...,a_m)$ hold at t?" can be answered for any fact $R(a_1,...,a_m)$ and time instant t. Alternatively, one can equip facts with time and ask: "Is $R(a_1,...,a_m \mid t)$ true?". Thus, one can abstract from the concrete temporal database representation, which may well contain interval-stamped facts [7]. Every time point t gives rise to a *(database) state* containing all facts true at t.

The following examples assume a time granularity of days. The facts *WorksFor*(John Smith, Pulse) and *Earns*(John Smith, 20K), true on 10 August 2007, express that John worked for the Pulse project and earned a salary of 20K at that date. The alternative encoding is *WorksFor*(John Smith, Pulse | 10 Aug 2007) and *Earns*(John Smith, 20K | 10 Aug 2007).

Although temporal integrity constraints can generally refer to any number of database states, many natural constraints involve only one or two states. In particular, *transition constraints* only refer to the current and the previous state; *static constraints* refer only to the current state.

The first-order temporal logic (FOTL) formulas (1)–(4) hereafter illustrate different types of integrity constraints. The constraint "*An employee who is dropped from the Pulse project cannot work for that project later on*" can be formulated in FOTL as follows:

$$\neg \exists x(WorksFor(x, \mathrm{Pulse}) \wedge \blacklozenge(\neg WorksFor(x, \mathrm{Pulse}) \wedge$$
$$\bullet \; WorksFor(x, \mathrm{Pulse})))$$

$$(1)$$

This constraint can be most easily understood by noticing that the subformula $\blacklozenge(\neg WorksFor(x, \mathrm{Pulse}) \wedge \bullet WorksFor(x, \mathrm{Pulse}))$ is true in the current database state for every employee x who was dropped from the Pulse project sometime in the past (this subformula will recur in the discussion of integrity checking later on). This constraint can be equivalently formulated in a two-sorted logic, using two temporal variables t_1 and t_2:

$$\neg \exists x \exists t_1 \exists t_2((t_1 < t_2) \wedge WorksFor(x, \mathrm{Pulse}|t_2)$$
$$\wedge \neg WorksFor(x, \mathrm{Pulse}|t_1) \wedge WorksFor(x, \mathrm{Pulse}|t_1 - 1))$$

The constraint "*Today's salary cannot be less than yesterday's*" is a transition constraint, and can be formulated as follows:

$$\neg \exists x \exists y \exists z(Earns(x, y) \wedge \bullet(Earns(x, z) \wedge y < z)) \quad (2)$$

Finally, static constraints are illustrated. The most fundamental static constraints in the relational data model are primary keys and foreign keys. The constraint "*No employee has two salaries*" implies that employee names uniquely identify *Earns*-tuples in any database state; it corresponds to a standard primary key. The constraint "*Every employee in the WorksFor relation has a salary*" means that in any database state, the first column of *WorksFor* is a foreign key that references (the primary key of) *Earns*. These constraints can be formulated in FOTL without using temporal connectives:

$$\forall x \forall y \forall z(Earns(x, y) \wedge Earns(x, z) \rightarrow y = z) \quad (3)$$

$$\forall x \forall y(WorksFor(x, y) \rightarrow \exists z Earns(x, z)) \quad (4)$$
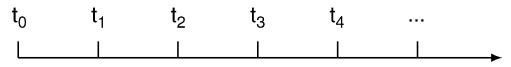
Formulas (3) and (4) show a nice thing about using FOTL for expressing temporal integrity: static constraints read as non-temporal constraints expressed in first-order logic.

1.2 Different Notions of Consistency The syntax and semantics of the temporal logic used in the previous section are defined next. Different notions of temporal constraint satisfaction are discussed.

Assume a countably infinite set **dom** of constants. In the following syntax, R is any relation name and each s_i is a variable or a constant:

$$C, C' ::= R(s_1, ..., s_m) | s_1 = s_2$$
$$| C \wedge C' | \neg C | \exists x(C) | \circ C | \diamond C | \bullet C | \blacklozenge C$$

The connectives $\bullet$ and $\blacklozenge$ are called *past operators*; $\circ$ and $\diamond$ are *future operators*. A *past formula* is a formula without future operators; a *future formula* is a formula without past operators. Other modal operators, like the past operator **since** and the future operator **until**, can be added to increase expressiveness. The set of time points is assumed to be infinite, discrete and linearly ordered with a smallest element. Thus, the time scale can be depicted as follows:

Discreteness of time is needed in the interpretation of the operators $\circ$ and $\bullet$. Formulas are interpreted relative to an infinite sequence $H = \langle H_0, H_1, H_2, ... \rangle$, where each H_i is a finite set of facts formed from relation names and constants of **dom**. Intuitively, H_i contains the facts true at time t_i. Such a sequence H is called an *infinite (database) history* and each H_i a *(database) state*. The following rules define inductively what it means for a closed formula C to be *satisfied* by H, denoted $H \models_{\mathrm{inf}} C$.

$H, i \models_{\mathrm{inf}} R(a_1, ..., a_m)$	iff	$R(a_1, ..., a_m) \in H_i$
$H, i \models_{\mathrm{inf}} a_1 = a_2$	iff	$a_1 = a_2$
$H, i \models_{\mathrm{inf}} C \wedge C'$	iff	$H, i \models_{\mathrm{inf}} C$ and $H, i \models_{\mathrm{inf}} C'$
$H, i \models_{\mathrm{inf}} \neg C$	iff	not $H, i \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \exists x(C)$	iff	$H, i \models_{\mathrm{inf}} C[x \mapsto a]$ for some $a \in$ **dom**, where $C[x \mapsto a]$ is obtained from C by replacing each free occurrence of x with a
$H, i \models_{\mathrm{inf}} \bullet C$	iff	$i > 0$ and $H, i - 1 \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \blacklozenge C$	iff	$H, j \models_{\mathrm{inf}} C$ for some j such that $0 \le j < i$
$H, i \models_{\mathrm{inf}} \circ C$	iff	$H, i + 1 \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \diamond C$	iff	$H, j \models_{\mathrm{inf}} C$ for some $j > i$

Note that the truth of each subformula is expressed relative to a *single* "reference" time point i (along with H). This characteristic allows efficient techniques for integrity checking [6] and seems crucial to the success of temporal logic. Finally:

$$H \models_{\mathrm{inf}} C \text{ iff } H, j \models_{\mathrm{inf}} C \text{ for each } j \ge 0$$

Consistency of infinite database histories is of theoretical interest. In practice, only a finite prefix of H will be known at any one time. Consider, for example, the situation where H_0 is the initial database state, and for each $i \ge 0$, the state H_{i+1} results from applying an update to H_i. Since every update can be followed by

another one, there is no last state in this sequence. However, at any one time, only some *finite history* $\langle H_0,...,H_n \rangle$ up to the most recent update is known, and it is of practical importance to detect constraint violations in such a finite history. It is reasonable to raise a constraint violation when the finite history obtained so far cannot possibly be extended to an infinite consistent history. For example, the constraints

$$\neg \exists x (Hire(x) \wedge \neg \Diamond (Promote(x) \wedge \Diamond Retire(x)))$$
$$\neg \exists x (Retire(x) \wedge \Diamond Retire(x))$$

express that all hired people are promoted before they retire, and that no one can retire twice. Then, the finite history $\langle \{Hire(\text{Ed})\}, \{Hire(\text{An})\}, \{Retire(\text{Ed})\} \rangle$ is inconsistent, because of the absence of *Promote*(Ed) in the second state. It is assumed here that the database history is append-only and that the past cannot be modified.

Two different notions, denoted $\models_{pot}$ and $\models_{fin}$, of satisfaction for finite histories are as follows:

1. $\langle H_0,...,H_n \rangle \models_{pot} C$ if $\langle H_0,...,H_n \rangle$ can be extended to an infinite history $H = \langle H_0,...,H_n, H_{n+1},... \rangle$ such that $H \models_{inf} C$.
2. $\langle H_0,...,H_n \rangle \models_{fin} C$ if $\langle H_0,...,H_n \rangle$ can be extended to an infinite history $H = \langle H_0,...,H_n, H_{n+1},... \rangle$ such that $H, i \models_{inf} C$ for each $i \in \{0, 1,...,n\}$.

Obviously, the first concept, called *potential satisfaction*, is stronger than the second one: $\langle H_0,...,H_n \rangle \models_{pot} C$ implies $\langle H_0,...,H_n \rangle \models_{fin} C$. Potential satisfaction is the more natural concept. However, Chomicki [3] shows how to construct a constraint *C*, using only past operators (● and ♦), for which $\models_{pot}$ is undecidable. On the other hand, $\models_{fin}$ is decidable for constraints *C* that use only past operators, because the extra states H_{n+1}, H_{n+2},... do not matter in that case. It also seems that for most practical past formulas, $\models_{pot}$ and $\models_{fin}$ coincide [6]. Chomicki and Niwiński [4] define restricted classes of future formulas for which $\models_{pot}$ is decidable.

1.3 Expressiveness of Temporal Constraints The only assumption about time used in the constraints shown so far, is that the time domain is discrete and linearly ordered. Many temporal constraints that occur in practice need additional structure on the time domain, such as granularity. The constraint "*The salary of an employee cannot change within a month*" assumes a grouping of time instants into months. It can be expressed in a two-sorted temporal logic extended with a built-in predicate $\text{month}(t_1, t_2)$ which is true if t_1 and t_2 belong to the same month:

$$\neg \exists x \exists y_1 \exists y_2 \exists t_1 \exists t_2 (\text{month } (t_1, t_2) \wedge Earns(x, y_1 | t_1)$$
$$\wedge Earns(x, y_2 | t_2) \wedge y_1 \neq y_2).$$

Arithmetic on the time domain may be needed to capture constraints involving time distances, durations, and periodicity. For example, the time domain 0, 1, 2,... may be partitioned into weeks by the predicate $\text{week}(t_1, t_2)$ defined by: $\text{week}(t_1, t_2)$ if $t_1 \backslash 7 = t_2 \backslash 7$, where $\backslash$ is the integer division operator. If $0 \leq t_2 - t_1 \leq 7$, then one can say that t_2 is *within a week from* t_1 (even though $\text{week}(t_1, t_2)$ may not hold).

Temporal databases may provide two temporal dimensions for valid time and transaction time. Valid time, used in the preceding examples, indicates when data are true in the real world. Transaction time records the history of the database itself. If both time dimensions are supported, then constraints can explicitly refer to both the history of the domain of discourse and the system's knowledge about that history [10]. Such types of constraints cannot be expressed in formalisms with only one notion of time.

Temporal logics have been extended in different ways to increase their expressive power. Such extensions include fixpoint operators and second-order quantification over sets of timepoints. On the other hand, several important problems, such as potential constraint satisfaction, are undecidable for FOTL and have motivated the study of syntactic restrictions to achieve decidability [4].

This entry focuses on temporal integrity of databases that use (temporal extensions of) the relational data model. Other data models have also been extended to deal with temporal integrity; time-based cardinality constraints in the Entity-Relationship model are an example.

1.4 Constraints on Interval-stamped Temporal Data All constraints discussed so far make abstraction of the concrete representation of temporal data in a (relational) database. They only assume that the database can tell whether a given fact holds at a given point in time. The notion of finite database history (let alone infinite history) is an abstract concept: in practice, all

information can be represented in a single database in which facts are timestamped by time intervals to indicate their period of validity. An interval-stamped relation is shown below.

Emp	Sal	FromTo
John Smith	10K	[1 May 2007, 31 Dec 2007]
John Smith	11K	[1 Jan 2008, 31 Dec 2008]

Following [10], constraints over such interval-stamped relations can be expressed in first-order logic extended with a type for time intervals and with Allen's interval relations. The following constraint, stating that "*Salaries of employees cannot decrease,*" uses temporal variables i_1, i_2 that range over time intervals:

$$\forall x \forall y \forall z \forall i_1 \forall i_2 (Earns(x, y|i_1) \wedge Earns(x, z|i_2) \\ \wedge \text{ before}(i_1, i_2) \Rightarrow y \leq z).$$

Nevertheless, it seems that many temporal integrity constraints can be most conveniently expressed under an abstract, point-stamped representation. This is definitely true for static integrity constraints, like primary and foreign keys. Formalisms based on interval-stamped relations may therefore provide operators like *timeslice* or *unfold* to switch to a point-stamped representation. Such "snapshotting" also underlies the *sequenced* semantics [8], which states that static constraints must hold independently at every point in time.

On the other hand, there are some constraints that concern only the concrete interval-stamped representation itself. For example, the interval-stamped relation *Earns* shown below satisfies constraint (3), but may be illegal if there is a constraint stating that temporal tuples need to be coalesced whenever possible.

Emp	Sal	FromTo
John Smith	10K	[1 May 2007, 30 Sep 2007]
John Smith	10K	[1 Oct 2007, 31 Dec 2007]
John Smith	11K	[1 Jan 2008, 31 Dec 2008]

2. Checking and Enforcing Temporal Integrity

Consider a database history to which a new state is added whenever the database is updated. Consistency is checked whenever a tentative update reaches the database. If the update would result in an inconsistent database history, it is rejected; otherwise the new database state is added to the database history. This scenario functions well if the entire database history is available for checking consistency. However, more efficient methods have been developed that allow checking temporal integrity without having to store the whole database history.

To check integrity after an update, there is generally no need to inspect the entire database history. In particular, static constraints can be checked by inspecting only the new database state; transition constraints can be checked by inspecting the previous and the new database state. Techniques for temporal integrity checking aim at reducing the amount of historical data that needs to be considered after a database update. In "history-less" constraint checking [3,9,11], all information that is needed for checking temporal integrity is stored, in a space-efficient way, in the current database state. The past states are then no longer needed for the purpose of integrity checking (though they may be needed for answering queries).

The idea is similar to Temporal Vacuuming and can be formalized as follows. For a given database schema **S**, let FIN_HISTORIES(**S**) denote the set of finite database histories over **S** and STATES(**S**) the set of states over **S**. Given a database schema **S** and a set **C** of temporal constraints, the aim is to compute a schema **T** and a computable function E : FIN_HISTORIES(**S**) $\rightarrow$ STATES(**T**), called *history encoding*, with the following properties:

- for every $H \in$ FIN_HISTORIES(**S**), the consistency of H with respect to **C** must be decidable from $E(H)$ and **C**. This can be achieved by computing a new set **C′** of (non-temporal) first-order constraints over **T** such that for every $H \in$ FIN_HISTORIES(**S**), $H \vDash_{\text{temp}} $**C** if and only if $E(H) \vDash $**C′**, where $\vDash_{\text{temp}}$ is the desired notion of temporal satisfaction (see the section on "Different Notions of Consistency"). Intuitively, the function E encodes, in a non-temporal database state over the schema **T**, all information needed for temporal integrity checking.
- E must allow an incremental computation when new database states are added: for every $\langle H_0,..., H_n \rangle \in$ FIN_HISTORIES(**S**), the result $E(\langle H_0,..., H_n \rangle)$ must be computable from $E(\langle H_0,...,H_{n-1} \rangle)$ and H_n. In turn, $E(\langle H_0,...,H_{n-1} \rangle)$ must be computable from $E(\langle H_0,...,H_{n-2} \rangle)$ and H_{n-1}. And so on.

Formally, there must be a computable function $\Delta : \text{STATES}(\mathbf{T}) \times \text{STATES}(\mathbf{S}) \rightarrow \text{STATES}(\mathbf{T})$ and an initial database state $J_{\text{init}} \in \text{STATES}(\mathbf{T})$ such that $E(\langle H_0 \rangle) = \Delta(J_{\text{init}}, H_0)$ and for every $n > 0$, $E(\langle H_0,...,H_n \rangle) = \Delta(E(\langle H_0,...,H_{n-1} \rangle), H_n)$. The state J_{init} is needed to get the computation off the ground.

Note that the history encoding is fully determined by the quartet $(\mathbf{T}, J_{\text{init}}, \Delta, \mathbf{C}')$, which only depends on $\mathbf{S}$ and $\mathbf{C}$ (and not on any database history).

Such history encoding was developed by Chomicki [3] for constraints expressed in Past FOTL (including the **since** modal operator). Importantly, in that encoding, the size of $E(H)$ is polynomially bounded in the number of distinct constants occurring in H, irrespective of the length of H. Chomicki's *bounded history encoding* can be illustrated by constraint (1), which states that an employee cannot work for the Pulse project if he was dropped from that project in the past. The trick is to maintain an auxiliary relation (call it *DroppedFromPulse*, part of the new schema $\mathbf{T}$) that stores names of employees who were dropped from the Pulse project in the past. Thus, *DroppedFromPulse(x)* will be true in the current state for every employee name x that satisfies $\blacklozenge(\neg WorksFor(x, \text{Pulse}) \wedge \bullet WorksFor(x, \text{Pulse}))$ in the current state. Then, constraint (1) can be checked by checking $\neg \exists x(WorksFor(x, \text{Pulse}) \wedge DroppedFromPulse(x))$, which, syntactically, is a static constraint. Note incidentally that the label "static" is tricky here, because the constraint refers to past information stored in the current database state. Since history-less constraint checking must not rely on past database states, the auxiliary relation *DroppedFromPulse* must be maintained incrementally (the function Δ): whenever an employee named x is dropped from the Pulse project (i.e., whenever the tuple *WorksFor(x, Pulse)* is deleted), the name x must be added to the *DroppedFromPulse* relation. In this way, one remembers who has been dropped from Pulse, but forgets when.

History-less constraint checking thus reduces dynamic to static constraint checking, at the expense of storing in auxiliary relations (over the schema $\mathbf{T}$) historical data needed for checking future integrity. Whenever a new database state is created as the result of an update, the auxiliary relations are updated as needed (using the function Δ). This technique is suited for implementation in active database systems [5,11]: a tentative database update will trigger an abort if it violates consistency; otherwise the update is accepted and will trigger further updates that maintain the auxiliary relations.

The approach described above is characterized by ad hoc updates. A different approach is operational: the database can only be updated through a predefined set of update methods (also called transactions). These transactions are specified in a transaction language that provides syntax for embedding elementary updates (insertions and deletions of tuples) in program control structures. Restrictions may be imposed on the possible execution orders of these transactions. Bidoit and de Amo [1] define dynamic dependencies in a declarative way, and then investigate transaction schemes that can generate all and only the database histories that are consistent.

Although it is convenient to use an abstract temporal representation for specifying temporal constraints, consistency checks must obviously be performed on concrete representations. Techniques for checking static constraints, like primary and foreign keys, need to be revised if one moves from non-temporal to interval-timestamped relations [8]. Primary keys can be enforced on a non-temporal relation by means of a unique-index construct. On the other hand, two distinct tuples in an interval-timestamped relation can agree on the primary key without violating consistency. For example, the consistent temporal relation shown earlier contains two tuples with the same name John Smith.

Key Applications

Temporal data and integrity constraints naturally occur in many database applications. Transition constraints apply wherever the legality of new values after an update depends on the old values, which happens to be very common. History-less constraint checking seems particularly suited in applications where temporal conditions need to be checked, but where there is no need for issuing general queries against past database states. This may be the case in monitor and control applications [12].

Cross-references
▶ Interval-based Temporal Models
▶ Point-stamped Temporal Models
▶ Temporal Constraints

Recommended Reading

1. Bidoit N. and de Amo S. A first step towards implementing dynamic algebraic dependences. Theor. Comput. Sci., 190(2):115–149, 1998.
2. de Castilho J.M.V., Casanova M.A., and Furtado A.L. A temporal framework for database specifications. In Proc. 8th Int. Conf. on Very Data Bases, 1982, 280–291.
3. Chomicki J. Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst., 20(2):149–186, 1995.
4. Chomicki J. and Niwinski D. On the feasibility of checking temporal integrity constraints. J. Comput. Syst. Sci., 51(3):523–535, 1995.
5. Chomicki J. and Toman D. Implementing temporal integrity constraints using an active DBMS. IEEE Trans. Knowl. Data Eng., 7(4):566–582, 1995.
6. Chomicki J. and Toman D. Temporal logic in information systems. In Logics for Databases and Information Systems. J. Chomicki and G. Saake (eds.). Kluwer, Dordecht, 1998, pp. 31–70.
7. Chomicki J. and Toman D. Temporal databases. In M. Fisher, D. M. Gabbay, and L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier Science, 2005.
8. Li W., Snodgrass R.T., Deng S., Gattu V.K., and Kasthurirangan A. Efficient sequenced integrity constraint checking. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 131–140.
9. Lipeck U.W. and Saake G. Monitoring dynamic integrity constraints based on temporal logic. Inf. Syst., 12(3):255–269, 1987.
10. Plexousakis D. Integrity constraint and rule maintenance in temporal deductive knowledge bases. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 146–157.
11. Sistla A.P. and Wolfson O. Temporal conditions and integrity constraints in active database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 269–280.
12. Sistla A.P. and Wolfson O. Temporal triggers in active databases. IEEE Trans. Knowl. Data Eng., 7(3):471–486, 1995.

Temporal Joins

DENGFENG GAO
IBM Silicon Valley Lab, San Jose, CA, USA

Definition

A temporal join is a join operation on two temporal relations, in which each tuple has additional attributes indicating a time interval. The temporal join predicates include conventional join predicates as well as a temporal constraint that requires the overlap of the intervals of the two joined tuples. The result of a temporal join is a temporal relation.

Besides binary temporal joins that operate on two temporal relations, there are n-ary temporal joins that operate on more than two temporal relations. Besides temporal overlapping, there are other temporal conditions such as "before" and "after" [1]. This entry will concentrate on the binary temporal joins with overlapping temporal condition since most of the previous work has focused on this kind of joins.

Historical Background

In the past, temporal join operators have been defined in different temporal data models; at times the essentially same operators have even been given different names when defined in different data models. Further, the existing join algorithms have also been constructed within the contexts of different data models. Temporal join operators were first defined by Clifford and Croker [4]. Later many papers studied more temporal join operators and the evaluation algorithms. To enable the comparison of join definitions and implementations across data models, Gao et al. [7] proposed a taxonomy of temporal joins and then use this taxonomy to classify all previously defined temporal joins.

Foundations

Starting from the core set of conventional relational joins that have long been accepted as "standard" [11]: Cartesian product (whose "join predicate" is the constant expression TRUE), theta-join, equijoin, natural join, left and right outerjoin, and full outerjoin, a temporal counterpart that is a natural, temporal generalization of the set can be defined. The semantics of the temporal join operators are defined as follows.

To be specific, the definitions are based on a single data model that is used most widely in temporal data management implementations, namely the one that timestamps each tuple with an interval. Assume that the time-line is partitioned into minimal-duration intervals, termed *chronons* [5]. The intervals are denoted by inclusive starting and ending chronons.

Two temporal relational schemas, R and S, are defined as follows.

$$R = (A_1,...,A_n, \mathrm{T}_s, \mathrm{T}_e)$$
$$S = (B_1,...,B_m, \mathrm{T}_s, \mathrm{T}_e)$$

The A_i, $1 \leq i \leq n$, and B_i, $1 \leq i \leq m$, are the *explicit attributes* that are found in corresponding snapshot schemas, and T_s and T_e are the timestamp start and end attributes, recording when the information recorded by the explicit attributes holds (or held or will hold) true. T will be used as a shorthand for the interval $[T_s, T_e]$, and A and B will be used as a shorthand for $\{A_1,...,A_n\}$ and $\{B_1,...,B_n\}$, respectively. Also, r and s are defined to be instances of R and S, respectively.

Consider the following two temporal relations. The relations show the canonical example of employees, the departments they work for, and the managers who supervise those departments.

Employee

EmpName	Dept	T
Ron	Ship	[1,5]
George	Ship	[5,9]
Ron	Mail	[6,10]

Manages

Dept	MgrName	T
Load	Ed	[3,8]
Ship	Jim	[7,15]

Tuples in the relations represent facts about the modeled reality. For example, the first tuple in the Employee relation represents the fact that Ron worked for the Shipping department from time 1 to time 5, inclusive. Notice that none of the attributes, including the timestamp attributes T, are set-valued – the relation schemas are in 1NF.

Cartesian Product

The temporal Cartesian product is a conventional Cartesian product with a predicate on the timestamp attributes. To define it, two auxiliary definitions are needed.

First, *intersect* (U,V), where U and V are intervals, returns TRUE if there exists a chronon t such that $t \in U \land t \in V$, and FALSE otherwise. Second, *overlap* (U,V) returns the maximum interval contained in its two argument intervals. If no non-empty intervals exist, the function returns. To state this

more precisely, let *first* and *last* return the smallest and largest of two argument chronons, respectively. Also let U_s and U_e denote the starting and ending chronons of U, and similarly for V.

$$overlap(U,V) =$$
$$\begin{cases} [last(U_s, V_s), first(U_e, V_e)] & \text{if } last(U_s, V_s) \\ \qquad\qquad \leq first(U_e, V_e) \\ \emptyset & \text{otherwise.} \end{cases}$$

The temporal Cartesian product, $r \times {}^T s$, of two temporal relations r and s is defined as follows.

$$r \times {}^T s = \{z^{(n+m+2)} | \exists x \in r \, \exists y \in s \, (intersect \, (x[T], y[T]) \land z[A] = x[A] \land z[B] = y[B] \land z[T] = overlap(x[T], y[T]) \land z[T] \neq \phi)\}$$

The first line of the definition ensures that matching tuples x and y have overlapping timestamps and sets the explicit attribute values of the result tuple z to the concatenation of the explicit attribute values of x and y. The second line computes the timestamp of z and ensures that it is non-empty. The *intersect* predicate is included only for later reference – it may be omitted without changing the meaning of the definition.

Consider the query "Show the names of employees and managers where the employee worked for the company while the manager managed some department in the company." This can be satisfied using the temporal Cartesian product.

Employee $\times$ ^T Manages

EmpName	Dept	Dept	MgrName	T
Ron	Ship	Load	Ed	[3,5]
George	Ship	Load	Ed	[5,8]
George	Ship	Ship	Jim	[7,9]
Ron	Mail	Load	Ed	[6,8]
Ron	Mail	Ship	Jim	[7,10]

The *overlap* function is necessary and sufficient to ensure *snapshot reducibility*, as will be discussed in detail later. Basically, the temporal Cartesian product acts as though it is a conventional Cartesian product applied independently at each point in time. When operating on interval-stamped data, this semantics corresponds to an intersection: the result will be

valid during those times when contributing tuples from *both* input relations are valid.

Theta-Join

Like the conventional theta-join, the temporal theta-join supports an unrestricted predicate *P* on the explicit attributes of its input arguments. The temporal theta-join, $r \bowtie_P^T s$, of two relations r and s selects those tuples from $r \times^T s$ that satisfy predicate $P(r[A], s[B])$. Let σ denote the standard selection operator.

The temporal theta-join, $r \bowtie_P^T s$, of two temporal relations r and s is defined as follows.

$$r \bowtie_P^T s = \sigma_{P(r[A], s[B])}(r \times^T s)$$

Equijoin

Like snapshot equijoin, the temporal equijoin operator enforces equality matching between specified subsets of the explicit attributes of the input relations.

The temporal equijoin on two temporal relations r and s on attributes $A' \subseteq A$ and $B' \subseteq B$ is defined as the theta-join with predicate $P \equiv r[A'] = s[B']$:

$$r \bowtie_{r[A']=s[B']}^T s \ .$$

Natural Join

The temporal natural join bears the same relationship to the temporal equijoin as does their snapshot counterparts. Namely, the temporal natural join is simply a temporal equijoin on identically named explicit attributes, followed by a subsequent projection operation.

To define this join, the relation schemas are augmented with explicit join attributes, C_i, $1 \leq i \leq k$, which are abbreviated by C.

$$R = (A_1, .., A_n, C_1, .., C_k, T_s, T_e)$$
$$S = (B_1, ..., B_m, C_1, ..., C_k, T_s, T_e)$$

The temporal natural join of r and s, $r \bowtie^T s$, is defined as follows.

$r \bowtie^T s = \{z^{(n+m+k+2)} | \exists x \in r \exists y \in s(x[C] = y[C] \wedge$
$z[A] = x[A] \wedge z[B] = x[B] \wedge z[C] = y[C] \wedge$
$z[T] = overlap(x[T], y[T]) \wedge z[T] \neq \phi)\}$

The first two lines ensure that tuples x and y agree on the values of the join attributes C and set the explicit attribute of the result tuple z to the concatenation of the non-join attributes A and B and a single copy of the join attributes, C. The third line computes the

timestamp of z as the overlap of the timestamps of x and y, and ensures that $x[T]$ and $y[T]$ actually overlap.

The temporal natural join plays the same important role in reconstructing normalized temporal relations as does the snapshot natural join for normalized snapshot relations [10]. Most previous work in temporal join evaluation has addressed, either implicitly or explicitly, the implementation of the temporal natural join (or the closely related temporal equijoin).

Outerjoins and Outer Cartesian Products

Like the snapshot outerjoin, temporal outerjoins and Cartesian products retain *dangling tuples*, i.e., tuples that do not participate in the join. However, in a temporal database, a tuple may dangle over a portion of its time interval and be covered over others; this situation must be accounted for in a temporal outerjoin or Cartesian product.

The temporal outerjoin may be defined as the union of two subjoins, analogous to the snapshot outerjoin. The two subjoins are the temporal left outerjoin and the temporal right outerjoin. As the left and right outerjoins are symmetric, only the left outerjoin is defined here.

Two auxiliary functions are needed. The *coalesce* function collapses value-equivalent tuples – tuples with mutually equal non-timestamp attribute values [9] – in a temporal relation into a single tuple with the same non-timestamp attribute values and a timestamp that is the finite union of intervals that precisely contains the chronons in the timestamps of the value-equivalent tuples. *(Finite unions of time intervals are termed *temporal elements* [6].)* The definition of *coalesce* uses the function *chronons* that returns the set of chronons contained in the argument interval.

$coalesce(r) = \{z^{(n+2)} | \exists x \in r(z[A] = x[A] \Rightarrow chronons$
$(x[\ T]) \subseteq z[\ T] \wedge$
$\forall x'' \in r \ (x[A] = x''[A] \Rightarrow (chronons(x''[\ T]) \subseteq z[T]))) \wedge$
$\forall t \in z[\ T] \exists x'' \in r(z[A] = x''[A] \wedge t \in chronons(x''[T]))\}$

The first two lines of the definition coalesce all value-equivalent tuples in relation r. The third line ensures that no spurious chronons are generated.

Now a function *expand* is defined that returns the set of maximal intervals contained in an argument temporal

element, *T*. Prior to defining *expand* an auxiliary function *intervals* is defined that returns the set of intervals contained in an argument temporal element.

$$intervals(T) = \{[t_s, t_e] | t_s \in T \wedge t_e \in T \wedge$$
$$\forall t \in chronons([t_s, t_e])(t \in T)\}$$

The first two conditions ensures that the beginning and ending chronons of the interval are elements of *T*. The third condition ensures that the interval is contiguous within *T*.

Using *intervals*, *expand* is defined as follows.
$$expand(T) = \{[t_s, t_e] | [t_s, t_e] \in intervals(T) \wedge$$
$$\neg \exists [t_s', t_e'] \in intervals(T)(chronons([t_s, t_e]) \subset chronons([t_s', t_e']))\}$$

The first line ensures that a member of the result is an interval contained in *T*. The second line ensures that the interval is indeed maximal.

The temporal left outerjoin is now ready to be defined. Let *R* and *S* be defined as for the temporal equijoin. $A' \subseteq A$ and $B' \subseteq B$ are used as the explicit join attributes.

The temporal left outerjoin, $r \underset{r[A']=s[B']}{\leftouter} s$ of two temporal relations *r* and *s* is defined as follows.

$r \underset{r[A']=s[B']}{\leftouter} s = \{z^{(n+m+2)} | \exists x \in coalesce(r) \exists y \in coalesce(s)$

$(x[A'] = y[B'] \wedge z[A] = x[A] \wedge z[T] \neq \phi \wedge$
$((z[B] = y[B] \wedge z[T] \in \{expand(x[T] \cap y[T])\}) \vee$
$(z[B] = null \wedge z[T] \in \{expand(x[T]) - expand(y[T])\}))) \vee$
$\exists x \in coalesce(r) \forall y \in coalesce(s)$
$(x[A'] \neq y[B'] \Rightarrow z[A] = x[A] \wedge z[B] = null \wedge$
$z[T] \in expand(x[T]) \wedge z[T] \neq \phi)\}$

The first four lines of the definition handle the case where, for a tuple *x* deriving from the left argument, a tuple *y* with matching explicit join attribute values is found. For those time intervals of *x* that are not shared with *y*, tuples with null values in the attributes of *y* are generated. The final three lines of the definition handle the case where no matching tuple *y* is found. Tuples with null values in the attributes of *y* are generated.

The temporal outerjoin may be defined as simply the union of the temporal left and the temporal right outerjoins (the union operator eliminates the duplicate equijoin tuples). Similarly, a temporal outer Cartesian product is a temporal outerjoin without the equijoin condition ($A' = B' = \phi$).

Temporal Joins. Table 1. Temporal join operators

Operator	Initial Citation	Taxonomy Operator	Restrictions
Θ-JOIN	[2]	Theta-join	None
EQUIJOIN	[2]	Equijoin	None
NATURAL-JOIN	[2]	Natural Join	None
TIME-JOIN	[2]	Cartesian Product	1
T-join	[8]	Cartesian Product	None
Cartesian product	[3]	Outer Cartesian Product	None
TE-JOIN	[13]	Equijoin	2
TE-OUTERJOIN	[13]	Left Outerjoin	2
EVENT-JOIN	[13]	Outerjoin	2
Valid-Time Theta-Join	[14]	Theta-join	None
Valid-Time Left Join	[14]	Left Outerjoin	None
GTE-Join	[15]	Equijoin	2, 3

Restrictions:

1 = restricts also the valid time of the result tuples

2 = matching only on surrogate attributes

3 = includes also intersection predicates with an argument surrogate range and a time range
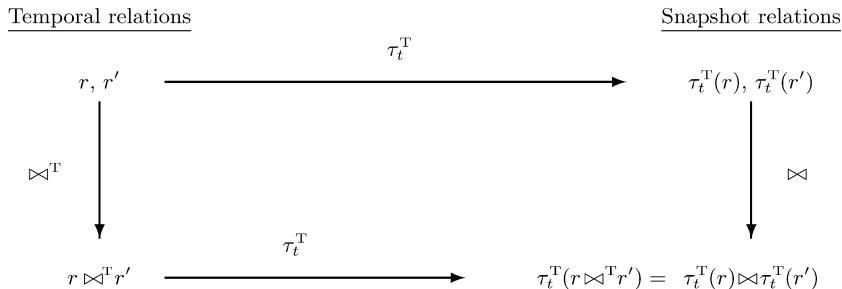
Table 1 summarizes how previous work is represented in the taxonomy. For each operator defined in previous work, the table lists the defining publication, researchers, the corresponding taxonomy operator, and any restrictions assumed by the original operators.

In early work, Clifford [4] indicated that an IN-TERSECTION-JOIN should be defined that represents the categorized non-outer joins and Cartesian products, and he proposed that an UNION-JOIN be defined for the outer variants.

Reducibility

The following shows how the temporal operators reduce to snapshot operators. Reducibility guarantees that the semantics of snapshot operator is preserved in its more complex, temporal counterpart.

For example, the semantics of the temporal natural join reduces to the semantics of the snapshot natural join in that the result of first joining two temporal

Temporal relations   Snapshot relations

Temporal Joins. Figure 1. Reducibility of temporal natural join to snapshot natural join.

relations and then transforming the result to a snapshot relation yields a result that is the same as that obtained by first transforming the arguments to snapshot relations and then joining the snapshot relations. This commutativity diagram is shown in Fig. 1 and stated formally in the first equality of the following theorem.

The timeslice operation τ^T takes a temporal relation r as argument and a chronon t as parameter. It returns the corresponding snapshot relation, i.e., with the schema of r, but without the timestamp attributes, that contains (the non-timestamp portion of) all tuples x from r for which t belongs to $x[T]$. It follows from the next theorem that the temporal joins defined here reduce to their snapshot counterparts.

Theorem 1

Let t denote a chronon and let r and s be relation instances of the proper types for the operators they are applied to. Then the following hold for all t:

$$\tau_t^T(r \bowtie^T s) = \tau_t^T(r) \bowtie \tau_t^T(s)$$
$$\tau_t^T(r \times^T s) = \tau_t^T(r) \times \tau_t^T(s)$$
$$\tau_t^T(r \bowtie_P^T s) = \tau_t^T(r) \bowtie_P \tau_t^T(s)$$
$$\tau_t^T(r \ltimes^T s) = \tau_t^T(r) \ltimes \tau_t^T(s)$$
$$\tau_t^T(r \bowtie^T s) = \tau_t^T(r) \bowtie \tau_t^T(s)$$

Due to the space limit, the proof of this theorem is not provided here. The details can be found in the related paper [7].

Evaluation Algorithms Algorithms for temporal join evaluation are necessarily more complex than their snapshot counterparts. Whereas snapshot evaluation algorithms match input tuples on their explicit join attributes,

temporal join evaluation algorithms typically must in addition ensure that temporal restrictions are met. Furthermore, this problem is exacerbated in two ways. Timestamps are typically complex data types, e.g., intervals, requiring inequality predicates, which conventional query processors are not optimized to handle. Also, a temporal database is usually larger than a corresponding snapshot database due to the versioning of tuples.

There are two categories of evaluation algorithms. Index-based algorithms use an auxiliary access path, i.e., a data structure that identifies tuples or their locations using a join-attribute value. Non-index-based algorithms do not employ auxiliary access paths. The large number of temporal indexes have been proposed in the literature [12]. Gao et al. [7] provided a taxonomy of non-index-based temporal join algorithms.

Key Applications
Temporal joins are used to model relationships between temporal relations with respect to the temporal dimensions. Data warehouses usually need to store and analyze historical data. Temporal joins can be used (alone or together with other temporal relational operators) to perform the analysis on historical data.

Cross-references
▶ Bi-Temporal Indexing
▶ Temporal Algebras
▶ Temporal Data Models
▶ Temporal Database
▶ Temporal Query Processing

Recommended Reading
 1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, November 1983.

2. Clifford J. and Croker A. The historical relational data model (HRDM) and algebra based on lifespans. In Proc. 3th Int. Conf. on Data Engineering, 1987, pp. 528–537.

3. Clifford J. and Croker A. The historical relational data model (HRDM) revisited. In Temporal Databases: Theory, Design, and Implementation, Chap. 1, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, 1993, pp. 6–27.

4. Clifford J. and Tansel A.U. On an algebra for historical relational databases: two views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 1–8.

5. Dyreson C.E. and Snodgrass R.T. Timestamp semantics and representation. Inf. Syst., 18(3):143–166, 1993.

6. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, 1988.

7. Gao D., Snodgrass R.T., Jensen C.S., and Soo M.D. Join operations in temporal databases. VLDB J., 14(1):2–29, 2005.

8. Gunadhi H. and Segev A. Query processing algorithms for temporal intersection joins. In Proc. 7th Int. Conf. on Data Engineering, 1991, pp. 336–3.

9. Jensen C.S. (ed.) The consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripadi (eds.). Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

10. Jensen C.S., Snodgrass R.T., and Soo M.D. Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng., 8(4):563–582, August 1996.

11. Mishra P. and Eich M. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, March 1992.

12. Salzberg B. and Tsotras V.J. Comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, June 1999.

13. Segev A. and Gunadhi H. Event-join optimization in temporal relational databases. In Proc. 15th Int. Conf. on Very Large Data Bases, 1989, pp. 205–215.

14. Soo M.D., Jensen C.S., and Snodgrass R.T. 1An algebra for TSQL2. In the TSQL2 Temporal Query Language, Chap. 27, R.T. Snodgrass (ed.). Kluwer, Hingham, MA, 1995, pp. 505–546.

15. Zhang D., Tsotras V.J., and Seeger B. Efficient temporal join processing using indices. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 103.

Temporal Layer

► Temporal Strata

Temporal Logic

► Time in Philosophical Logic

Temporal Logic in Database Query Languages

Jan Chomicki[1], David Toman[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]University of Waterloo, Waterloo, ON, Canada

Definition

The term "temporal logic" is used in the area of formal logic, to describe systems for representing and reasoning about propositions and predicates whose truth depends on time. These systems are developed around a set of *temporal connectives*, such as *sometime in the future* or *until*, that provide implicit references to time instants. First-order temporal logic is a variant of temporal logic that allows first-order predicate (relational) symbols, variables and quantifiers, in addition to temporal connectives. This logic can be used as a natural *temporal query language* for point-stamped temporal databases. A query (a temporal logic formula) is evaluated with respect to an *evaluation point (time instant)*. Each such point determines a specific database snapshot that can be viewed as a relational database. Thus, the evaluation of temporal logic queries resembles the evaluation of first-order (relational calculus) queries equipped with an additional capability to "move" the evaluation point using *temporal connectives*. In this way, it becomes possible to refer in a single query to multiple snapshots of a given temporal database. The answer to a temporal logic query evaluated with respect to all time instants forms a point-stamped temporal relation.

Historical Background

Temporal logic was developed (originally under the name of *tense logic* by Arthur Prior in the late 1950s), for representing and reasoning about natural languages. It was introduced to computer science by Amir Pnueli [8] as a tool for formal verification of software systems. The first proposal for using a temporal logic in the database context was by Sernadas [9]. Subsequently, de Castilho et al. [3] initiated the study of temporal-logic integrity constraints in relational databases. Tuzhilin and Clifford [12] considered temporal logic as a query language for temporal databases.

T

Foundations

Temporal Logic is a variant of *modal* logic, tailored to expressing statements whose truth is relative to an underlying time domain which is commonly a *linearly ordered* set of time instants. The modalities are expressed using natural-language statements of the form *sometime in the future*, *always in the future*, etc. and are captured in the syntax of the logic using *temporal connectives*.

Temporal logics are usually rooted in *propositional logic*. However, for the purposes of querying (single-dimensional, valid time) point-stamped temporal databases, *linear-time first-order temporal logic (FOTL)*, an extension of first-order logic (relational calculus) with *temporal connectives*, is used. More formally, given a relational schema ρ (of a snapshot of a point-stamped temporal database), the syntax of FOTL queries is defined as follows:

$$Q ::= r(x_{i_1},...,x_{i_k}) \mid x_i = x_j \mid Q \wedge Q \mid \neg Q \mid$$
$$\exists x.Q \mid Q \text{ since } Q \mid Q \text{ until } Q$$

for $r \in \rho$. The additional **since** and **until** connectives are the *temporal connectives*. The connectives completely *encapsulate* the structure of time: FOTL only refers to time *implicitly* using these connectives. In contrast, temporal relational calculus (TRC) uses *explicit* temporal variables and attributes to refer to time. Note that there are no restrictions on the nesting of the temporal connectives, the Boolean connectives, and the quantifiers.

The meaning of all temporal logic formulas is defined relative to a particular time instant called the *evaluation point*. Intuitively, the evaluation point can be viewed as representing the *current instant* or *now*. The standard first-order parts of FOTL formulas are then evaluated in the snapshot of the temporal database determined by the evaluation point. The temporal connectives make it possible to change the evaluation point, i.e., to "move" it to the past or to the future. In this way the combination of first-order constructs and temporal connectives allows to ask queries that refer to multiple snapshots of the temporal database. For example the query Q_1 **since** Q_2 asks for all answers that make Q_2 true sometime in the past and Q_1 true between then and now. Similarly, queries about the future are formulated using the **until** connective. More formally, answers to FOTL queries are defined using a *satisfaction relation* that, for a given FOTL query, links a temporal database and an evaluation point (time instant) with the valuations that make the given query true with respect to the snapshot of that database at that particular evaluation point. This definition, given below, extends the standard definition of satisfaction for first-order logic.

Definition [FOTL Semantics] Let *DB* be a point-stamped temporal database with a data domain D, a point-based time domain T_P, and a (snapshot) schema ρ.

The *satisfaction relation DB*, $\theta, t \models Q$, where Q is an FOTL formula, θ a valuation, and $t \in T_P$, is defined inductively with respect to the structure of the formula Q: as shown in Figure 2. where $\mathbf{r}_j^{DB(t)}$ is the instance of the relation r_j in the snapshot $DB(t)$ of the database *DB* at the instant t.

The answer to an FOTL query Q over DB is the set of tuples:

$$Q(DB) := \{(t, \theta(x_1),...,\theta(x_k)) : DB, \theta, t \models Q\},$$

where $x_1,...,x_k$ are the free variables of Q.

Note that answers to FOTL queries are (valid-time) point-stamped temporal relations.

Other commonly used temporal connectives, such as $\diamond$ (*sometime in the future*), $\square$ (*always in the future*), $\blacklozenge$ (*sometime in the past*), and $\blacksquare$ (*always in the past*) can be defined in terms of **since** and **until** as follows:

$$\diamond X_1 := \text{true } \textbf{until } X_1 \quad \blacklozenge X_1 := \text{true } \textbf{since } X_1$$
$$\square X_1 := \neg \diamond \neg X_1 \qquad \blacksquare X_1 := \neg \blacklozenge \neg X_1$$

For a discrete linear order, the $\circ$ (*next*) and $\bullet$ (*previous*) operators are defined as follows:

$$\circ X_1 := \text{false } \textbf{until } X_1 \quad \bullet X_1 := \text{false } \textbf{since } X_1$$

The connectives **since**, $\blacklozenge$, $\blacksquare$, and $\bullet$ are called *past temporal connectives* (as they refer to the past) and **until**, $\diamond$, $\square$, and $\circ$ *future temporal connectives*.

Example. The sensor information about who enters or exits a room is kept in the relations *Entry* (Fig. 1a) and *Exit* (Fig. 1b). Consider the query Q_a: "*For every time instant, who is in the room Bell 224 at that instant?*" It can be written in temporal logic as:

$$\exists r. ((\neg Exit(r, p)) \textbf{ since } Entry(r, p)) \wedge r = ''\text{Bell 224}''$$

The answer to Q_a in the given database is presented in Fig. 1c, under the assumption that time instants correspond to minutes.

Entry		
Time	Room	Person
8 : 30	Bell 224	John
9 : 00	Bell 224	Mary
11 : 00	Capen 10	John

Exit		
Time	Room	Person
10 : 30	Bell 224	John
10 : 00	Bell 224	Mary
12 : 00	Capen 10	John

Time	Person
8 : 31	John
...	...
10 : 30	John
9 : 01	Mary
...	...
10 : 00	Mary

a   b   c

Temporal Logic in Database Query Languages. Figure 1. The *Entry*, the *Exit*, and the *Who is in Bell 224* relations.

$DB, \theta, t \models r_j(x_{i1},...,x_{ik})$	if $(\theta(x_{i1}),...,\theta(x_{ik})) \in r_j^{DB(t)}$
$DB, \theta, t \models x_i = x_j$	if $\theta(x_i) = \theta(x_j)$
$DB, \theta, t \models Q_1 \wedge Q_2$	if $DB, \theta, t \models Q_1$ and $DB, \theta, t \models Q_2$
$DB, \theta, t \models \neg Q_1$	if not $DB, \theta, t \models Q_1$
$DB, \theta, t \models \exists x_i.Q_1$	if there is a $\in D$ such that $DB, \theta[x_i \mapsto a], t \models Q_1$
$DB, \theta, t \models Q_1$ **since** Q_2	if $\exists t_2.t_2 < t$ and $DB, \theta, t_2 \models Q_2$ and $(\forall t_1.t_2 < t_1 < t$ implies $DB, \theta, t_1 \models Q_1)$
$DB, \theta, t \models Q_1$ **until** Q_2	if $\exists t_2.t_2 > t$ and $DB, \theta, t_2 \models Q_2$ and $(\forall t_1.t_2 > t_1 > t$ implies $DB, \theta, t_1 \models Q_1)$

Temporal Logic in Database Query Languages. Figure 2. The satisfaction relation for FOTL.

Extensions

Several natural extensions of FOTL are obtained by modifying various components of the language:

Multiple temporal contexts (multi-dimensional TLs). Standard temporal logics use a single *evaluation point* to relativize the truth of formulas with respect to time. However, there is no principled reason to not use more than one evaluation point simultaneously. The logics taking this path are called *multidimensional temporal logics* or, more precisely, *n-dimensional* temporal logics (for $n \geq 1$). The satisfaction relation is extended, for a particular *n*, in a natural way, as follows:

$$DB, \theta, t_1,...,t_n \models Q$$

where $t_1,...,t_n$ are the evaluation points. In a similar fashion the definitions of *temporal connectives* are extended to this setting. Two-dimensional connectives, for example, seem to be the natural basis for temporal logic-style query language for the bitemporal data model. Unfortunately, there is no consensus on the selection of two-dimensional temporal operators.

An interesting variant of this extension are *interval temporal logics* that associate truth with *intervals* – these, however, can be considered *pairs* of time points [5,13].

More complex time domains. While most temporal logics assume that the time domain is equipped with a linear order only, in many practical settings the time domain has additional structure. For example,

there may be a way to refer to *duration* (the distance of two time points). The linear-order temporal connectives are then generalized to *metric temporal connectives*:

$DB, \theta, t \models Q_1$ **since**$_{\sim m}$ Q_2 if $\exists t_2.t - t_2 \sim m$
and
$DB, \theta, t_2 \models Q_2$
and
$(\forall t_1.t_2 < t_1 < t$ implies $DB, \theta, t_1 \models Q_1)$

$DB, \theta, t \models Q_1$ **until**$_{\sim m}$ Q_2 if $\exists t_2.t_2 - t \sim m$
and
$DB, \theta, t_2 \models Q_2$
and
$(\forall t_1.t_2 < t_1 < t$ implies $DB, \theta, t_1 \models Q_1)$

for $\sim \in \{<, \leq, =, \geq, >\}$. Intuitively, these connectives provide means of placing constraints on how far in the past/future certain subformulas must be true. The resulting logic is then the *Metric First-order Temporal Logic*, a first-order variant of *Metric Temporal Logic* [7].

Example. To demonstrate the expressive power of Metric Temporal Logic, consider the query Q_b: "*For every time instant, who has been in the room Bell 224 at that instant for at least 2 hours?*":

$$\exists r. ((\neg Exit(r, p))\ \textbf{since}_{\geq 2:00}$$
$$Entry(r, p)) \wedge r = ''Bell\ 224''$$

More powerful languages for defining temporal connectives. Another extension introduces a more powerful *language* for specifying temporal connectives over the underlying linearly-ordered time domain. Vardi and Wolper show that temporal logics with connectives defined using first-order formulas cannot express various natural conditions such as "every other day". To remedy this shortcoming, they propose temporal connectives defined with the help of *regular expressions* (ETL [15]) or *fixpoints* (temporal μ-calculus [14]). Such extensions carry over straightforwardly to the first-order setting.

More complex underlying query languages. Last, instead of relational calculus, temporal connectives can be added to a more powerful language, such as Datalog. The resulting language is called Templog [2]. With suitable restrictions, query evaluation in this language is decidable and the language itself is equivalent to Datalog$_{1S}$ [2].

Expressive Power

The **since** and **until** temporal connectives can be equivalently defined using *formulas* in the underlying theory of linear order as follows:

$$X_1 \text{ since } X_2 \quad := \quad \exists t_2.t_0 > t_2 \wedge X_2 \wedge \forall t_1$$
$$(t_0 > t_1 > t_2 \to X_1)$$
$$X_1 \text{ until } X_2 \quad := \quad \exists t_2.t_0 < t_2 \wedge X_2 \wedge \forall t_1$$
$$(t_0 < t_1 < t_2 \to X_1)$$

where X_1 and X_2 are *placeholders* that will be substituted with other formulas to be evaluated at the time instants t_1 and t_2, respectively. This observation indicates that every FOTL query can be equivalently expressed in TRC. The explicit translation parallels the inductive definition of FOTL satisfaction, uniformly parameterizing the formulas by t_0. In this way, an atomic formula $r(x_1,...,x_k)$ (where r is a non-temporal snapshot relation) becomes $R(t_0, x_1,...,x_k)$ (where R is a point-timestamped relation), and so on. For a particular t_0, evaluating $r(x_1,...,x_k)$ in $DB(t_0)$, the snapshot of the database DB at t_0, yields exactly the same valuations as evaluating $R(t_0, x_1,...,x_k)$ in DB. The embedding of the temporal connectives uses the definitions above. For example, the embedding of the **since** connective looks as follows:

$$\text{Embed}(Q_1 \text{ since } Q_2) = \exists t_2(t_0 > t_2 \wedge$$
$$\forall t_0(t_2 = t_0 \to \text{Embed}(Q_2)) \wedge$$
$$\forall t_1(t_0 > t_1 > t_2 \to \forall t_0(t_1 = t_0 \to \text{Embed}(Q_1))))).$$

Note that t_0 is the only free variable in Embed(Q_1), Embed(Q_2), and Embed(Q_1 **since** Q_2). Thus, in addition to applying the (first-order) definition of the temporal connective, the embedding performs an additional *renaming* of the temporal variable denoting the evaluation point for the subformulas, because the only free variable outside of the expanded temporal connectives must be called t_0.

Additional temporal connectives can be defined using the same approach, as formulas in the underlying theory of the temporal domain. However, for linear orders – the most common choice for such a theory – Kamp [6] has shown that all the connectives that are first-order definable in terms of linear order can be readily formulated using **since** and **until**.

In addition, it is easy to see on the basis of the above embedding that all FOTL queries (and their subqueries) define point-stamped temporal relations. This *closure property* makes FOTL amenable to specifying operators for temporal relational algebra(s) over the point-stamped temporal model. On the other hand, many, if not most, other temporal query languages, in particular various temporal extensions of SQL, are based on TRC and use temporal variables and attributes to explicitly access timestamps. These languages do not share the above closure property. Surprisingly, and in contrast to the propositional setting, one can prove that query languages based on FOTL are strictly weaker than query languages based on TRC [1,11]. The query

SNAPSHOT EQUALITY: "are there two distinct time instants at which a unary relation R contains exactly the same values?"

cannot be expressed in FOTL. On the other hand, SNAPSHOT EQUALITY can be easily expressed in TRC as follows:

$$\exists t_1, t_2.t_1 < t_2 \wedge \forall x.R(t_1, x) \Leftrightarrow R(t_2, x).$$

Intuitively, the subformula "$\forall x.R(t_1, x) \Leftrightarrow R(t_2, x)$" in the above query requires the simultaneous use of *two* distinct evaluation points t_1 and t_2 in the scope of a universal quantifier, which is not possible in FOTL. The result can be rephrased by saying that FOTL and other temporal query languages closed over the point-stamped temporal relations fail to achieve (the temporal variant of) Codd's completeness. In particular, there cannot be a temporal relational algebra over the (single-dimensional) point-stamped temporal model that can express all TRC queries.

In addition, this weakness is inherent to other languages with *implicit access* to timestamps, provided the underlying time domain is a linear order. In particular:

1. Adding a finite number of additional temporal connectives defined in the theory of linear order (including constants) is not sufficient for expressing SNAPSHOT EQUALITY [11];
2. Introducing *multidimensional temporal connectives* [4], while sufficient to express SNAPSHOT EQUALITY, is still not sufficient to express all TRC queries [10]. This also means that in the bitemporal model, the associated query languages cannot simultaneously preserve closure with respect to bitemporal relations and be expressively equivalent to TRC;
3. Using temporal connectives that are defined by fixpoints and/or regular expressions (see the earlier discussion) is also insufficient to express SNAPSHOT EQUALITY. Due to their non-first-order nature, the resulting query language(s) are incomparable, in terms of their expressive power, to TRC [1].

The only currently known way of achieving first-order completeness is based on using temporal connectives defined over a time domain whose structure allows the definition of pairing and projections (e.g., integer arithmetic). In this way temporal connectives can use pairing to simulate an unbounded number of variables and in turn the full TRC. However, such a solution is not very appealing, as the timestamps in the intermediate temporal relations do not represent time instants, but rather (encoded) tuples of such instants.

Key Applications

The main application area of FOTL is in the area of temporal integrity constraints. It is based on the observation that a sequence of relational database states resulting from updates may be viewed as a snapshot temporal database and constrained using Boolean FOTL formulas. Being able to refer to the past states of the database, temporal logic constraints generalize dynamic constraints. Temporal logic is also influential in the area of design and analysis of temporal query languages such as temporal relational algebras.

Cross-references

Recommended Reading
1. Abiteboul S., Herr L., and Van den Bussche J. Temporal versus first-order logic to query temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 49–57.
2. Baudinet M., Chomicki J., and Wolper P. Temporal deductive databases. In Temporal Databases: Theory, Design, and Implementation, Chap. 13, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. and Snodgrass (eds.). Benjamin/Cummings, Reading, MA, 1993, pp. 294–320.
3. de Castilho J.M.V., Casanova M.A., and Furtado A.L. A temporal framework for database specifications. In Proc. 8th Int. Conf. on Very Data Bases, 1982, pp. 280–291.
4. Gabbay D., Hodkinson I., and Reynolds M. Temporal Logic: Mathematical Foundations and Computational Aspects. Oxford University Press, New York, 1994.
5. Goranko V., Montanari A., and Sciavicco G. A road map of interval temporal logics and duration calculi. J. Appl. Non-Classical Logics, 14(1–2):9–54, 2004.
6. Kamp J. Tense Logic and the Theory of Linear Order. PhD Thesis, University of California, Los Angeles, 1968.
7. Koymans R. Specifying real-time properties with metric temporal logic. Real-Time Systems, 2(4):255–299, 1990.
8. Manna Z. and Pnueli A. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, Berlin, 1992.
9. Sernadas A. Temporal aspects of logical procedure definition. Inf. Syst., 5:167–187, 1980.
10. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2003, pp. 99–106.
11. Toman D. and Niwinski D. First-order queries over temporal databases inexpressible in temporal logic. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp, 307–324.
12. Tuzhilin A. and Clifford J. A temporal relational algebra as a basis for temporal relational completeness. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 13-23.
13. van Benthem J. The Logic of Time. D. Reidel, 2nd edn., 1991.
14. Vardi M.Y. A temporal fixpoint calculus. In Proc. 15th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 1988, pp. 250–259.
15. Wolper P. Temporal logic can be more expressive. Inf. Contr., 56:72–99, 1983.

T

Temporal Logical Models

ARIE SHOSHANI
Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Synonyms
Historical data models

Definition
Temporal logical models refer to the logical structure of data that captures the temporal behavior and operations over such structures. The term "logical" is used to distinguish such temporal structures form the physical storage organization and implementation. For example, the behavior of temporal events and operations over them can be described logically in a way that is independent of the physical structure (e.g., linked lists) or indexing of the events. Temporal logical models include concepts of data values that are collected or are changed over time, such as continuous physical phenomena, a series of discrete events, and interval data over time. The challenge is one of having a single comprehensive model that captures this diversity of behavior.
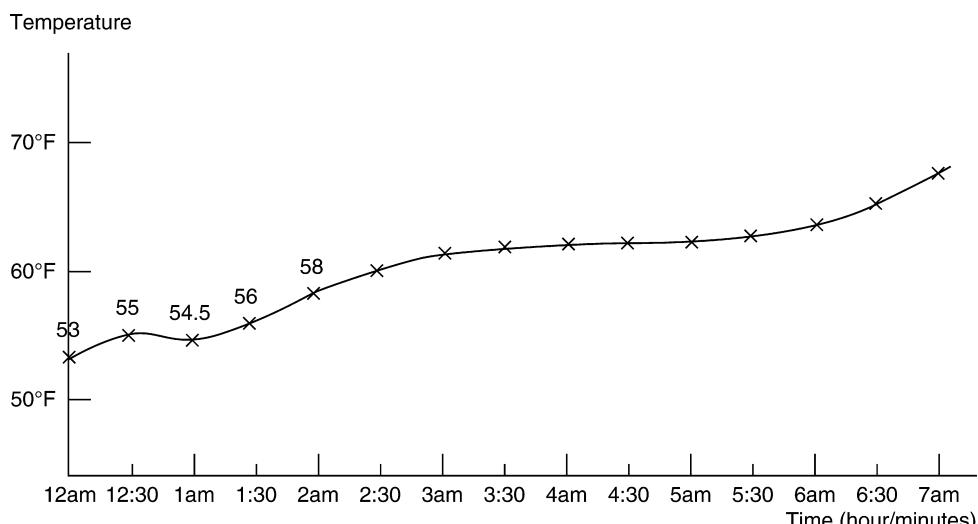
Historical Background
In the 1980s, several researchers focused on dealing with temporal data, both on the modeling concepts and on physical organization and indexing of temporal data. This led to the temporal database field to be established, and several books were written or edited on the subject (for example [4,5,12]). Since then, the subject continues to appear in specific application domains, or in combination with other concepts, such as spatio-temporal databases, and managing streaming data.

Foundations

The Treatment of Time in Database Systems
Time is a natural part of the physical world and an indispensable part of human activity, yet many database models treat temporal behavior as an afterthought. For example, weather (temperature, clouds, and storms) is a continuous phenomenon in time, yet it is treated as discrete events per day or per hour. In contrast, some human activities are fundamentally discrete events, such as salary which may change annually, but are treated as continuous concepts, where the salary is the same for the duration between the discrete events of salary changes. The main reason for the inconsistent treatment of time is that temporal objects and their semantics are not explicit in the data model. Consider for example, temperature measurements at some weather station as shown in Fig. 1. These are represented in conventional database systems (such as relational data models) as a two-part concept of time-of-measurement and value-of-measurement attributes, but the fact that the measurements are taken at evenly spaced intervals (e.g., every half an hour) and that the temperature represents a continuous

Temporal Logical Models. Figure 1. Continuous behavior of temperature measurements.

phenomenon is not captured. Consequently, if one asks what the temperature was at 12:35A.M, no such value exists. Furthermore, the interpolation function associated with getting this value is unknown. It could be a simple weighted averaging of the two nearest values, or a more sophisticated curve interpolation function.

Temporal Data Behavior

Temporal logical models are models designed to capture the behavior of temporal data sequences. First, some examples that illustrate the concepts that need to be captured by the model are presented.

Example 1: wind velocity. Usually, the measurements of wind velocity are taken by devices at regular time periods, for example every hour. These are referred to as "time series." In this example, the measured quantity is not a single value, but has a more complex structure. It measures the direction of the wind and the velocity of the wind, which can be represented as a three-dimensional vector. The measured phenomenon is continuous, of course, but for this application it is determined by the database designers that a certain time granularity for queries is desired, such as values by minutes. Since the values are collected only hourly, an interpolation function must be provided and associated with this time sequence. The behavior is similar to the temperature behavior shown in Fig. 1, except that the measured values are three-dimensional vectors for each time point.
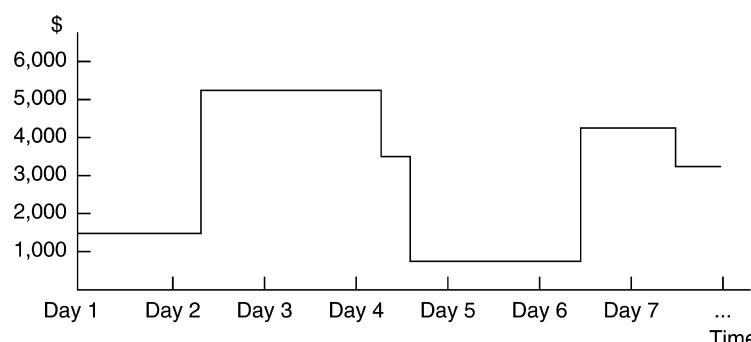
Example 2: bank account. The amount of money in the bank account changes when transactions take place. Money can be added or taken out of the account at irregular times. The value of the account is the same for the duration of time between transactions. This is shown in Fig. 2, where the granularity of the time points is in minutes. Note that the days shown should have precise dates in the database. Another aspect in this example is that in the case of a deposit of a check, funds may not be available until the check clears. Thus, there are two times associated with the deposit, the time of the transaction, and the time when funds are made available.
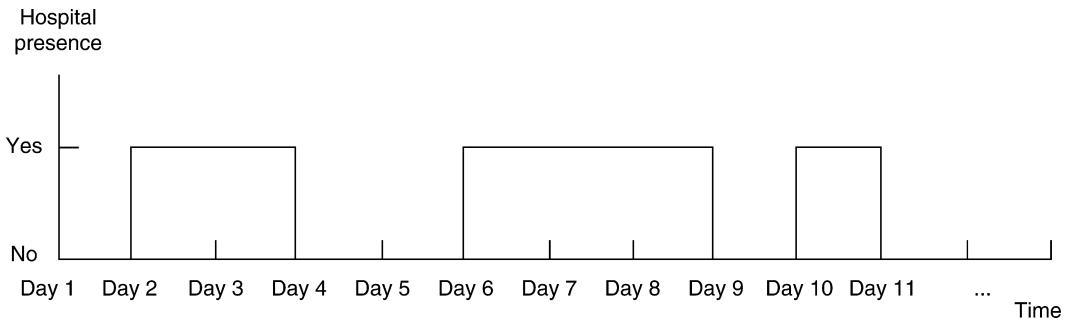
Example 3: hospitalization visits. Hospital visits of an individual occur typically at irregular times, and each can last a period of time, usually measured in days. The value associated with the hospital visit time sequence is Boolean; that is, only the fact that a visit took place or did not. This is an example where the concept of an interval must be represented in the data model. This is shown in Fig. 3, where the granularity is a day, and the interval durations span days. Here again, the days shown will have to have precise dates in the database.

Example 4: store revenue. Suppose that a store owner wishes to keep in a database the total revenue per day. The time sequence is regular, i.e., a time series (not counting days when the store is closed). The values per day do not represent continuous phenomena, but rather they are discrete in time, collected every day at the end of that day. This is the same as representing discrete events, such as the time of an accident, etc. In general, it does not make sense to apply interpolation to such a time sequence. However, if some data are missing, an interpolation rule could be used to infer the missing values. This is shown in Fig. 4. This is a time series, because only the days of business are shown (Monday – Friday).
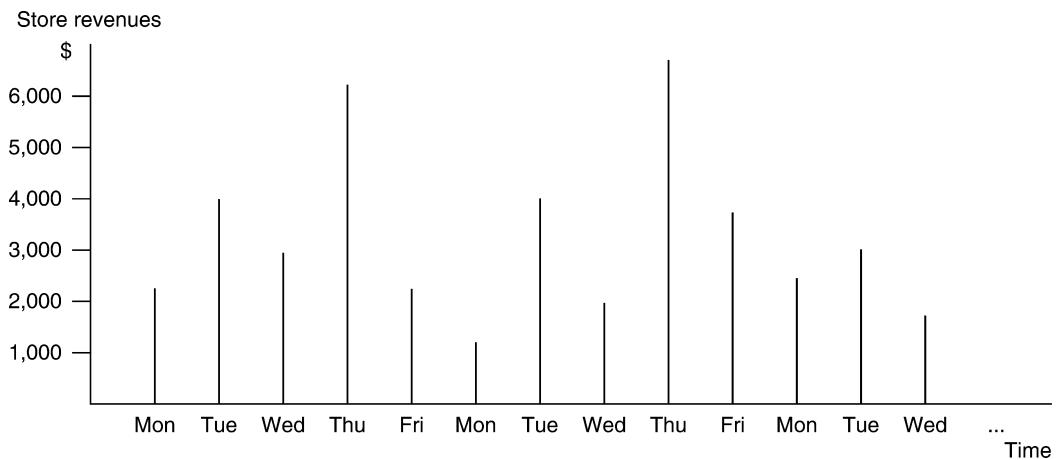
In the example above, only a single time sequence is shown, but there could be a collection of related time sequences. For example, time sequences of the quantity of each item sold in a store. For each item, there is a

Temporal Logical Models. Figure 2. Step-wise constant behavior of a bank account.

Hospital
presence

Temporal Logical Models. Figure 3. Interval behavior of hospital visits.

Store revenues

Temporal Logical Models. Figure 4. Discrete behavior of store revenues.

time sequence. However, all time sequences in this case have the same behavior, and they are collected in tandem per day. Such groups of related time sequences are referred to as "time sequence collections" [11]. As is discussed later, this concept is important for operations performed over collections of time sequences.

Behavioral Properties of Temporal Sequences
As is evident from the above examples, there are certain properties that can be specified to capture the logical behavior of temporal sequences. If such properties were supported by database systems, it is only necessary in such systems to store temporal data as time-value pairs in the general case, or simply ordered sequences of values for time series. These properties are discussed next, along with the possible category values they can assume.

Time-granularity: value and unit
The time-granularity indicates the time points for which data values can exist in the model. It is the smallest unit of time measure between time points in the time sequence. For example, if deposits and withdrawals to a bank account can be recorded with a minute precision, then the time granularity is said to be a minute. However, in cases where data values can be interpolated, an interpolation-granularity needs to be specified as well. For example, the temperatures shown in Fig. 1 are recorded every half an hour, and therefore the time granularity is 30 minutes, but given that values can be interpolated up to a minute precision, it is necessary to specify that the interpolation-granularity is a minute. This point is discussed further below in the section on "interpolation rule." Note that for regular time sequences (time

series), it is often necessary to describe the origin of the time sequence, and the time granularity is relative to that origin. A formal treatment of time granularity can be found in [1].

1) Regularity: regular (time series), irregular

As mentioned above time series are regular sequences. They can be described by specifying the "time-step" between the time points. The time-step together with the "life span" (described next) specify fully the time points for which data values are expected to be provided. Because of its regular nature, it is not necessary to store the time points in the databases – these can be calculated. However, this is a physical implementation choice of the system, and the time values can be stored explicitly to provide faster access. Time series are prevalent in many applications, such as statistics gathering, stock market, etc.

Irregular time sequences are useful for event data that occurs in unpredictable patterns, such as bank withdrawals, visits to the library, or making a phone call. A typical phenomena in such sequences, is that most of the time points have no values associated with them. For example, suppose that the time granularity for recording phone calls is a minute. The time sequence of phone calls will typically be mostly empty (or null). For this reason, irregular time sequences are usually represented as time-value pairs in the database.

Life span: begin-time, end-time

The life span indicates for what period of time the time sequence is valid. The begin-time is always necessary, and has to be specified with the same precision of the time granularity. For example, if for the temperature time series in Fig. 1, the begin-time was January 1, 1:15 A.M, and the granularity was 30 minutes, then the time points will be 1:15 A.M, 1:45 A.M, 2:15 A.M, etc.

The life span end-time can be specified as "open-ended." That means that this time series is active.

Behavior type: continuous, step-wise-constant, interval, discrete.

These types were illustrated in the examples of the previous section. For example 1, on wind velocity, the type is continuous. For example 2, the amount available in a bank account, the type is step-wise-constant. For example 3, of hospital visits, the type is interval. For example 4, the store revenues per day, the type is discrete. Note that the interval type can be considered as a special case of step-wise-constant type having the Boolean values (0 or 1). Another thing worth noting is

that discrete time sequences cannot be associated with an interpolation rule. The interpolation rules for the other types are discussed next.

Interpolation rule: interpolation-granularity, interpolation-function.

The interpolation-granularity has to be specified in order for the underlying system to enforce the data points for which interpolation can be applied in response to queries. Note that the interpolation-granularity has to be in smaller units than the time-granularity, and the number of interpolation-granularity points in a time-granularity unit must be an integer. For example, while temperature in the example of Fig. 1 has time-granularity of 30 minutes, the interpolation-granularity can be 5 minutes.

The interpolation-function for the step-wise-constant and interval types are straightforward, and are implied by the type. But, for a continuous type, an interpolation-function must be specified. It should be possible to provide the system with such a function for each continuous time sequence. If no interpolation-function is provided, the system can use a default function.

Value type: binary, numeric, character, multi-valued, etc.

This property of temporal sequences is no different from specifying attribute types in conventional database systems. The case of a binary type is special to interval events, and is not always supported by conventional system. Also, multi-valued or multi-component attributes are special requirements for more sophisticated time sequences that exist in scientific data, such as the wind velocity in example 2.

Transaction vs. valid time: transaction, valid

Similar to the bank account in example 2 where the deposit time was different from the time when funds are available, there are many examples where temporal data are recorded in the database before the data values are valid. This concept was explored extensively in [13], and referred to as "transaction time" and "valid time." Actually, there are situations where the transaction time can occur after the valid time for retroactive cases. For example, a salary raise can be added to a database in March of some year, but is retroactive to January of that year. This concept has led to an extensive literature on representing it as an extension of query languages, including a temporal extension to the SQL query language, referred to as TSQL [12]. It is worth noting that other concepts of multiple

temporal dimensions were also introduced in the literature, in addition to transaction and valid times. For example, [3] introduced the concept of "event time" – times of the events that initiates and terminates the interval validity, and "availability time" – the time interval during which facts are available.

If multiple temporal dimensions are needed in the model, they can be thought of as multiple correlated time sequences. However, in general, each time dimension can have different properties. For example, the transaction time sequence for bank deposits can have a granularity of a minute, while the valid time for the available funds can be daily.

Operation over Temporal Data

Because of the inherent time order of temporal data, operations over them, such as "when," "preceding," "following," etc. are based on the time order. Similarly, the concept of a "time window" is natural. Various researchers have developed precise semantics to query languages by adding temporal operators to existing query languages, including relational query languages, such as SQL, relational algebras, such as QUEL, functional query languages, such as DAPLEX, deductive query languages, such a Datalog, and entity-relationship languages. Many such examples can be found in the books on temporal databases [4,5]. In order to explain the various operators, they are classified into the following four categories.

Predicate Operators Over Time Sequences Predicate operators refer to either specific times or a time interval. For specific times, the obvious predicates include "before," "after," "when," etc. But, in addition, there are operators that refer to the life span, such as "first, and "last." For time intervals, operators such as "during" or "interval" are used. Also, when specifying an interval, the keyword "now" is used to refer to time sequences that are active, such as "interval" (Jan 01, 2007, now). Note that the time values used must be expressed at the granularity of the time sequence (or the interpolation-granularity if interpolation is allowed). In some time sequences, it is useful to use an integer to refer to the nth time instance, such as t-33 to mean the 33rd time point in the time sequence. However, this is not included in most proposed query languages.

Another purpose of predicate operators is to get back the time periods where some condition on the data values hold. For example, suppose that a time sequences represents temperature at some location. The query "get periods where temperature >100" (the units are Prof.F) will return a (Boolean) interval time sequence, where the temperature was greater than 100. Note that "periods" is treated as a keyword.

Aggregation Operators Over Time Windows The usual statistical operators supported by database systems (sum, count, average, min, max) can be applied to a specific time window (t_begin, t_end), to the entire time sequence (first, last), or to the combinations (first, t_end), (t_begin, last). In general, "first" or "last" can be substituted by an instance number, such as "t-33" mentioned above. Here, again, the time has to be specified in the granularity of the time sequence.

Another way to apply operators over windows is to combine that with the "group by" concept. This is quite natural for temporal sequence that involve calendar concepts of month, day, minute, second, etc. For example, suppose that a time sequence represents daily sales. One can have a query "sum sales by month." This is the same as asking for multiple windows, each over the days in each month.

Aggregation Operators Over Time Sequence Collections In a typical database, time sequences are defined over multiple object instances. For example, one can have in an organization the salary history of all of its employees. Asking for the "average salary over all employees" over time requires the average operation to be applied over the entire collection of time sequences. This operation is not straight forward if all salary raises do not occur at the same time. This operation will generate a time sequence whose time points are the union of all the time points of the time sequences, where the average values are performed piecewise on the resulting intervals.

Similar to the case of aggregation over time windows, where the "group by" operation can be applied, it is possible to group by object instances in this case. For example, if employees are organized by departments, one can ask for "average salary over all employees per department."

Composition of Time Sequences Composition refers to algebraic operations over different time sequences. For example, suppose that in addition to salary history

recorded for each employee in an organization, the history of commissions earned is recorded. In order to obtain "total income," the salary and the commission time sequences have to be added for each employee. This amounts to the temporal extension of algebraic operations on multiple attributes in non-temporal query languages.

Combinations of the Above Operators   It is conceptually reasonable to combine the above operators in query languages. For example, it should be possible to have the aggregation and composition operators applied only to a certain time window, such as getting the "average salary over all employees for the last three years." Furthermore, it should be possible to apply a temporal operator to the result of another temporal operator. This requires that the result of operations over time sequences is either a time sequence or a scalar. If it is a time sequence, temporal operators can be applied. If it is a scalar (a single value) it can be applied as a predicate value. This requirement is consistent with other languages, such as the relational language, where the operation on relations always generates a relation or a scalar.

Additional Concepts   There are many concepts introduced in the literature that capture other logical aspects of temporal operations and semantics. This broad literature cannot be covered here; instead, several concepts are mentioned next. Temporal specialization and generalization are explored in [6]. Unified models for supporting point-based and interval-based semantics are developed in [2,14]. It is argued in [8] that temporal data models have to include explicitly the concept of ordered data, and a formal framework for that is proposed. A single framework for supporting both time series and version-based temporal data are developed in [9]. There are also many papers that discuss how to efficiently support temporal operations (such as aggregation), see for example [7,10]. Finally, in the context of streaming data, temporal aggregation operations on time windows have in explored – see entries on "stream data management" and "stream mining."

Key Applications

Temporal data are ubiquitous. It naturally exists in applications that have time series data, such as stock market historical data, or history of transactions in bank accounts. In addition, it is a basic requirement of scientific databases collecting data from instruments or performing simulation over time steps. In the past, many databases contained only the current (most updated) data, such as current salary of employees, current inventories in a store or a warehouse, etc. The main reason for that was the cost of storage and efficiency of processing queries. One could not afford keeping all the historical data. More recently, as the cost of storage is plummeting, and compute engines are faster and can operate in parallel, historical data are routinely kept. While it is still worth keeping a version of current data for some applications for efficiency of access, many applications now use historical data for pattern and trend analysis over time, especially in data warehouse applications.

Future Directions

While a lot of research was done on temporal data, the concepts and operations over such data are only partially supported, if at all, in commercial and open source database system. Some support only the concept of date_time (it is complex enough, crossing time zones and century boundaries), but the support for properties of time sequences and operations over them are still not generally available. Building such database systems is still a challenge.

Cross-references

▶ Data Models
▶ Data Warehouse
▶ Database Design
▶ Query Language
▶ Spatial Network Databases
▶ Stream Data Analysis
▶ Stream Mining

Recommended Reading

1. Bettini C, Wang X.S., and Jajodia S., A general framework for time granularity and its application to temporal reasoning. Ann. Math. Artif. Intell., 22(1–2):29–58, 1998.
2. Chen C.X., and Zaniolo C. Universal temporal extensions for database languages. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 428–437.
3. Combi C., and Montanari A. Data models with multiple temporal dimensions: completing the picture. In Proc. 13th Int. Conf. on Advanced Information Systems Eng., 2001, pp. 187–202.

4. Etzion O, Jajodia S, and Sripada S.M., (eds.). Temporal Databases: Research and Practice. Springer, Berlin Heidelberg, 1998.
5. Tansel A.U., Clifford J., Gadia S.K., Jajodia S., Segev A., and Snodgrass R.T. Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Redwood City, CA, 1993.
6. Jensen C.S., and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 6(6):954–974, 1994.
7. Kang S.T., Chung Y.D., and Kim M.-Y., An efficient method for temporal aggregation with range-condition attributes. Inf. Sci., 168(1–4):243–265, 2004.
8. Law Y.N., Wang H., and Zaniolo C. Query languages and data models for database sequences and data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 492–503.
9. Lee J.Y., Elmasri R., and Won J. An integrated temporal data model incorporating time series concept. Data Knowl. Eng., 24 (3):257–276, 1998.
10. Moon B., López I.F.V., and Immanuel V. Efficient algorithms for large-scale temporal aggregation. IEEE Trans. Knowl. Data Eng., 15(3):744–759, 2003.
11. Segev A, and Shoshani A. Logical modeling of temporal data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 454–466.
12. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer, Norwell, MA, 1995.
13. Snodgrass R.T., and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.
14. Terenziani P., and Terenziani R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a treatment of the Telic/Atelic distinction. IEEE Trans. Knowl. Data Eng., 16(5):540–551, 2004.

Temporal Middleware

► Temporal Strata

Temporal Object-Oriented Databases

Carlo Combi
University of Verona, Verona, Italy

Definition

In a strict sense, a temporal object-oriented database is a database managed by an object-oriented database system able to explicitly deal with (possibly) several temporal dimensions of data. The managed temporal dimensions are usually valid and/or transaction times. In a wider sense, a temporal object-oriented database is a collection of data having some temporal aspect and managed by an object-oriented database system.

Historical Background

Research studies on time, temporal information, and object-oriented data started at the end of 1980s and continued in the 1990s. From the seminal work by Clifford and Croker on objects in time [4], several different topics have been discussed. Segev and Rose studied both the modeling issues and the definition of suitable query languages [12]; Wuu and Dayal showed how to use an object-oriented data model to properly represent several temporal aspects of data [14]; Goralwalla et al. studied the adoption and extension of an object-oriented database system, TIGUKAT, for managing temporal data, and finally proposed a framework allowing one to classify and define different temporal object-oriented data models, according to the choices adopted for representing time concepts and for dealing with data having temporal dimensions [8]. Schema evolution in the context of temporal object-oriented databases has been studied by Goralwalla et al. and by Edelweiss et al. [6,9]. Bertino et al. studied the problem of formally defining temporal object-oriented models [1] and, more recently, they proposed an extension of the ODMG data model, to deal with temporalities [2]. Since the end of 1990s, research has focused on access methods and storage structures, and on temporalities in object relational databases, which, in contrast to object-oriented databases, extend the relational model with some object-oriented features still maintaining all the relational structures and the related, well studied, systems. As for the first research direction, for example, Norvag studied storage structures and indexing techniques for temporal object databases supporting transaction time [10]. Recently, some effort has been done on extending object-relational database systems with some capability to deal with temporal information; often, as in the case of the work by Zimanyi and colleagues, studies on temporalities in object-relational databases are faced in the context of spatio-temporal databases, where temporalities have to be considered together with the presence of spatial information [15]. In these years, there were few survey papers on temporal object-oriented databases, even though they were considered as a part of survey papers on temporal databases [11,13]. Since the 1990s, temporal object-oriented databases have also been studied in some more application-oriented research efforts: among them are temporal object models and languages for multimedia and for medical data, as, for example, in [5,7].

Foundations

Object-oriented (OO) methodologies and technologies applied to the database field have some useful features – abstract data type definition, inheritance, complex object management – in modeling, managing, and storing complex information, as that related to time and to temporal information. Objects and times have, in some sense, a twofold connection: from a first point of view, complex, structured types could be suitably defined and used to manage both complex time concepts and temporal dimensions of the represented information; on the other side, object-oriented data models and languages may be suitably extended with some built-in temporal features. In the first case, the object-oriented technology is performed to show that complex concepts, as those related to time and temporal information, can be suitably represented and managed through types, generalization hierarchies, and so on. In the second case, the focus is slightly different and is on extending object-oriented models and languages to consider time-related concepts as first-class citizens: so, for example, each object, besides an object identifier, will have a lifespan, managed by the system.

Even though different approaches and proposals in the literature do not completely agree on the meaning of different terms used by the "object-oriented" community, there is a kind of agreement on the basic concepts of any object-oriented data model. An *object* may represent any entity of the real world, e.g., a patient, a therapy, a time interval. The main feature of an object is its *identity*, which is immutable, persists during the entire existence of the object, and is usually provided by the database system through an identifier, called OID (Object IDentifier). An object is characterized by a state, described by properties (*attributes* and *associations* with other objects) and by a behavior, defined by *methods*, describing modalities, by which it is possible to interact with the object itself. Objects are created as instances of a *type*; a *class* is the collection (also named *extent*) of all the objects of a given type stored into the database at a certain moment. A *type* describes (i) the structure of properties through attributes and associations, and (ii) the behavior of its instances through methods applicable to objects instances of that type.

To introduce the faced topics, a simple clinical database is used, storing data on symptoms and therapies of patients and will graphically represent its schema through the well-known UML notation for class diagrams, adopting the primitive types provided by the ODMG data model [3].
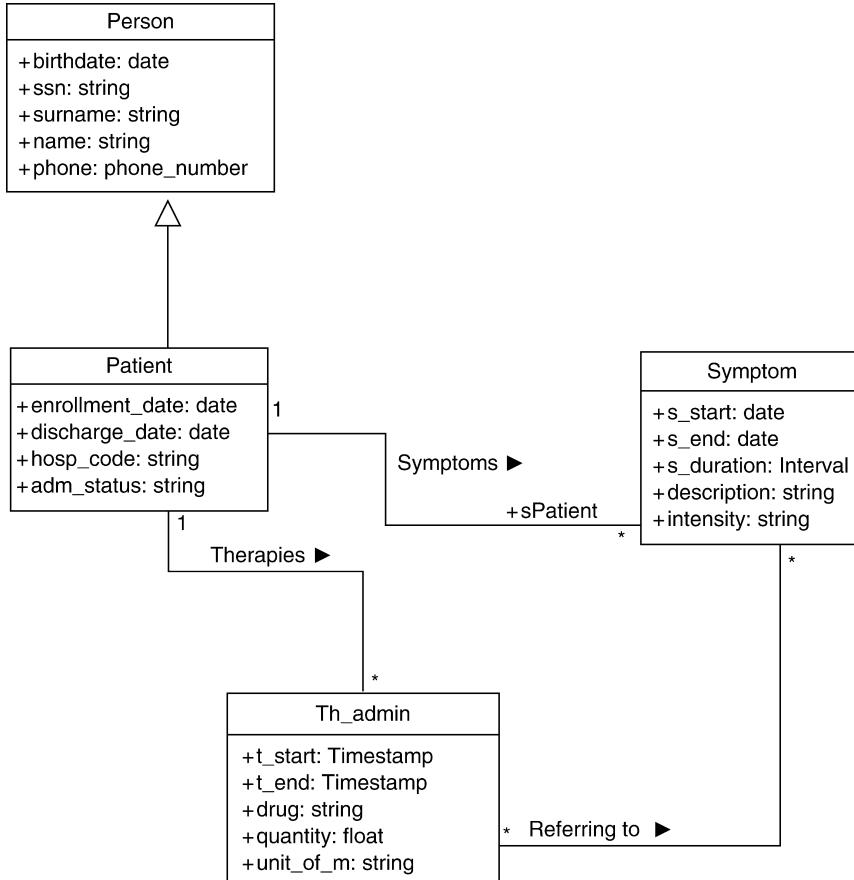
Time and Abstract Data Types

Through suitable abstract data types (ADTs), it is possible to represent several and different features of time primitives; more specifically ADTs can be suitably used to represent anchored (i.e., time points and time periods) and unanchored (i.e., durations) temporal primitives. Usually, any OO database system allows one to define ADTs, which can be possibly used to represent complex temporal concepts; moreover, any database system usually provides several built-in primitive and complex data types, allowing the user to represent the most usual time concepts.

Considering, for example, the clinical database depicted in Fig. 1, it is possible to observe different data types for time; according to the ODMG data model, the types Date and Timestamp represent time points corresponding to days and to seconds, respectively, while the type Interval stands for spans of time, i.e., distances between two time points.

According to this approach, as underlined by Wuu and Dayal [14], different types of time could be defined, starting from some general concept of time, through aggregation and generalization. For example, the ODMG type Timestamp could be viewed as a specialization of a supertype Point, having two operations defined, i.e., those for comparisons (= and >), as its behavior. Various time types can be defined as subtypes of Point. The two operations defined for Point are inherited, and could be suitably redefined for different subtypes. Indeed, properties of specific ordering relationships determine different time structures, e.g., total vs. partial, or dense vs. discrete. Moreover, through the basic structured types provided by the model, it is possible to build new types for time concepts. As an example, it could be important to specify a type allowing one to represent periods of time; this way, it is possible to represent in a more compact and powerful way the periods over which a therapy has been administered, a symptom was holding, a patient was hospitalized. Thus, in the database schema a new type Period will be used as part of types Patient, Th_admin, and Symptom, as depicted in Fig. 2.

ADTs may be used for managing even more complex time concepts [5], allowing one to deal with times given at different granularities or with indeterminacy in a uniform way.

Temporal Object-Oriented Databases. Figure 1. Object-oriented schema of the database about patients' symptoms and therapies, using ODMG standard time types.
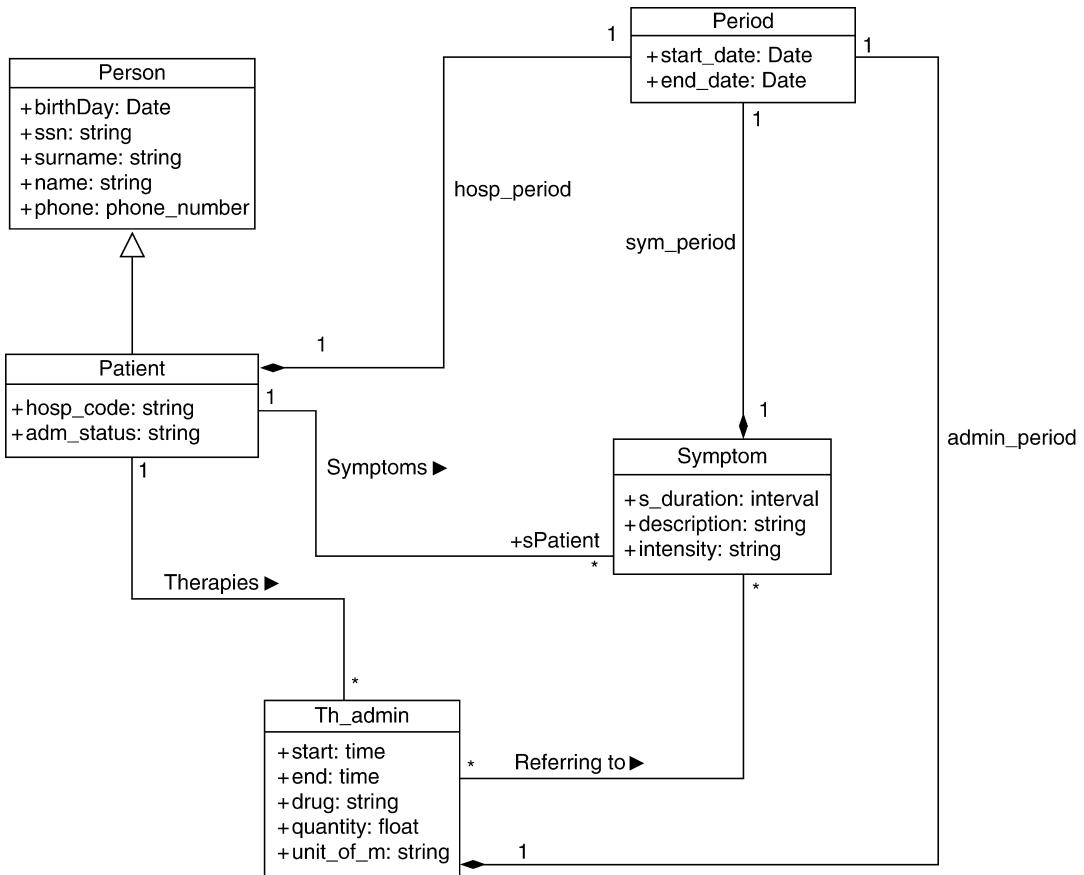
Temporal Object Data Models

Up to this point, the proposed solutions just use ADTs to represent complex time concepts; the sound association of time to objects for representing temporal data, i.e., time evolving information, according to the well-known temporal dimensions, such as valid and/or transaction times, is left to the user. Focusing, without loss of generality, on valid time, in the given example the semantics of valid time must be properly managed by the application designer, to check that symptoms occurred and therapies were administered when the patient was alive, that therapies related to symptoms were administered after that symptoms appeared, that there are no different objects describing the same symptom with two overlapping periods, and so on.

Managing this kind of temporal aspects in object databases has been the main focus of several research studies [13]: the main approaches adopted in dealing with the temporal dimension of data in object-oriented data models are (i) the direct use of an object-oriented data model (sometimes already extended to deal with other features as version management), and (ii) the modeling of the temporal dimensions of data through ad-hoc data models. The first approach is based on the idea that the rich (and extensible) type system usually provided by OO database systems allows one to represent temporal dimensions of data as required by different possible application domains. The second approach, instead, tries to provide the user with a data model where temporal dimensions are first-class citizens, avoiding the user the effort of modeling from scratch temporal features of data for each considered application domain.

General OO Models Using OO Concepts for Modeling Temporal Dimensions Among the proposed object-oriented systems able to deal with temporal
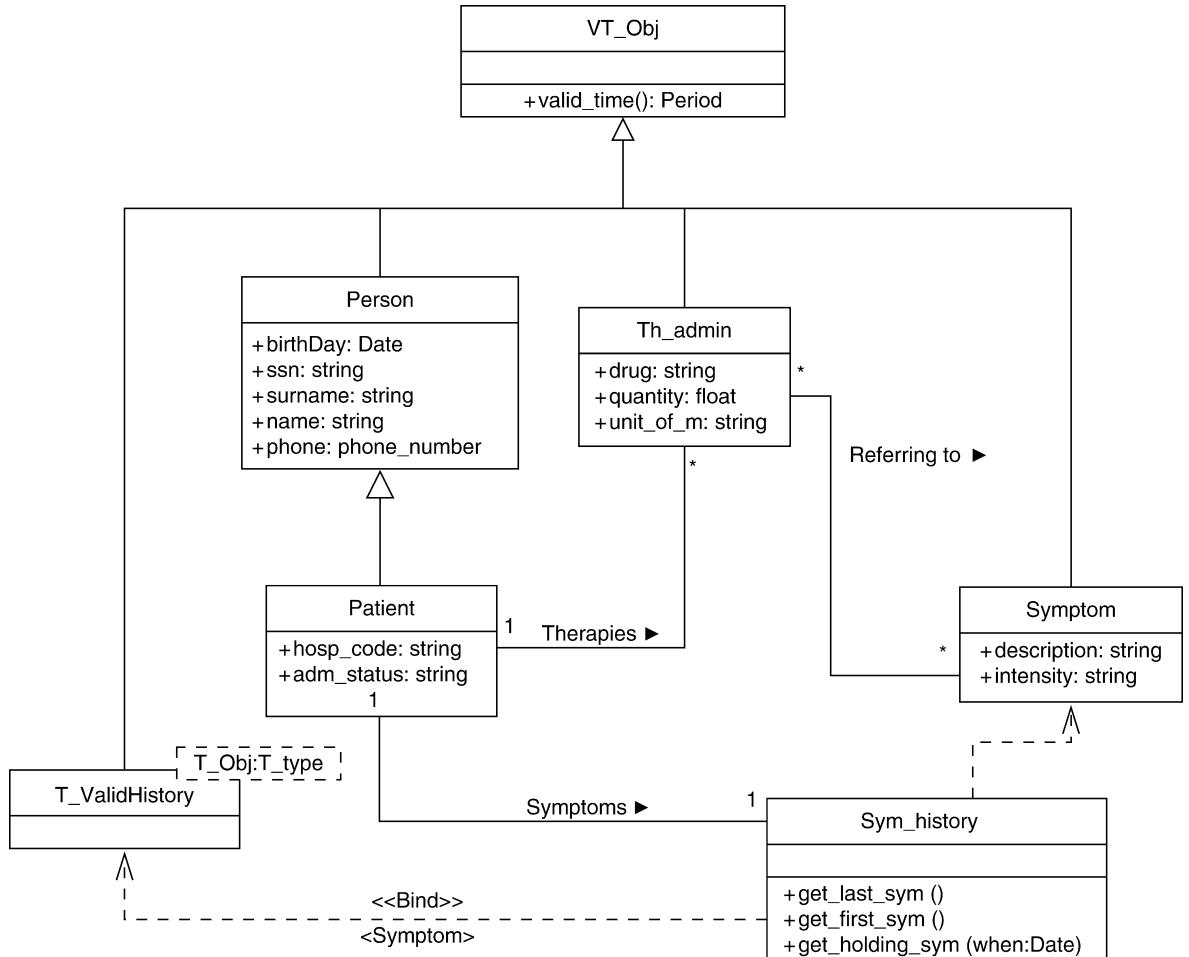
Temporal Object-Oriented Databases. Figure 2. Object-oriented schema of the database about patients' symptoms and therapies, introducing the type Period.

information, OODAPLEX and TIGUKAT adopt the direct use of an object-oriented data model [7,14]. In these systems suitable data types allow the database designer to model temporal information. For example, TIGUKAT models the valid time at the level of objects and of collections of objects. More particularly, for each single application it is possible to use the rich set of system-supported types, to define the real semantics of valid (or transaction) time.

According to this approach, the example database schema could be modified, to manage also the valid time semantics: as depicted in Fig. 3, objects having the valid time dimension are explicitly managed through the type VT_Obj, the supertype of all types representing temporal information, while the type Sym_History, based on the (template) type T_valid History is used to manage the set of objects representing symptoms.

OO Models Having Explicit Constructs for Temporal Dimensions of Data Besides the direct use of an OO data model, another approach which has been widely adopted in dealing with the temporal dimension of data by object-oriented data models consists of temporally-oriented extensions, which allow the user to explicitly model and consider temporal dimensions of data. As an example of this approach, Fig. 4 depicts the schema related to patients' symptoms and therapies: the temporal object-oriented data model underlying this schema allows one to manage the valid time of objects, often referred to as *lifespan* in the object-oriented terminology (this aspect is represented through the stereotype temporal and the operation lifespan() for all types); moreover, the temporal model allows the designer to specify that attributes within types can be time-varying according to different granularities: for example, the intensity of a symptom
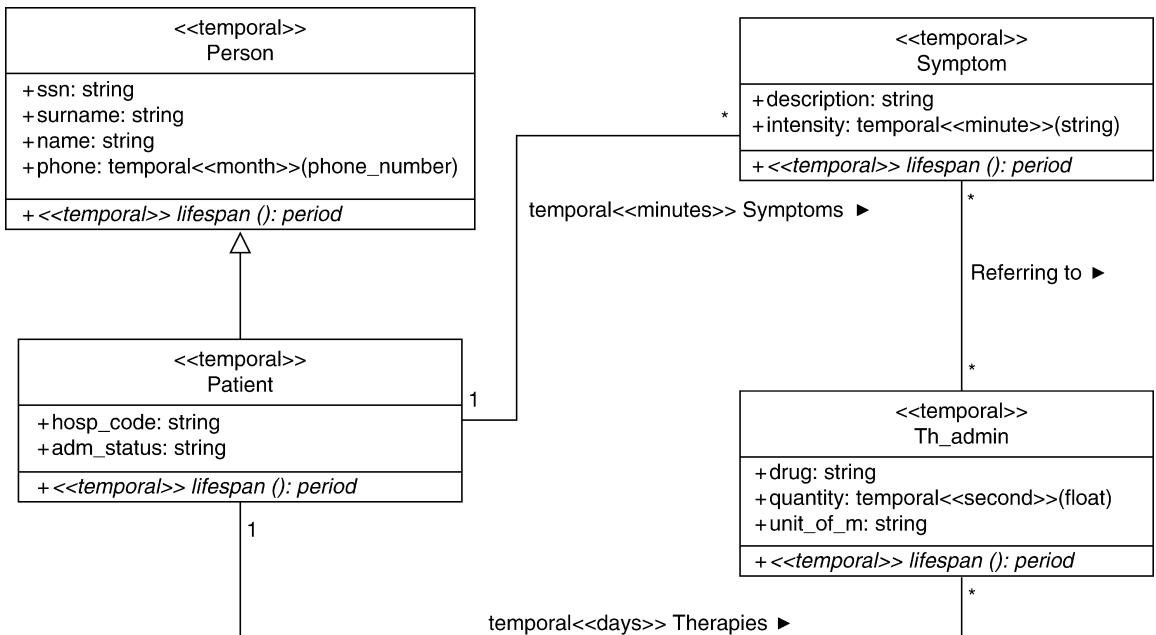
Temporal Object-Oriented Databases. Figure 3. Object-oriented schema of the database about patients' symptoms and therapies, modeling temporal dimensions through suitable types.

may change over time. Finally, temporalities can be represented even for associations, specifying the granularity to consider for them. Several constraints can be defined, and implicitly verified, when defining a temporal object-oriented data model. For example, in this case, any object attribute could be constrained to hold over some subinterval of the lifespan of the considered object: $\forall o \in$ Symptom $(o.\text{intensity}.VT() \subseteq o.$ lifespan()$)$, where $VT()$ returns the overall period over which the (varying) values of a temporal attribute have been specified. In a similar way, a temporal relationship is allowed only when both the related objects exist. As for the inheritance, objects could be allowed to move from some supertype to a subtype during their existence; the constraint here is that their lifespan as instances of a subtype must be a subinterval of their

lifespan as instances of a supertype: $\forall o \in$ Patient $(o.\text{lifespan}() \subseteq ((\text{Person})o).\text{lifespan}())$

Temporal Object Query Languages

According to the approaches for object-oriented data models dealing with temporal information, even when querying temporal data, it is possible to either adopt a generic object-oriented query language and use directly its constructs for temporal data or extend an (atemporal) query language with some ad-hoc keywords and clauses to deal with temporal data in a more powerful and expressive way [3,5,12]. For both the approaches, the main focus is on querying data: data definition and data manipulation are usually performed through the object-oriented language provided by the database system [5].

Temporal Object-Oriented Databases. Figure 4. The temporal object-oriented schema of the database about patients' symptoms and therapies.

Temporal Object-Oriented Database Systems
Usually, temporal object-oriented database systems, offering a temporal data model and a temporal query language, are realized on top of object-oriented database systems, through a software layer able to translate temporalities in data and queries into suitable data structures and statements of the underlying OO system [5]. According to this point of view, only recently some studies have considered technical aspects at the physical data level; among them it is worth mentioning here the indexing of time objects and storage architectures for transaction time object databases [10].

Key Applications

Clinical database systems are among the applications of temporal object-oriented databases, i.e., the real world databases where the structural complexity of data needs for the object-oriented technology. Indeed, clinical database systems have to manage temporal data, often with multimedia features, and complex relationships among data, due both to the healthcare organization and to the medical and clinical knowledge. Attention in this field has been paid on the management of clinical data given at different granularities and with indeterminacy [5,7]. Another interesting application domain

is that of video database systems, where temporal aspects of object technology have been studied for the management of temporal aspects of videos. Complex temporal queries have been studied in this context, involving spatio-temporal constrains between moving objects.

Future Directions

New and more intriguing topics have attracted the attention of the temporal database community in these last years; however, the results obtained for temporal object-oriented databases could be properly used in different contexts, such as that of temporal object-relational databases, which seem to attract also the attention of the main companies developing commercial database systems, and that of semi-structured temporal databases, where several OO concepts could be studied and extended to deal with temporalities for partially structured information (as that represented through XML data).

Cross-references
▶ Object-Oriented Data Model
▶ Object Query Language
▶ Object-Relational Data Model
▶ Spatio-temporal Data Models

Recommended Reading

1. Bertino E., Ferrari E., and Guerrini G. A formal temporal object-oriented data model. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 342–356.

2. Bertino E., Ferrari E., Guerrini G., and Merlo I. T-ODMG: an ODMG compliant temporal object model supporting multiple granularity management. Inf. Syst., 28(8):885–927, 2003.

3. Cattel R.G.G. and Barry D.K. (eds.). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.

4. Clifford J. and Croker A. Objects in time. IEEE Data Eng. Bull., 11(4):11–18, 1988.

5. Combi C., Cucchi C., and Pinciroli F. Applying object-oriented technologies in modeling and querying temporally-oriented clinical databases dealing with temporal granularity and indeterminacy. IEEE Trans. Inf. Tech. Biomed., 1:100–127, 1997.

6. Galante R.M., dos Santos C.S., Edelweiss N., and Moreira A.F. Temporal and versioning model for schema evolution in object-oriented databases. Data Knowl. Eng., 53(2):99–128, 2005.

7. Goralwalla I.A., Özsu M.T., and Szafron D. Modeling medical trials in pharmacoeconomics using a temporal object model. Comput. Biol. Med., 27:369–387, 1997.

8. Goralwalla I.A., Özsu M.T., and Szafron D. An object-oriented framework for temporal data models. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.), Springer, 1998, pp. 1–35.

9. Goralwalla I.A., Szafron D., Özsu M.T., and Peters R.J. A temporal approach to managing schema evolution in object database systems. Data Knowl. Eng., 28(1):73–105, 1998.

10. Nørvåg K. The vagabond approach to logging and recovery in transaction-time temporal object database systems. IEEE Trans. Knowl. Data Eng., 16(4):504–518, 2004.

11. Ozsoyoglu G. and Snodgrass R.T. Temporal and real-time databases: a survey. IEEE Trans. Knowl. Data Eng., 7(4):513–32, 1995.

12. Rose E. and Segev A. TOOSQL – A Temporal Object-Oriented Query Language. In Proc. 12th Int. Conf. on Entity-Relationship Approach, 1993, pp. 122–136.

13. Snodgrass R.T. Temporal Object-Oriented Databases: A Critical Comparison. In Modern Database Systems: The Object Model, Interoperability and Beyond. W. Kim (ed.). Addison-Wesley, 1995, pp. 386–408.

14. Wuu G.T.J. and Dayal U. A Uniform Model for Temporal and Versioned Object-oriented Databases. In Temporal Databases. A.U. Tansel, J. Clifford, S.K. Godia, A. Segev, R. Snodgrass (eds.), 1993, pp. 230–247.

15. Zimányi E. and Minout M. Implementing conceptual spatio-temporal schemas in object-relational dbmss. In OTM Workshops (2). LNCS, Vol. 4278, R. Meersman, Z. Tari, P. Herrero (eds.), 2006, pp. 1648–1657.

Temporal Periodicity

Paolo Terenziani
University of Turin, Turin, Italy

Definition

Informally, *periodic events* are events that repeat regularly in time (e.g., each Tuesday), and *temporal periodicity* is their temporal *periodic pattern* of repetition. A pattern is *periodic* if it can be represented by specifying a finite portion of it, and the duration of each repetition. For instance, supposing that day 1 is a Monday, the pair <*'day 2,' '7 days'*> may implicitly represent all Tuesdays.

A useful generalization of periodic patterns are *eventually* periodic ones, i.e., patterns that can be expressed by the union of a periodic pattern and a finite non-periodic one.

The above notion of periodic events can be further extended. For instance, Tuzhilin and Clifford [14] distinguish between *"strongly"* periodic events, that occur at equally distant moments in time (e.g., a class, scheduled to meet once a week, on Wednesday at 11A.M), *"nearly periodic"* events, occurring at regular periods, but not necessarily at equally distant moments of time (e.g., a meeting, that has to be held once a week, but not necessarily on the same day), and *"intermittent"* events, such that if one of them occurred then the next one will follow some time in the future, but it is not clear when (e.g., a person visiting "periodically" a pub). Most of the approaches discussed in the following cope only with "strongly" periodic events.

Finally, it is worth highlighting that "(periodic) temporal granularity," "calendar," and "calendric system" are notions closely related to temporal periodicity.

Historical Background

Temporal periodicity is pervasive of the world all around us. Many natural and artificial phenomena take place at periodic time, and temporal periodicity seems to be an intrinsic part of the way humans approach reality. Many real-world applications, including process control, data access control, data broadcasting, planning, scheduling, multimedia, active databases, banking, law and so on need to deal with periodic events. The problem of how to store and query

periodic data has been widely studied in the fields of databases, logic, and artificial intelligence.

In all such areas it is widely agreed that, since many different data conventions exist, a pre-defined set of periodicities would not suffice. For instance, Snodgrass and Soo [12] have emphasized that the use of a calendar depends on the cultural, legal, and even business orientation of the users, and listed many examples of different calendric systems. As a consequence, many approaches to *user-defined* temporal periodicity have been proposed. The core issue is the definition of expressive *formal languages* to *represent* and *query* user-defined temporal periodicity. In particular, an *implicit* representation [2] is needed, since it allows one to cope with data holding at periodic times in a compact way instead of explicitly listing all the instances (extensions) of the given periodicity (e.g., all "days" in a possibly infinite frame of time). Additionally, also the *set-theoretic operations* (intersection, union, difference) on definable periodicities can be provided (e.g., in order to make the formalism a suitable candidate to represent periodic data in temporal databases). Operationally, also *mapping functions* between periodicities are an important issue to be taken into account.

Within the database community, the problem of providing an implicit treatment of temporal periodicity has been intensively investigated since the late 1980s. Roughly speaking, such approaches can be divided into three mainstreams (the terminology is derived from Baudinette et al. [2] and Niezette and Stevenne [9]):

(1) *Deductive rule*-based approaches, using deductive rules. For instance, Chomicki and Imielinsky [4] dealt with periodicity via the introduction of the successor function in Datalog;
(2) *Constraint*-based approaches, using mathematical formulae and constraints (e.g., [6]);
(3) *Symbolic* approaches, providing symbolic formal languages to cope with temporal periodicity in a compositional (and hopefully *natural* and *commonsense*) way (consider, e.g., [9,8]).

Tuzhilin and Clifford [14] have proposed a comprehensive survey of many works in such mainstreams, considering also several approaches in the areas of logics and of artificial intelligence.

Foundations

In the following, the main features of the three mainstreams mentioned above are analyzed.

Deductive Rule-Based Approaches

Probably the first milestone among *deductive rule-based* approaches is the seminal work by Chomicki and Imielinski [4], who used $Datalog_{1S}$ to represent temporal periodicity. $Datalog_{1S}$ is the extension to Datalog with the successor function. In their approach, *one* temporal parameter is added to $Datalog_{1S}$ predicates, to represent time. For instance, in their approach, the schedule of backups in a distributed system can be represented by the following rules:

$$backup(T + 24, X) \leftarrow backup(T, X)$$

$$backup(T, Y) \leftarrow dependent(X, Y), backup(T, X)$$

The first rule states that a backup on a machine should be taken every 24 hours. The second rules requires that all backups should be taken simultaneously on all dependent machines (e.g., sharing files). Of course, $Datalog_{1S}$ programs may have infinite least Herbrand models, so that infinite query answers may be generated. However, in their later works Chomicki and Imielinski have provided a finite representation of them.

Another influential rule-based approach is based on the adoption of Templog, an extension of logic programming based on temporal logic. In this language, predicates can vary with time, but the time point they refer to is defined implicitly by temporal operators rather than by an explicit temporal argument [2]. Specifically, three temporal operators are used in Templog: *next*, which refers to the next time instant, *always*, which refers to the present and all the future time instants, and *eventually*, which refers to the present or to some future time instant.

Constraint-Based Approaches

While deductive rule-based approaches rely on deductive database theory, the approaches of the other mainstreams apply to relational (or object-oriented) databases.

Kabanza et al. [6] have defined a *constraint-based* formalism based on the concept of *linear repeating points* (henceforth lpr's). A lrp is a set of points $\{x(n)\}$ defined by an expression of the form $x(n) = c + kn$

where k and c are integer constants and n ranges over the integers.

A *generalized tuple* of temporal arity k is a tuple with k temporal attributes, each one represented by a lrp, possibly including *constraints* expressed by linear equalities or disequalities between temporal attributes. Semantically, a generalized tuple denotes a (possibly infinite) set of (ordinary) tuples, one tuple for each value of the temporal attributes satisfying the lrp's definitions and the constraints. For instance, the generalized tuple

$$(a_1,...,a_n|[5+4n_1, 7+4n_2]\Lambda\, X_1 = X_2 - 2)$$

(with data part $a_1,...,a_n$) represents the infinite set of tuples with temporal attributes X_1 and X_2, such that $X_1 = 5 + 4n_1$, $X_2 = 7 + 4n_2$, $X_1 = X_2-2$, for some integers n_1 and n_2, i.e.,

$$\{\dots (a_1,...,a_n|[1,3]), (a_1,...,a_n|[5,7]),$$
$$(a_1,...,a_n|[9,10]),...\}$$

A *generalized relation* is a finite set of generalized tuples of the same schema.

In Kabanza et al. [6], the algebraic operations (e.g., intersection) have been defined over generalized relations as mathematical manipulations of the formulae coding lrp's.

A comparative analysis of deductive rule-based and constraint-based approaches has been provided, e.g., by Baudinet et al. [2], showing that they have the same *data expressiveness* (i.e., the set of temporal databases that can be expressed in such languages is the same). Specifically, as concerns the temporal component, they express *eventually periodic* sets of points (which are, indeed, points that can be defined by *Presburger Arithmetics* – see below). In such an approach, the *query* expressiveness and the computational complexity of such formalisms have also been studied.

Symbolic Approaches

In constraint-based approaches, temporal periodicity is represented through mathematical formulae. Several authors have suggested that, although expressive, such approaches do not cope with temporal periodicity in a "commonsense" (in the sense of "human-oriented") way, arguing in favor of a *symbolic* representation, in which complex periodicities can be compositionally built in terms of simpler ones (see, e.g., the discussions in [9,13]). A milestone in the area of symbolic

approaches to temporal periodicity is the early approach by Leban et al. [8]. In Leban's approach, *collections* are the core notion. A *collection* is a *structured* set of *intervals* (elsewhere called *periods*). A base collection, partitioning the whole timeline (e.g., the collection of seconds), is used as the basis of the construction. A new partition P' of the timeline (called *calendar* in Leban's terminology) can be built from another partition P as follow:

$$P' = <P; s_0, \dots s_{n-1}>$$

where $s_0,...,s_{n-1}$ are natural numbers greater than 0. Intuitively, $s_0,...,s_{n-1}$ define the number of periods of P whose unions yield periods of P', intending that the union operation has to be repeated cyclically. For example, Weeks = {Days;7} defines weeks as aggregations of seven days. Analogously, months (not considering leap years for the sake of brevity) can be defined by the expression Months = {Days; 31,28,31,30,31,30,31,31,30,31,30,31}.
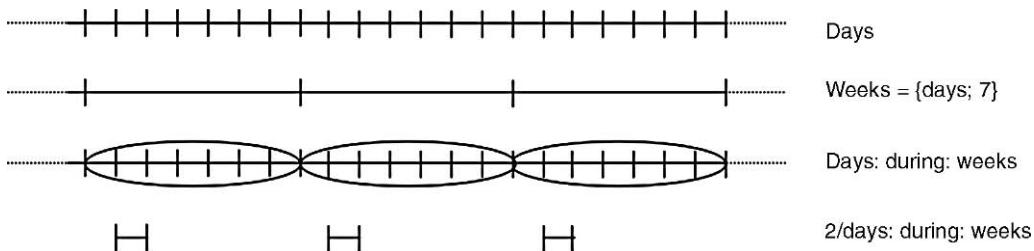
Two classes of operators, *dicing* and *slicing*, are defined in order to operate on collections.

The dicing operators provide means to further divide each period in a collection within another collection. For example, given the definitions of *Days* and *Weeks*, Day:during:Weeks breaks up weeks into the collection of days they contain. Other dicing operators are allowed (taken from a subset of Allen's relations [1]). Slicing operators provide a way of selecting periods from collections. For instance, in the expression 2/Day:during:Weeks the slicing operator "2/" is used in order to select the second day in each week. Different types of slicing operators are provided (e.g., -n/selects the *n-th* last interval from each collection).

Collection expressions can be arbitrarily built by using a combination of these operators (see, e.g., The examples in Figure 1).

While Leban's approach was mainly aimed to represent periodicity within knowledge bases (i.e., in artificial intelligence contexts), later on it has played an increasingly important role also in the area of temporal databases. Recently, Terenziani [8] has defined a temporal extension to relational databases, in which the valid time of periodic tuples can be modeled through (an extension of) Leban's language, and symbolic manipulation is used in order to perform algebraic operations on the temporal relations.

Another early symbolic approach which has been widely used within the database and artificial

Temporal Periodicity. Figure 1. Operators in Leban's language.

intelligence areas is the one by Niezette and Stevenne [9], who provided a formalism as well as the algebraic operations on it, mostly as an upper layer built upon *linear repeating points* [6]. A comparison between such a formalism and Leban's one, as well as an analysis of the relationships between the periodicities defined by such languages and *periodic granularities* has been provided by Bettini and De Sibi [3]. A recent influential symbolic approach, based on the notion of periodic granularities, has been proposed by Ning et al. [10].

An important issue concerns the formal analysis of the expressiveness (and of the semantics) of the implicit representation formalisms proposed to cope with temporal periodicity. *Presburger Arithmetics*, i.e., the first-order theory of addition and ordering over integers, is a natural reference to evaluate the expressiveness (and semantics) of such languages, because of its simplicity, decidability, and expressiveness, since it turns out that all sets definable in Presburger Arithmetics are *finite*, *periodic*, or *eventually periodic*. A recent comparison of the expressiveness of several constraint-based and symbolic approaches, based on Presburger Arithmetics, has been provided by Egidi and Terenziani [15].

While the above-mentioned approaches in [9,6,13] mainly focused on extending the relational model to cope with temporal periodicity, Kurt and Ozsoyoglu [7] devoted more attention to the definition of a *periodic temporal object oriented SQL*. They defined the temporal type of *periodic elements* to model strictly periodic and also eventually periodic events. Periodic elements consist of both an aperiodic part and a periodic part, represented by the repetition pattern and the period of repetition. They also defined the set-theoretic operations of union, intersection, complement and difference, which are closed with respect to periodic elements.

Moreover, in the last years, periodicity has also started to be studied in the context of moving objects.

In particular, Revesz and Cai [11] have taken into account also periodic (called *cyclic* periodic) and eventually periodic (called *acyclic* periodic) *movements* of objects. They have proposed an extended relational data model in which objects are approximated by unions of *parametric rectangles.* Movement is coped with by modeling the x and y dimensions of rectangles through functions of the form f(t mod p), where t denotes time, p the repetition period, and mod the module function. Revesz and Cai have also defined the algebraic operations on the data model.

Key Applications

Temporal periodicity plays an important role in many application areas, including process control, data access control, office automation, data broadcasting, planning, scheduling, multimedia, active databases, banking, and so on. Languages to specify user-defined periodicities in the *queries* have been already supported by many approaches, including commercial ones. For instance, Oracle provides a language to specify periodicity in the queries to *time series* (e.g., to ask for the values of IBM at the stock exchange *each Monday*). Even more interestingly, such languages can be used in the *data*, in order to express (finite and infinite) *periodic valid times* in an implicit and compact way. However, such a move involves an in-depth revision and extension of "standard" database theory and technology, which have been partly addressed within the temporal database research community, but have not yet been fully implemented in commercial products.

Cross-references
▶ Allen's Relations
▶ Calendar
▶ Calendric System
▶ Temporal Granularity
▶ Time Series Query
▶ Valid Time

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Baudinet M., Chomicki J., and Wolper P. Temporal deductive databases. In Temporal Databases, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.). Benjamin/Cummings, 1993, pp. 294–320.
3. Bettini C. and De Sibi R. Symbolic representation of user-defined time granularities. Ann. Math. Artif. Intell., 30(1–4):53–92, 2000.
4. Chomicki J. and Imielinsky T. Temporal deductive databases and infinite objects. In Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1988, pp. 61–73.
5. Egidi L. and Terenziani P. A mathematical framework for the semantics of symbolic languages representing periodic time. Ann. Math. Artif. Intell., 46:317–347, 2006.
6. Kabanza F., Stevenne J.-M., and Wolper P. Handling infinite temporal data. In Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1990, pp. 392–403.
7. Kurt A. and Ozsoyoglu M. Modelling and querying periodic temporal databases. In Proc. 6th Int. Conf. and Workshop on Database and Expert Systems Applications, 1995, pp. 124–133.
8. Leban B., McDonald D.D., and Forster D.R. A representation for collections of temporal intervals. In Proc. 5th National Conf. on AI, 1986, pp. 367–371.
9. Niezette M. and Stevenne J.-M. An efficient symbolic representation of periodic time. In Proc. Int. Conf. on Information and Knowledge Management, 1992.
10. Ning P., Wang X.S., and Jajodia S. An algebraic representation of calendars. Ann. Math. Artif. Intell., 36(1–2):5–38, 2002.
11. Revesz P. and Cai M. Efficient querying and animation of periodic spatio-temporal databases. Ann. Math. Artif. Intell., 36(4):437–457, 2002.
12. Snodgrass R.T. and Soo M.D. *1*Supporting multiple calendars. In The TSQL2 Temporal Query Language, R.T. (ed.). R.T. Snodgrass (ed.). Kluwer, Norwell, MA, 1995, pp. 103–121.
13. Terenziani P. Symbolic user-defined periodicity in temporal relational databases. IEEE Trans. Knowl. Data Eng., 15(2):489–509, 2003.
14. Tuzhilin A. and Clifford J. On periodicity in temporal databases. Inf. Syst., 20(8):619–639, 1995.

Temporal Projection

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Temporal assignment

Definition

In a query or update statement, *temporal projection* pairs the computed facts with their associated times, usually derived from the associated times of the underlying facts.

The generic notion of temporal projection may be applied to various specific time dimensions. For example, *valid-time projection* associates with derived facts the times at which they are valid, usually based on the valid times of the underlying facts.

Key Points

While almost all temporal query languages support temporal projection, the flexibility of that support varies greatly.

In some languages, temporal projection is implicit and is based the intersection of the times of the underlying facts. Other languages have special constructs to specify temporal projection.

The term "temporal projection" has been used extensively in the literature. It derives from the `retrieve` clause in Quel as well as the `SELECT` clause in SQL, which both serve the purpose of the relational algebra operator (generalized) projection, in addition to allowing the specification of derived attribute values.

The related concept called "temporal assignment" roughly speaking is a function that maps a set of time values to a set of values of an attribute. One purpose of a temporal assignment would be to indicate when different values of the attribute are valid.

Cross-references

► Projection
► SQL
► SQL-Based Temporal Query Languages
► Temporal Database
► Temporal Query Languages
► TSQL2
► Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Query Languages

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]University of Aalborg, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Historical query languages

Definition

A temporal query language is a database query language that offers some form of built-in support for the querying and modification of time-referenced data, as well as enabling the specification of assertions and constraints on such data. A temporal query language is usually quite closely associated with a temporal data model that defines the underlying data structures to which the language applies.

Historical Background

When the relational data model was proposed by Codd, he also proposed two query languages: the relational calculus and the relational algebra. Similarly, temporal data models are closely coupled with temporal query languages.

Most databases store time-referenced, or temporal, data. The ISO standard Structured Query Language SQL [4] is often the language of choice when developing applications that utilize the information captured in such databases. In spite of this, users realize that temporal data management is often challenging with SQL. To understand some of the difficulties, it is instructive to attempt to formulate the following straightforward, realistic queries, assertions, and modifications in SQL. An intermediate SQL programmer can express all of them in SQL for a database without time-referenced data in perhaps 5 minutes. However, even SQL experts find these same queries challenging to do in several *hours* when the data are time-referenced [6].

- An `Employee` table has three attributes: `Name`, `Manager`, and `Dept`. Temporal information is retained by adding a fourth attribute, `When`, of data type `PERIOD`. Attribute `Manager` is a foreign key for `Employee.Name`. This means that at each point in time, the `Manager` value of a tuple also occurs as a `Name` value (probably in a different tuple). This cannot be expressed via SQL's foreign-key constraint, which fails to take time into account. Formulating this constraint as an assertion is challenging.

- Consider the query "List those employees who are not managers." This can easily be expressed in SQL, using `EXCEPT` or `NOT EXISTS`, on the original, non-temporal relation with three attributes. Things are just a little harder with the `When` attribute; a `WHERE` predicate is required to extract the current employees. To formulate the query "List those employees who were not managers, and indicate when," `EXCEPT` and `NOT EXISTS` do not work because they do not consider time. This simple temporal query is hard to formulate, even for SQL experts.

- Consider the query "Give the number of employees in each department." Again, this is a simple SQL query using the `COUNT` aggregate when formulated on non-temporal data. To formulate the query on temporal data, i.e., "Give *the history of* the number of employees in each department," is very difficult without built-in temporal support in the language.

- The modification "Change the manager of the Tools department for 2004 to Bob" is difficult in SQL because only a portion of many validity periods are to be changed, with the information outside of 2004 being retained.

Most users know only too well that while SQL is an extremely powerful language for writing queries on the current state, the language provides much less help when writing temporal queries, modifications, and assertions and constraints.

Hence there is a need for query languages that explicitly "understand" time and offer built-in support for the management of temporal data. Fortunately, the outcomes of several decades of research in temporal query languages demonstrate that it is indeed possible to build support for temporal data management into query languages so that statements such as the above are easily formulated.

Foundations

Structure may be applied to the plethora of temporal query languages by categorizing these languages according to different concerns.

Language Extension Approaches

One attempt at bringing structure to the diverse collection of temporal query languages associates these languages with four approaches that emphasize how temporal support is being added to a non-temporal query language [1].

Abstract Data Types for Time From a design and implementation perspective, the simplest approach to improving the temporal data management capabilities of an existing query language is to introduce time data types with associated predicates and functions. This approach is common for query languages associated with temporal object-oriented databases [7].

Data types for time points, time intervals (periods), and for durations of time may be envisioned, as may data types for temporal elements, i.e., finite unions of time intervals. The predicates associated with time-interval data types are often inspired by Allen's 13 interval relationships. With reference to these, different sets of practical proposals for predicates have been proposed.

However, while being relatively easy to achieve, the introduction of appropriate time data types results only in modest improvements of the temporal data management capabilities of the query language.

Use of Point Timestamps An interval timestamp associated with a tuple in a temporal relational data model is often intended to capture the fact that the information recorded by the tuple is valid at each time point contained in the interval. This way, the interval is simply a compact representation of a set of time points. Thus, the same information can be captured by a single tuple timestamped with an interval and a set of identical tuples, each timestamped with a different time point from the interval (with no time points missing).

One attraction of intervals is that their representing is of fixed size. Another is that they appear to be very intuitive to most users — the notion of an interval is conceptually very natural, and people use it frequently in their daily thinking and interactions. In some respects, the most straightforward and simplest means of capturing temporal aspects is to use interval-valued timestamps.

However, the observation has also been advanced that the difficulty in formulating temporal queries on relations with interval-timestamped tuples stem exactly from the intervals – Allen has shown that there are 13 possible relations between a pair of intervals. It has been argued that a language such as SQL is unprepared to support something (an interval) that represents something (a convex set of time points) that it is not.

Based on this view, it is reasonable to equip an SQL extended with interval-valued timestamps with the ability to *unfold* and *fold* timestamps. The unfold function maps an interval-stamped tuple (or set of tuples) to the corresponding set of point-stamped tuples (set of tuples), and the fold function collapses a set of point-stamped tuples into the corresponding interval-stamped tuple(s). This way, it is possible to manipulate both point- and interval-stamped relations in the same language. If deemed advantageous when formulating a statement, one can effectively avoid the intervals by first unfolding all argument relations. Then the statement is formulated on the point-stamped relations. At the end, the result can be folded back into an interval-stamped format that lends itself to presentation to humans.

A more radical approach to designing a temporal query language is to completely abandon interval timestamps and use only point timestamps. This yields a very clean and simple design, although it appears that database modification and the presentation of query results to humans must still rely on intervals and thus are "outside" the approach.

The strength of this approach lies in its generalization of queries on non-temporal relations to corresponding queries on corresponding temporal relations. The idea is to extend the non-temporal query with equality constraints on the timestamp attribute of the temporal relation, to separate different temporal database states during query evaluation, thereby naturally supporting sequenced semantics.

Syntactic Defaults This approach introduces what may be termed syntactic defaults along with the introduction of temporal abstract data types, the purpose being to make the formulation of common temporal queries more convenient. Common defaults concern the access to the current state of a temporal relation and the handling of temporal generalizations of common non-temporal queries, e.g., joins. The temporal generalization of a non-temporal join is one where two tuples join if their timestamps intersect and where the timestamp of the result tuple is the intersection of the timestamps. Essentially, the non-temporal query is computed for each point in time, and the results of these queries are consolidated into a single result. The nature of the consolidation depends on the data type of the timestamps; if intervals are used, the consolidation involves coalescing.

The most comprehensive approach based on syntactic defaults is the TSQL2 language [5], but many of the earlier query languages that the creators of this language were attempting to consolidate also follow this approach. As an example, TSQL2 includes a default valid clause that computes the intersection of the valid times of the tuples in the argument relations mentioned in a statement's from clause, which is then returned in the result. So as explained above, the timestamp of a tuple that results from joining two relations is the intersection of the timestamps of the two argument tuples that produce the tuple. When there is only one argument relation, the valid clause produces the original timestamps.

Such defaults are very effective in enabling the concise formulation of common queries, but they also tend to complicate the semantics of the resulting temporal query language.

Semantic Defaults The use of semantic defaults is motivated in part by the difficulties in systematically extending a large and unorthogonal language such as SQL with syntactic defaults that are easy to understand and that do not interact in unintended ways. With this approach, so-called *statement modifiers* are added to a non-temporal query language, e.g., SQL, in order to obtain built-in temporal support [8].

It was argued earlier that statements that are easily formulated in SQL on non-temporal relations can be very difficult to formulate on temporal relations. The basic idea is then to make it easy to systematically formulate temporal queries from non-temporal queries. With statement modifiers, a temporal query is then formulated by first formulating the "corresponding" non-temporal query (i.e., assuming that there are no timestamp attributes on the argument relations) and then applying a statement modifier to this query.

For example, to formulate a temporal join, the first step is to formulate the corresponding non-temporal join. Next, a modifier is placed in front of this query to indicate that the non-temporal query is to be rendered temporal by computing it at each time point. The modifier ensures that the argument timestamps overlap and that the resulting timestamp is the intersection of the argument intervals. The attraction of using statement modifiers is that these may be placed in front of any non-temporal query to render that query temporal.

Statement modifiers are capable of specifying the semantics of the temporal queries unambiguously, independently of the syntactic complexity of the queries that the modifiers are applied to. This renders semantic defaults scalable across the constructs of the language being extended. With modifiers, the users thus need not worry about which predicates are needed on timestamps and how to express timestamps to be associated with result tuples. Further, the use of statement modifiers makes it possible to give more meaning to interval timestamps; they need no longer be simply compact representations of convex sets of time points.

Additional Characterizations of Temporal Query Languages

Several additional dimensions exist on which temporal query languages can be characterized. One is *abstractness*: is the query language at an abstract level or at a concrete level. Examples of the former are Temporal Relational Calculus and First-order Temporal Logic; an example of the latter is TSQL2.

Another dimension is *level*: is the query language at a logical or a physical level? A physical query language assumes a specific representation whereas a logical query language admits several representations. Examples of the former are examined in McKenzie's survey of relational algebras [3]; an example of the latter is the collection of algebraic operators defined on the Bitemporal Conceptual Data Model [2], which can be mapped to at least five representational models.

A third dimension is whether the query language supports a *period-stamped temporal model* or a *point-stamped temporal model*.

Other entries (indicated in *italics*) examine the long and deep research into temporal query languages in a more detailed fashion. *Qualitative temporal reasoning* and *temporal logic in database query languages* provide expressive query facilities. *Temporal vacuuming* provides a way to control the growth of a database. *TSQL2* and its successor SQL/Temporal provided a way for many in the temporal database community to coordinate their efforts in temporal query language design and implementation. *Temporal query processing* involves disparate architectures, from *temporal strata* outside the conventional DBMS to adding native temporal support within the DBMS. *Supporting transaction time* generally requires changes within the kernel of a DBMS. *Temporal algebras* extend the conventional relational algebra. Some specific operators (e.g., *temporal aggregation*, *temporal coalescing*, *temporal joins*) have received special attention. Finally, the

Oracle database management system includes support for valid and transaction time, both individually and in concert, in its extension of.

Future Directions

Given the substantial decrease in code size (a factor of three [6]) and dramatic decrease in conceptual complexity of temporal applications that temporal query languages offer, it is hoped that DBMS vendors will continue to incorporate temporal language constructs into their products.

Cross-references

▶ Abstract Versus Concrete Temporal Query Languages
▶ Allen's Relations
▶ Now in Temporal Databases
▶ Period-Stamped Temporal Models
▶ Point-Stamped Temporal Models
▶ Qualitative Temporal Reasoning
▶ Schema Versioning
▶ Sequenced Semantics
▶ Supporting Transaction Time Databases
▶ Temporal Aggregation
▶ Temporal Algebras
▶ Temporal Coalescing
▶ Temporal Database
▶ Temporal Data Models
▶ Temporal Joins
▶ Temporal Logic in Database Query Languages
▶ Temporal Object-Oriented Databases
▶ Temporal Query Processing
▶ Temporal Strata
▶ Temporal Upward Compatibility
▶ Temporal Vacuuming
▶ Temporal Visual Languages
▶ Temporal XML
▶ TSQL2

Recommended Reading

1. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data?. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2006, pp. 121–136.
2. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December 1994.
3. McKenzie E. and Snodgrass R.T. An evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, December 1991.
4. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, 1993.
5. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer, Boston, MA, USA, 1995.
6. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, USA, 1999.
7. Snodgrass R.T. Temporal Object Oriented Databases: A Critical, Comparison, Chapter 19 in Modern Database System: The Object Model, Interoperability and Beyond, W. Kim, editor, Addison-Wesley/ACM Press, 1995, pp. 386–408.
8. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal Statement Modifiers, ACM Transactions on Database Sytems, 25(4):407–456.

Temporal Query Processing

MICHAEL BÖHLEN
Free University of Bolzano-Bozen, Bolzano, Italy

Definition

Temporal query processing refers to the techniques used by database management system to process temporal statements. This ranges from the implementation of query execution plans to the design of system architectures. This entry surveys different system architectures. It is possible to identify three general system architectures that have been used to systematically offer temporal query processing functionality to applications [6]: The *layered* approach uses an off-the-shelf database system and extends it by implementing the missing functionality in a layer between the database system and the applications. The *monolithic* approach integrates the necessary application-specific extensions directly into the database system. The *extensible* approach relies on a database system that allows to plug user-defined extensions into the database system.

Historical Background

In order to deploy systems that offer support for temporal query processing new systems must be designed and implemented. Temporal extensions of database systems are quite different from other database system extensions. On the one hand, the complexity of the newly added types, often an interval, is usually quite low. Therefore, existing access structures and query processing techniques are often deemed sufficient to manage temporal information [7,8,11]. On the other hand the temporal aspect is ubiquitous and affects all parts of a database system. This is quite different

if extensions for multimedia, geographical or spatio-temporal data are considered. Such extensions deal with complex objects, for example, objects with complex boundaries, moving points, or movies, but their impact on the various components of a database system is limited.

A first overview of temporal database system implementations appeared in 1996 [2]. While the system architecture was not the focus, the overview describes the approached followed by the systems. Most systems follow the layered approach, including ChronoLog, ARCADIA, TimeDB, VT-SQL, and Tiger. A comprehensive study of a layered query processing architecture was done by Slivinskas et al. [10]. The authors use the Volcano extensible query optimizer to optimize and process queries. A monolithic approach is pursued by HDBMS, TDBMS, T-REQUIEM, and T-squared DBMS. A successful implementation of an index structure for temporal data has been done with the help of Informix's datablade technology [1].

Foundations

Figure 1 provides an overview of the different architectures that have been used to systematically provide temporal database functionality: the layered, monolithic, and extensible architectures. The gray parts denote the temporal extensions. The architecture balances initial investments, functionality, performance and lock-ins.

Functionalities

The different operators in a temporal database system have different characteristics with respect to query processing.

Temporal selection, temporal projection and temporal union resemble their non-temporal counterparts and they do not require dedicated new database functionality. Note though that this only holds if the operators may return uncoalesced relation instances [3] and if no special values, such as *now*, are required. If the operators must returned coalesced relations or if *now* must be supported the support offered by conventional database systems is lacking.

Temporal Cartesian product (and temporal joins) are also well supported by standard database systems. In many cases existing index structures can be used to index interval timestamps [7].
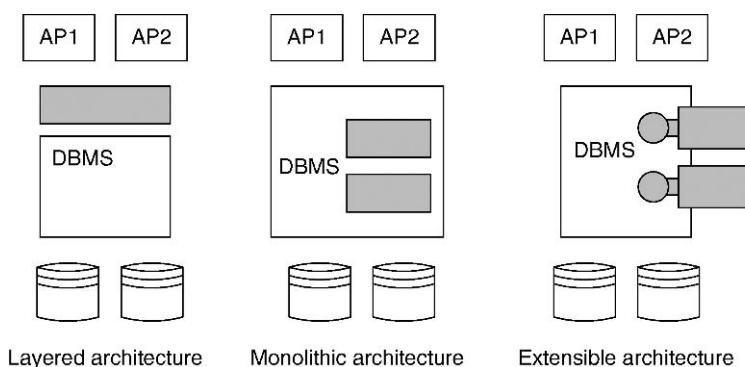
Temporal difference is more subtle and not well-supported by traditional database systems. It is possible to formulate temporal difference in, e.g., SQL, but it is cumbersome to do so. Algebraic formulations and efficient implementations have been studied by Dunn et al. [4].

Temporal aggregation is even worse than temporal difference. Formulating a temporal aggregation in SQL is a challenge for even advanced SQL programmers and yields a statement that current database systems cannot evaluate efficiently [11].

Temporal coalescing and temporal integrity constraints [11] are other examples of operations that current database system do not handle efficiently. *Temporal coalescing* is an algebraic operator and a declarative SQL implementation is inefficient.

The Layered Architecture

A common approach to design an extended database systems with new data types and operations for time-referenced data are to use an off-the-shelf database system and implement a layer on top providing data types

Temporal Query Processing. Figure 1. Illustration of architectural choices for temporal database systems

and services for temporal applications. The database system with such a component is then used by different applications having similar data type and operation requirements. Database systems enhanced in this way exploit the standard data types and data model, often the relational model, as a basis. They define new data types and possibly a new layer that provides application specific support for data definition and query language, query processing and optimization, indexing, and transaction management. Applications are written against the extended interface.

The layered approach has the advantage of using standard components. There is a clear separation of responsibilities: application-specific development can be performed and supported independent of the database system development. Improvements in the database system component are directly available in the whole system with almost no additional effort. On the other hand, the flexibility is limited. Development not foreseen in the database system component has to be implemented bypassing the database system. The more effort is put into such an application-specific data management extension, the more difficult it gets to change the system and take advantage of database system improvements. Also (legacy) applications might access the database system over different interfaces. For such applications accessing the extended database system through a special purpose layer is not always an option.

The reuse of a standard database system is an advantage but at the same time also a constraint. The layer translates and delegates temporal requests to sequences of nontemporal request. If some of the functionality of the database systems should be extended or changed, e.g., a refined transaction processing for transaction time, this cannot be done easily with a layered approach. For the advanced temporal functionality the layered architecture might not offer satisfactory performance.

The Monolithic Architecture

Many systems that use a monolithic architecture have originally been designed as stand-alone applications without database functionality. The designers of a monolithic architecture then extend their system with database system functionality. They add query functionality, transaction management, and multi-user capabilities, thereby gradually creating a specialized database system. The data management aspects traditionally associated with database system and the application-specific functionality are integrated into one component.

Instead of adding general database system functionality to an application it is also possible to incorporate the desired application domain semantics into the database system. Typically, this is done by database companies who have complete control over and knowledge of their source code or by open source communities.

Because of the tight integration of the general data management aspects and the application specific functionality, monolithic systems can be optimized for the specific application domain. This results in good performance. Standard and specialized index structures can be combined for good results. Transaction management can be provided in a uniform way for standard as well as new data types. However, implementing a monolithic system is difficult and a big (initial) effort, since all aspects of a database system have to be taken into account. Another drawback is that enterprises tend to be reluctant to replace their database system, which increases the threshold for the adoption of the temporal functionality. With monolithic systems, there is is a high risk of vendor lock-ins.

The Extensible Architecture

Extensible database systems can be extended with application-specific modules. Traditional database functionality like indexing, query optimization, and transaction management is supported for new data types and functions in a seamless fashion.

The first extensible system prototypes have been developed to support non-standard database system applications like geographical, multimedia or engineering information systems. Research on extensible systems has been carried out in several projects, e.g., Ingres [12], Postgres[13], and Volcano [5]. These projects addressed, among other, data model extensions, storage and indexing of complex objects as well as transaction management and query optimization in the presence of complex objects. Today a number of commercial approaches are available, e.g., datablades from Informix, cartridges from Oracle, and extenders from DB2. A limitation is that the extensions only permit extension that were foreseen initially. A comprehensive temporal support might require support that goes beyond data types and access structures.

The SQL99 standard [9] specifies new data types and type constructors in order to better support advanced applications.

The extensible architecture balances the advantages and disadvantages of the layered and monolithic architectures, respectively. There is a better potential to implement advanced temporal functionality with a satisfactory performance than in the layered architecture. However, functionality not foreseen by the extensible database system might still be difficult to implement.

Key Applications

All applications that want to provide systematic support for time-varying information must choose one of the basic system architectures. The chosen architecture balances (initial) investments, future lock-ins, and performance.

Future Directions

The architecture determines the initially required effort to provide support for time-varying information and may limit functionality and performance. Changing from one architecture to another is not supported. Such transitions would be important to support the graceful evolution of temporal database applications.

Cross-references
▶ Temporal Data Model
▶ Temporal Database
▶ Temporal Strata

Recommended Reading

1. Bliujute R., Saltenis S., Slivinskas G., and Jensen C.S. Developing a datablade for a new index. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 314–323.
2. Böhlen M.H. Temporal database system implementations. ACM SIGMOD Rec., 24(4):16, December 1995.
3. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 180–191
4. Dunn J., Davey S., Descour A., and Snodgrass R.T. Sequenced subset operators: definition and implementation. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 81–92.
5. Graefe G. and McKenna W.J. The volcano optimizer generator: extensibility and efficient search. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209-218.
6. Koubarakis M., Sellis T.K., Frank A.U., Grumbach S., Güting R.H., Jensen C.S., Lorentzos N.A., Manolopoulos Y., Nardelli E., Pernici B., Schek H., Scholl M., Theodoulidis B., and Tryfona N. (eds.). Spatio-Temporal Databases: The CHOROCHRONOS Approach. Springer, Berlin, 2003.
7. Kriegel H.-P., Pötke M., and Seidl T. Managing intervals efficiently in object-relational databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 407-418.
8. Leung T.Y.C. and Muntz R.R. Stream processing: temporal query processing and optimization. In Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., Snodgrass R.T. (eds.). Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993, pp. 329-355.
9. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, Los Altos, CA, 1993.
10. Slivinskas G., Jensen C.S., and Snodgrass R.T. Adaptable query optimization and evaluation in temporal middleware. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 127–138.
11. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, Los Altos, CA, 1999.
12. Stonebraker M. (ed.). The INGRES Papers: Anatomy of a Relational Database System. Addison-Wesley, Reading, MA, 1986.
13. The Postgresql Global. POSTGRESQL developer's guide.

Temporal Relation

▶ Bitemporal Relation

Temporal Relational Calculus

JAN CHOMICKI[1], DAVID TOMAN[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]University of Waterloo, Waterloo, ON, Canada

Synonyms
Two-sorted first-order logic

Definition
Temporal Relational Calculus (TRC) is a temporal query language extending the relational calculus. In addition to data variables and quantifiers ranging over a data domain (a universe of *uninterpreted constants*), temporal relational calculus allows *temporal variables* and quantifiers ranging over an appropriate time domain [1].

T

Key Points

A natural temporal extension of the relational calculus allows explicit variables and quantification over a given time domain, in addition to the variables and quantifiers over a data domain of uninterpreted constants. The language is simply the two-sorted version (variables and constants are temporal or non-temporal) of first-order logic over a data domain D and a time domain T.

The syntax of the two-sorted first-order language over a database schema $\rho = \{R_1,...,R_k\}$ is defined by the grammar rule:

$$Q ::= R(t_i, x_{i_1},..., x_{i_k}) \mid t_i < t_j \mid x_i = x_j \mid$$
$$Q \wedge Q \mid \neg Q \mid \exists x_i.Q \mid \exists t_i.Q$$

In the grammar, t_i's are used to denote temporal variables and x_i's to denote data (non-temporal) variables. The atomic formulae $t_i < t_j$ provide means to refer to the underlying ordering of the time domain. Note that the schema ρ contains schemas of *time-stamped temporal relations*.

Given a point-timestamped database DB and a two-sorted valuation θ, the semantics of a TRC query Q is defined in the standard way (similarly to the semantics of relational calculus) using the *satisfaction relation* $DB, \theta \vDash Q$:

$DB, \theta \vDash R_j(t_i, x_{i_1},..., x_{i_k})$ if $R_j \in \rho$ and $(\theta(t_i),$
                                     $\theta(x_{i_1}),..., \theta(x_{i_k})) \in R_j^{DB}$
$DB, \theta \vDash t_i < t_j$ if $\theta(t_i) < \theta(t_j)$
$DB, \theta \vDash x_i = x_j$ if $\theta(x_i) = \theta(x_j)$
$DB, \theta \vDash Q_1 \wedge Q_2$ if $DB, \theta \vDash Q_1$ and
                                  $DB, \theta \vDash Q_2$
$DB, \theta \vDash \neg Q_1$ if not $DB, \theta \vDash Q_1$
$DB, \theta \vDash \exists t_i.Q_1$ if there is $s \in T$ such
                                  that $DB, \theta[t_i \mapsto s] \vDash Q_1$
$DB, \theta \vDash \exists x_i.Q_1$ if there is $a \in D$ such
                                  that $DB, \theta[x_i \mapsto a] \vDash Q_1$

where R_j^{DB} is the interpretation of the predicate symbol R_j in the database DB.

The *answer to a query Q over DB* is the set $Q(DB)$ of valuations that make Q true in DB. Namely, $Q(DB) := \{\theta_{|FV\ (Q)} : DB, \theta \vDash Q\}$ where $\theta_{|FV\ (Q)}$ is the restriction of the valuation θ to the free variables of Q.

In many cases, the definition of TRC imposes additional restrictions on valid TRC queries:

Restrictions on free variables: Often the number of free temporal variables in TRC queries can be restricted to guarantee closure over the underlying data model (e.g., a single-dimensional timestamp data model or

the bitemporal model). Note that this restriction applies only to queries, not to subformulas of queries.

Range restrictions: Another common restriction is to require queries to be range restricted to guarantee domain independence. In the case of TRC (and many other abstract query languages), these restrictions depend crucially on the chosen *concrete encoding* of temporal databases. For example, no range restrictions are needed for temporal variables when queries are evaluated over interval-based database encodings, because the complement of an interval can be finitely represented by intervals.

The schemas of atomic relations, $R_j(t_i, x_{i_1},..., x_{i_k})$, typically contain a single temporal attribute/variable, often in fixed (e.g., first) position: This arrangement simply reflects the choice of the underlying temporal data model to be the single-dimensional valid time model. However, TRC can be similarly defined for multidimensional temporal data models (such as the bitemporal model) or for models without a predefined number of temporal attributes by appropriately modifying or relaxing the requirements on the structure of relation schemas.

An interesting observation is that a variant of TRC, in which temporal variables range over *intervals* and that utilizes *Allen's interval relations* as basic comparisons between interval values, is equivalent to TRC over two-dimensional temporal relations, with the two temporal attributes standing for interval endpoints.

Cross-references

- ► Abstract Versus Concrete Temporal Query Languages
- ► Point-Stamped Temporal Models
- ► Relational Calculus
- ► Relational Model
- ► Temporal Logic in Database Query Languages
- ► Temporal Query Languages
- ► Temporal Relation
- ► Time Domain
- ► Time Instant
- ► TSQL2
- ► Valid Time

Recommended Reading

1. Chomicki J. and Toman D. Temporal databases. In Handbook of Temporal Reasoning in Artificial Intelligence. M. Fischer, D. Gabbay, and L. Villa Foundations of Artificial Intelligence. Elsevier, New York, NY, USA, 2005, pp. 429–467.

Temporal Restriction

▶ Temporal Specialization

Temporal Semi-Structured Data

▶ Temporal XML

Temporal Specialization

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms
Temporal restriction
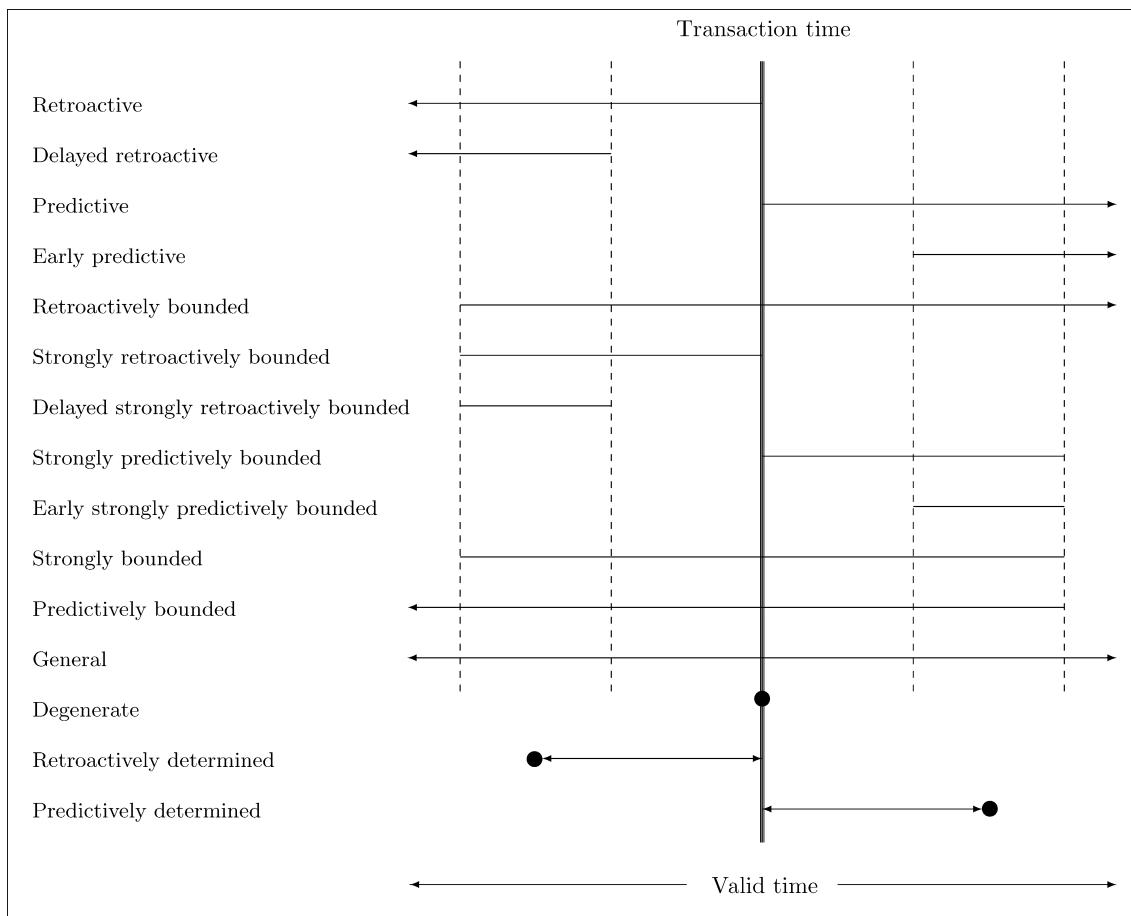
Definition
Temporal specialization denotes the restriction of the interrelationships between otherwise independent (implicit or explicit) timestamps in temporal relations. An example is a relation where tuples are always inserted after the facts they record were valid in reality. In such a relation, the transaction time of a tuple would always be after the valid time. Temporal specialization may be applied to relation schemas, relation instances, and individual tuples.

Key Points
Data models exist where relations are required to be specialized, and temporal specializations often constitute important semantics about temporal relations that may be utilized for, e.g., improving the efficiency of query processing.

Temporal specialization encompasses several kinds of specialization. One is based on the relationships

Temporal Specialization. Figure 1. Temporal Specialization based on isolated events – restrictions on the valid timestamp relative to the transaction timestamp. Adapted from Jensen and Snodgrass (1994).

between isolated events, and one based on inter-event relationships. Two additional kinds consider intervals instead of events, and one is based on the so-called completeness of the capture of the past database states.

The taxonomy based on *isolated events*, illustrated in Fig. 1, considers the relationship between a single valid time and a single transaction time. For example, in a *retroactive* relation, an item is valid before it is operated on (inserted, deleted, or modified) in the database. In a *degenerate* relation, there is no time delay between sampling a value and storing it in the database. The valid and transaction timestamps for the value are identical.

The interevent-based taxonomy is based on the *interrelationships* among multiple event timestamped items, and includes non-decreasing, non-increasing, and sequential. Regularity is captured through the categories of transaction time event regular, valid time event regular, and temporal event regular, and strict versions of these. The interinterval-based taxonomy uses Allen's relations.

To understand the last kind of specialization, which concerns the *completeness* of the capture of past states, recall that a standard transaction — time database captures all previously current states – each time a database modification occurs, a new previously current state is created. In contrast a valid-time database captures only the current database state. In-between these extremes, one may envision a spectrum of databases with incomplete support for transaction time. For example, consider a web archive that takes a snapshot of a collection of web sites at regular intervals, e.g., every week. If a site was updated several times during the same week, states would be missing from the database. Such incomplete databases are considered specializations of more complete ones.

Concerning the synonym, the chosen term is more widely used than the alternative term. The chosen term indicates that specialization is done with respect to the temporal aspects of the data items being timestamped. It is natural to apply the term temporal generalization to the opposite of temporal specialization. "Temporal restriction" has no obvious opposite term.

Cross-references
▶ Allen's Relations
▶ Bitemporal Relation
▶ Temporal Database
▶ Temporal Generalization
▶ Transaction Time
▶ Valid Time

Recommended Reading
1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Jensen C.S. and Snodgrass R.T. Specialized temporal relations. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 594–603.
3. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 5(6):954–974, 1994.

Temporal Strata

Kristian Torp
Aalborg University, Aalborg, Denmark

Synonyms
Temporal layer; Layered architecture; Temporal middleware; Wrapper

Definition
A temporal stratum is an architecture for implementing a temporal DBMS. The stratum is a software layer that sits on top of an existing DBMS. The layer translates a query written in a temporal query language into one or more queries in a conventional query language (typically SQL). The translated queries can then be executed by the underlying DBMS. The DBMS returns the result of the query/queries to the user directly or via the stratum. The core idea of the stratum is to provide new temporal query functionality to the users without changing the underlying DBMS. A temporal stratum can be implemented as a simple translator (temporal SQL to standard SQL) or as an advanced software component that also does part of the query processing and optimization. In the latter case, the temporal stratum can implement query processing algorithms that take the special nature of temporal data into consideration. Examples are algorithms for temporal join, temporal coalescing, and temporal aggregation.

Historical Background

Applications that store and query multiple versions of data have existed for a very long time. These applications have been implemented using, for example, triggers and log-like tables. Supporting multiple versions of data can be time consuming to build and computationally intensive to execute, since the major DBMS vendors only have limited temporal support. See [8] for details.

Parallel to the work in the software industry the research community has proposed a large number of temporal data models and temporal query languages. For an overview see [7]. However, most of this research is not supported by implementations. A notable exception to this is the Postgres DBMS that has built-in support for transaction time [6]. Postgres has later evolved into the popular open-source DBMS PostgreSQL that does not have transaction-time support. Most recently, the Immortal DB research prototype [2] has looked at how to built-in temporal support in the Microsoft SQL Server DBMS.

The DBMS vendors have not been interested in implementing the major changes to the core of their DBMSs that are needed to add temporal support to the existing DBMSs. In addition, the proposal to add temporal support to SQL3, called "SQL/Temporal" part 7 of the ISO SQL3 standard, has had very limited support. The temporal database research community has therefore been faced with a challenge of how to experimentally validate their proposals for new temporal data models and temporal query languages since it is a daunting task to build a temporal DBMS from scratch.

To meet this challenge it has been proposed to implement a temporal DBMS as a software layer on top of an existing DBMS. This has been termed a temporal stratum approach. Some of the first proposals for a temporal stratum mainly consider translating a query in a temporal query language to one or more queries in SQL [9].

The temporal database research community has shown that some temporal queries are very inefficient to execute in plain SQL, either formulated directly in SQL or translated via a simple temporal stratum from a temporal query language to SQL. For this reason it has be researched how to make the temporal stratum approach more advanced such that it uses the underlying DBMS when this is efficient and does the query execution in the layer when the underlying DBMS is found to be inefficient [5,4].

Scientific Fundamentals

A bitemporal database supports both valid time and transaction time. In the following it is assumed that all tables have bitemporal support. Figure 1a shows the bitemporal table emp. The table has two explicit columns, name and dept, and four implicit timestamp columns: VTS, VTE, TTS, and TTE. The first row in the table says that Jim was in the New York department from the third (assuming the month of September 2007) and is still there, indicated by the variable *now*. This information was also entered on the third and is still considered to be the best valid information, indicated by the variable *uc* that means until changed.

It is straightforward to implement a bitemporal table in a conventional DBMS. The implicit attributes are simply made explicit. For the emp table an SQL table with six columns are created. This is shown in Fig. 1b. Existing DBMSs do not support variables and therefore the temporal stratum has to convert the variables *now* and *uc* to values in the domain of columns VTE and TTE. It has been shown that it is the most convenient to use the maximum value (9999-12-31) in the date domain for both *now* and *uc* [10].

Primary key and unique key constraints are quite complicated to implement in a temporal stratum.

Name	Dept	VTS	VTE	TTS	TTE
Jim	NY	3	*now*	3	*uc*
Joe	LA	4	*now*	4	11
Joe	LA	4	11	11	*uc*
Joe	UK	11	*now*	11	*uc*
Sam	NY	12	*now*	12	14
Sam	NY	12	14	14	*uc*

```
Create table emp(
    name   varchar2 (30)   not   null,
    dept   varchar2 (30)   not   null,
    vts    date            not   null,
    vte    date            not   null,
    tts    date            not   null,
    tte    date            not   null)
```

Temporal Strata. Figure 1. (a) The Bitemporal Table emp (b) SQL Implementation.

A temporal query language such as ATSQL [1] has to be temporal upwards compatible, i.e., all non-temporal SQL statements have to work as before. For primary keys this means that a primary key on a bitemporal table cannot be directly mapped to a primary key in the underlying DBMS. As an example, if the `emp` only has to store the current version of where employees are, the `name` column can be used as a primary key. However, since `emp` has bitemporal support there are now three rows in the example table where the name is "Joe." Due to space constraints, an example is not listed here. Please see [8] for a concrete example.

To look at how modifications are handled in a temporal stratum, first consider the temporal insert statement shown in Fig. 2a. This is a temporal upward compatible insert statement (it looks like a standard SQL statement). The mapping to SQL is shown in Fig. 2b. The columns `vts` and `tts` are set to the current date (fourth of September 2007). The *now* and *uc* variables are set to the maximum `date`. The result is the second row in Fig. 1a.

At time 11 Joe is updated from being in the LA department to the UK department. The temporal upwards compatible update statement for this is shown in Fig. 3a. This update statement is mapped to an SQL update of the existing row and two SQL insert statements. The update ends the current belief by updating the `TTE` column to the current date. The first SQL insert statement stores for how long it was believed that Joe was in the LA department. The second SQL insert statement stores the new belief that Joe is in the UK department. The temporal update corresponds to the updated second row plus the third and fourth row in Fig. 1a. A delete statement is mapped like the SQL update statement and the first SQL insert statement in Fig. 3b. In Fig. 1a a temporal insert of Sam at time 12 and a temporal delete of Sam at time 14 are shown as rows number 5 and 6. Note that the SQL delete statement is never used for mapping temporal modification statements. For a complete coverage of implementing temporal modification statements in a temporal stratum, please see [10].

It is possible to see past states of the database. In particular the following will look at the `emp` table as of the 13th. This is called a time slicing, i.e., the database is rewound to 13th to see the content of the database as of this date.

The ATSQL query in Fig. 4a is a sequenced query that selects the explicit attribute and the valid-time attributes (`vts` and `vte`). The equivalent SQL query translated by a temporal stratum is shown in Fig. 4b. The result of the query is shown in Fig. 4c (using *now*

```
-- at time 4
insert into emp value ('Joe', 'LA')
```
a

```
insert into emp values
   ( 'Joe',  'LA',
     '2007-09-04', '9999-12-31',
     '2007-09-04', '9999-12-31');
```
b

Temporal Strata. Figure 2. (a) Temporal Insert (b) Mapping to SQL.

```
-- at time 11
update set dept = 'UK'
where name = 'Joe'
```
a

```
-- stop the old row
update emp set
   vte = '2007-09-11'
where   vts     <=  '2007-09-11'
and     vte     >   '2007-09-11'
and     tte     =   '9999-12-31'
and     name    =   'Joe';
-- insert knowledge about the past
insert into emp values
   ('Joe',' LA', '2007-09-04', '2007-09-11',
                 '2007-09-11', '9999-12-31');
-- insert new knowledge
insert into emp values
   ('Joe', 'UK', '2007-09-11', '9999-12-31',
                 '2007-09-11', '9999-12-31');
```
b

**Temporal Strata. Figure 3.** (a) temporal update (b) mapping to SQL.
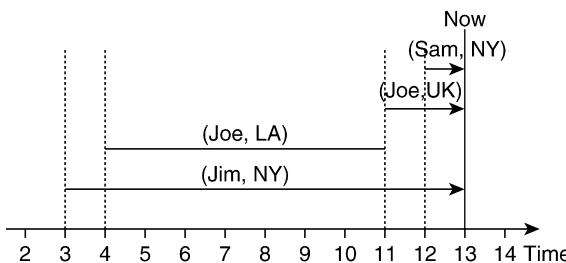
Temporal Strata. Figure 4. Transaction-time slicing.

Temporal Strata. Figure 5. Visual representation of the content in Fig. 4c.

Temporal Strata. Figure 6. Temporal aggregation

instead of the maximum date). The temporal SQL query is only slightly simpler than the equivalent standard SQL query.

To see the benefits of a temporal stratum it is necessary to look at more complicated queries. In the following, the focus is on temporal aggregation queries. Alternative complicated queries are temporal join or temporal coalescing.

Assume that a boss wants to see how many employees a company has had over (valid) time looking at the database as of the 13th. The result shown in Fig. 4c is used as an input for this query. For convenience it is assumed that this timeslice query is converted to a view call empAt13.

The content of the database is illustrated in Fig. 5, where the vertical dotted lines indicates the time where the result has to be split. The temporal query that expresses this is shown in Fig. 6a and the result of the query is shown in Fig. 6b. Note that the result is not coalesced.

To execute this query the temporal stratum has to find the constant periods, i.e., the periods where the count is the same. Here the temporal stratum can do either direct conversion to standard SQL or do part of the query processing in the stratum. The direct converted query is listed in Fig. 7. The standard SQL is

very complicated compared to the equivalent temporal SQL query in Fig. 6a. The benefit for the user should be obvious. In line 1 of Fig. 7, the count and the valid-time start and end associated with the count are selected. In line 2 the view empat13 from Fig. 4b is used. In addition, the const_period is introduced in lines 2–39. In line 40, only those periods that overlap the group currently being considered are included. In line 41, the groups are formed based on the valid-time start and valid-time end. Finally, in line 42 the output is listed in the valid-time start order.

The next question is then the efficiency of executing the query in the underlying DBMS or doing part of the query processing in the temporal stratum. It has been experimentally shown that for temporal aggregation, it can be up to ten times more efficient to do part of the query processing in a temporal stratum [5].

A different approach to adding temporal support to an existing DBMS is to use an extensible DBMS such as IBM Informix, Oracle, or DB2. This is the approach taken in [11]. Here temporal support is added to IBM Informix. Compared to a stratum approach it is not possible in the extension approach to use a new temporal SQL. The temporal extension are accessed by the user via new operators or function calls.

Key Applications

There is no support for valid-time or transaction-time in existing DBMSs. However, in Oracle 10g there is a flashback option [3] that allows a user to see the state

```
1   select  count(name),   const_period.vts as vts,  const_period.vte as vte
    from   empat13, (select   t1.vts as vts, t1.vte as vte
3               from      empat13 t1
                where    not exists ( select *
5                        from     empat13 t2
                        where  (t1.vts < t2.vts and t2.vts < t1.vte) or
7                               (t1.vts < t2.vte and t2.vte < t1.vte))
                union
9               select   t1.vts as vts, t2.vts as vte
                from     empat13 t1, empat13 t2
11              where    t1.vts < t2.vts and t2.vts < t1.vte
                and      not exists (select *
13                       from     empat13 t3
                        where  (t1.vts < t3.vts and t3.vts < t2.vts) or
15                              (t1.vts < t3.vte and t3.vte < t2.vts))
                union
17              select   t1.vts as vts, t2.vte as vte
                from     empat13 t1, empat13 t2
19              where    t1.vts < t2.vte and t2.vte < t1.vte
                and      not exists (select *
21                       from     empat13 t3
                        where  (t1.vts < t3.vts and t3.vts < t2.vte) or
23                              (t1.vts < t3.vte and t3.vte < t2.vte))
                union
25              select   t1.vte as vts, t2.vts as vte
                from     empat13 t1, empat13 t2
27              where    t1.vte < t2.vts
                and      not exists (select *
29                       from     empat13 t3
                        where  (t1.vte < t3.vts and t3.vts < t2.vts) or
31                              (t1.vte < t3.vte and t3.vte < t2.vts))
                union
33              select   t1.vte as vts, t2.vte as vte
                from     empat13 t1, empat13 t2
35              where    t2.vts < t1.vte and t1.vte < t2.vte
                and      not exists (select *
37                       from     empat13 t3
                        where  (t1.vte < t3.vts and t3.vts < t2.vte) or
39                              (t1.vte < t3.vte and t3.vte < t2.vte))) const_period
    where empat13.vts <= const_period.vts and const_period.vte <= empat13.vte
41  group by const_period.vts, const_period.vte
    order by const_period.vts
```

Temporal Strata. Figure 7. Temporal aggregation in standard SQL.

```
flashback table emp to timestamp ('2007-09-01 08:00:00');
```

Temporal Strata. Figure 8. A flashback in the Oracle DBMS.

of the entire database or a single table as of a previous instance in time. As an example, the flashback query in Fig. 8 will get the state of the emp table as of the 1st of September 2007 at 08.00 in the morning. The result can be used for further querying.

Future Directions

The Sarbanes-Oxley Act is a US federal law that requires companies to retain all of there data for a period of five or more years. This law may spur additional research with temporal databases and in particular temporal database architecture such as a temporal stratum.

Cross-references

► Databases
► Supporting Transaction Time
► Temporal Data Model

► Temporal Query Languages
► Temporal Query Processing

Recommended Reading

1. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, 2000.
2. Lomet D., Barga R., Mokbel M.F., Shegalov G., Wang R., and Zhu Y. Transaction time support inside a database engine. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
3. Oracle Corp. Oracle Flashback Technology. http://www.oracle.com/technology/deploy/availability/htdocs/Flashback_Overview.htm, as of 4.9.2007.
4. Slivinskas G. and Jensen C.S. Enhancing an extensible query optimizer with support for multiple equivalence types. In Proc. 5th East European Conf. Advances in Databases and Information Systems, 2001, pp. 55–69.
5. Slivinskas G., Jensen C.S., and Snodgrass R.T. Adaptable query optimization and evaluation in temporal middleware. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 127–138.
6. Stonebraker M. and Rowe L.A. The design of POSTGRES. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 340–355.
7. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer Academic, Dordrecht, 1995.
8. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 1999.
9. Torp K., Jensen C.S., and Snodgrass R.T. Stratum approaches to temporal DBMS implementation. In Proc. Int. Conf. on Database Eng. and Applications, 1998, pp. 4–13.
10. Torp K., Jensen C.S., and Snodgrass R.T. Effective timestamping in databases. VLDB J., 8(3–4):267–288, 2000.
11. Yang J., Ying H.C., and Widom J. TIP: a temporal extension to informix. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.

Temporal Structure

► Time Domain

Temporal Type

► Temporal Granularity

Temporal Upward Compatibility

► Current Semantics

Temporal Vacuuming

JOHN F. RODDICK[1], DAVID TOMAN[2]
[1]Flinders University, Adelaide, SA, Australia
[2]University of Waterloo, Waterloo, ON, Canada

Synonyms

Data expiration

Definition

Transaction-time temporal databases are inherently append-only resulting, over time, in a large historical sequence of database states. Data vacuuming allows for a strategic, and irrevocable, deletion of obsolete data.

Historical Background

The term *vacuuming* was first used in relation to databases in the Postgres database system as a mechanism for moving old data to archival storage [10]. It was later refined by Jensen and Mark in the context of temporal databases to refer to the removal of *obsolete* information [4] and subsequently developed into a comprehensive and usable adjunct to temporal databases [6,8,11]. Data expiration has also been investigated in the context of data warehouses by Garcia-Molina et al. [2] and others [9].

Foundations

In many applications, data about the past needs be retained for further use. This idea can be formalized, at least on the conceptual level, in terms of an append-only transaction-time temporal database or a history. However, a naive and unrestricted storage of all past data inevitably leads to unreasonable demands on storage and subsequently impacts negatively on efficiency of queries over such histories. Hence techniques that allow selective removal of *no longer needed* data have been developed and, at least in prototype systems, deployed.

The parts of the historical data that are to be retained/deleted are specified in terms of *vacuuming specifications*. These specifications state, for example, that data beyond certain absolute or relative time point is *obsolete* (as opposed to merely *superceded*) and thus can be removed. For example, the regulation

► *"Taxation data must be retained for the last 5 years"*

can be considered a specification of what data individuals must retain concerning their taxation returns and what can be discarded. However, one must be

careful when designing such specifications as once a part of the history is deleted, it can no longer be reconstructed from the remaining data. Consider the alternative regulation

▶ *"Taxation data must be retained for past years, except for the last year."*

While this specification seems to suggest that the data of the last year can be discarded, doing so would lead to a problem in the following year (as this year's return won't be the last one any more). Hence, this specification is intuitively not well formed and must be avoided. Vacuuming specifications can be alternatively phrased in terms of what may be deleted, rather than what should be retained. For example, rather than

▶ *"Taxation data must be retained for the last 5 years"*

it would be better to rephrase it as

▶ *"Taxation data over 5 years old may be deleted."*

The reason for this is that vacuuming specifications indicate specific deletion actions and there is thus less chance of misinterpretation.

Another issue with vacuuming specifications relates to the granularity of data that is removed from the history. For example, if data items (such as tax receipts) are temporally correlated with other items that may appear in different, perhaps much older, parts of the history (such as investments), those parts of the history may have to be retained as well. Such considerations must be taken into account when deciding whether the specifications are allowed to refer to the complete history or whether selective vacuuming of individual data items is permitted. In both cases, issues of data integrity need to be considered.

Formal Vacuuming Specifications

A transparent way to understand the above issues is to consider the result of applying a vacuuming specification to a history (i.e., the *retained* data) to be a view defined on the original history.

Definition. Let $H = \langle S_0, S_1, ..., S_k \rangle$ be a history. The instances S_i represent the state of the data at time i and all states share a common fixed schema. $\mathbf{T}_H$ and $\mathbf{D}_H$ are used to denote the *active* temporal and data domains of H, respectively. A *vacuuming specification* is a function (a view) $E : H \rightarrow H'$, where H' is called the *residual history* (with some, potentially different but fixed schema).

The idea behind this approach is that the instance of the view represents the *result* of applying the vacuuming specification to the original history and *it is this instance that has to be maintained* in the system. While such view(s) usually map histories to other histories (sometimes called the *residual histories*), in principle, there is no restriction on the schema of these views nor on the language that defines the view. This approach allows us to address the two main questions concerning a vacuuming specification:

- **Is a given specification well formed?** The first question relates to anomalies such as the one outlined in the introductory example. Since only the instance of the view and not the original history itself is *stored*, a new instance of the view must be definable in terms of the current instance of the view $E(H)$ whenever a new state of the history S is created by progression of time. This condition can be formalized by requiring:

$$E(H; S) = \Delta(E(H), S)$$

for some function (query) Δ where $H;S$ is the extension of H with a new state S. To start this process, a constant, $\emptyset$, is technically needed to represent the instance of the view in the beginning (i.e., for an empty initial history). The condition above essentially states that the view E must be *self-maintainable* in terms of the pair $(\emptyset, \Delta)$. The pair $(\emptyset, \Delta)$ is called a *realization* of E.

- **What queries does a specification support?** The second question concerns which queries can be correctly answered over the residual histories. Again, for a query Q to be answerable, it must be the case that

$$Q(H) = Q'(E(H))$$

for some function (query) Q' and all histories H. Q' is a *reformulation* of Q with respect to E. This requirement states that queries preserved by the vacuuming process are exactly those that can be answered only using the view E.

In addition, for the approach to be *practical*, the construction of Δ and $\emptyset$ from E and of Q' from Q and E, respectively, must be effective.

Definition. A vacuuming specification represented by a self-maintainable view E is a *faithful history encoding for a query Q* if Q is answerable using the view E.

Given a vacuuming specification E over a history H that is self-maintainable using $(\emptyset, \Delta)$ and a query Q' that answers Q using E; the triple $(\emptyset, \Delta, Q')$ is then called the *expiration operator* of Q for H.

Space/Storage Requirements

Understanding vacuuming specifications in terms of self-maintainable materialized views also provides a natural tool for comparing different specifications with respect to *how well they remove unnecessary data*. This can be measured by studying the size of the instances of E with respect to several parameters of H:

- The size of the history itself, $|H|$,
- The size of the active data domain, $|\mathbf{D}_H|$, and
- The length of the history, $|\mathbf{T}_H|$.

In particular, the dependency of $|E(H)|$ on $|\mathbf{T}_H|$ is important as the progression of time is often the major factor in the size of H. It is easy to see that vacuuming specification with a *linear* bound in terms of $\mathbf{T}_T$ always exists: it is, e.g., the identity used to define both E and Q'. However, such a specification is not very useful and better results can be probably achieved using standard *compression algorithms*. Therefore the main interest is in two main cases defined in terms of $|\mathbf{T}_H|$:

1. Specifications bounded by O(1), and
2. Specifications bounded by O($log(|\mathbf{T}_H|)$).

In the first case the vacuuming specification provides a *bounded encoding of a history*. Note that in both cases, the size of $E(H)$ will still depend on the other parameters, e.g., $|\mathbf{D}_H|$. This, however, must be expected, as intuitively, the more H refers to different constants (individuals), the larger $E(H)$ is likely to be (for example, to store the *names* of the individuals).

Vacuuming in Valid-Time Databases. In contrast to transaction-time temporal databases (or histories), valid-time temporal databases allow *arbitrary* updates of the temporal data. Hence information about future can be recorded and data about the past can be modified and/or deleted. This way, vacuuming specifications reduce to appropriate updates of the valid time temporal database.

Moreover, when allowing arbitrary updates of the database, it is easy to show that the only *faithful history encodings* are those that are lossless (in the sense that H can be reconstructed from the instance of E).

Example. Consider a valid time temporal database H with a schema $\{R\}$ and a query Q asking "*return the contents of the last state of R recorded in H.*" Then, for a vacuuming specification E to be a *faithful encoding* of H (w.r.t. Q), it must be possible to answer Q using only the instance of E after updating of H. Now consider a sequence of updates of the form "*delete the last state of R in H.*" These updates, combined with Q, can *reconstruct* the contents of R for an arbitrary state of H. This can only be possible if E is lossless.

This, however, means that any such encoding must occupy roughly the same storage as the original database, making vacuuming useless. Similar results can be shown even for valid time databases in which updates are restricted to insertions.

Approaches to Vacuuming

The ability to vacuum data from a history depends on the expressive power of the query language in which queries over the history are formulated and on the number of the actual queries. For example, allowing an arbitrary number of *ad-hoc* queries precludes any possibility effective vacuuming of data, as finite relational structures can be completely characterized by first-order queries. Thus, for common temporal query languages, this observation leaves us with two essential options:

1. An *administrative* solution is adopted and a given history is vacuumed using a set of *policies* independent of queries. Ad-hoc querying of the history can be allowed in this case. However, queries that try to access already expired values (i.e., for which the view is not faithful history encoding) have to fail in a predefined manner, perhaps by informing the application that the returned answer may be only approximate, or

2. A query driven data expiration technique is used. Such a technique, however, can only work for a fixed set of queries *known in advance*.

Administrative Approaches to Vacuuming

One approach to vacuuming data histories, and, in turn, to defining *expiration operators*, can be based on *vacuuming specifications* that define query/application-independent policies. However, when data are removed from a history in such a way, the system should be able to characterize queries whose answers are not affected. A particular way to provide vacuuming specifications (such as through the ideas of Skyt et al. [6,8]) is using *deletion* (ρ) and *keep* (κ)

expressions. These would be invoked from time to time, perhaps by a vacuuming daemon. The complete specification may contain both deletion and keep specifications. For example:

$$\rho(EmpDep) : \sigma_{TT_{end} \leq NOW-1yr}(EmpDep)$$
$$\kappa(EmpDep) : \sigma_{EmpStatus='Retain'}(EmpDep)$$
$$\rho(EmpDep) : \sigma_{VT_{end} \leq NOW-7yrs}(EmpDep)$$

This specification states that unless the Employee has a status of "Retain," all corrected data should be vacuumed after 1 year and all superceded data vacuumed after 7 years. For safety, keep specifications always override delete specifications (note that the ordering of the individual deletion and keep expressions is significant).

Vacuuming in Practice. Vacuuming specifications are generally given as either part of the relation definition or as a stand-alone vacuuming specification. In TSQL2, for example, a CREATE TABLE command such as:

```
CREATE TABLE EmpDep (
    Name            CHAR(30) NOT NULL,
    Dept            CHAR(30) NOT NULL,
AS TRANSACTION YEAR(2) TO DAY
VACUUM NOBIND (DATE 'now - 7 days');
```

specifies *inter alia* that only queries referencing data valid within the last 7 days are permissible [3] while

```
CREATE TABLE EmpDep ( ... )
  VACUUM DATE '12 Sep 2007';
```

specifies that only query referencing any data entered on or after 12 September 2007 are permissible. The VACUUM clause provides a specification of what temporal range constitutes a valid query with the NOBIND keyword allowing DATE to be the date that the query was executed (as opposed to the date that the table was created).

An alternative is to allow tuple-level expiration of data. In this case, the expiration date of data are specified on insert. For example, in the work of Schmidt et al. [5] users might enter:

```
INSERT INTO EmpDep
VALUES ('Plato', 'Literature', ...)
EXPIRES TIMESTAMP '2007-09-12 23:59:59';
```

to indicate that the tuple may be vacuumed after the date specified.

Application/Query-Driven Approaches
There are many applications that collect data over time but for which there are no *natural* or *a priori* given vacuuming specifications. However, it is still important to control the size of the past data needed. Hence, it is a natural question whether appropriate specifications can be derived *from the (queries in the) applications* themselves (this requires an *a priori* fixed *finite* set of queries – in the case of ad-hoc querying such a specification cannot exist. Formally, given a query language $\mathcal{L}$, a computable mapping of queries $Q \in \mathcal{L}$ to triples $(\emptyset, \Delta, Q')$, such that $(\emptyset, \Delta, Q')$ is an expiration operator for Q over H, has to be constructed. Figure 1 summarizes the results known for various temporal query languages and provides references to the actual techniques and proofs.

Key Applications
The major application domains for vacuuming are historical databases (that, being append only, need a mechanism to limit their size), logs (particularly those collected for more than one purpose with different statutes and business processes), monitoring applications (with rollback requirements) and garbage collection (in programming languages). Also, as *data streams* are essentially *histories*, the techniques and results

Temporal Query Language	Lower bound	Upper bound	Reference
Past FO Temporal Logic	bounded	$POLY(\mathbf{D}_H)$	[1]
Past Temporal μ-Calculus	bounded	$POLY(\mathbf{D}_H)$	[12]
Temporal Relational Calculus	bounded	$ELEM(\mathbf{D}_H)$	[11]
Future Temporal μ-Calculus	$\Omega(|\mathbf{T}_H|)$	$O(|H|)$	[14]
Propositional Past TL w/duplicates	$\Omega(|\mathbf{T}_H|)$	$O(|H|)$	[14]
Past Temporal μ-Calculus w/bounded duplicates	$\Omega(|\log(\mathbf{T}_H|))$	$\Omega(|\log(\mathbf{T}_H|))$	[14]
Conjunctive Past TL Queries w/duplicates	$\Omega(|\log(\mathbf{T}_H|))$	$\Omega(|\log(\mathbf{T}_H|))$	[14]
Past TL Queries w/counting	$\Omega(|\log(\mathbf{T}_H|))$	$O(|H|)$	[13]

Temporal Vacuuming. Figure 1. Space bounds for residual histories.

developed for vacuuming and data expiration can be applied to query processing over data streams. In particular, expiration operators for a given query yield immediately a *synopsis* for the same query in a streaming setting. This observation also allows the transfer of the space complexity bounds.

Future Directions

Most approaches have concentrated on specifying what data to retain for given queries to continue to be answered perfectly. There are two other possibilities:

- Given a particular vacuuming specification and a query that is not supported fully by this specification, can the degree can this query be answered by the residual history be determined? Some suggestions are given by Skyt and Jensen [7] who propose that queries that may return results affected by vacuuming should also provide suggestions for an alternative, similar query.
- Given a requirement that certain queries should not be answered (e.g., for legal reasons), what would be the vacuuming specifications that would guarantee this, in particular in the conjunction with the issue of approximate answers above?

Both of these are areas for further research. Finally, most vacuuming research assumes a static schema definition (or at least, an overarching applicable schema definition). Having the versioning of schema while also handling the vacuuming of data is also an open problem.

Cross-references

▶ Point-Stamped Temporal Models
▶ Query Rewriting Using Views
▶ Schema Versioning
▶ Self-Maintenance of Views
▶ Synopses for Data Streams
▶ Temporal Query Languages

Recommended Reading

1. Chomicki J. Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst., 20(2):149–186, 1995.
2. Garcia-Molina H., Labio W., and Yang J. Expiring data in a warehouse. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 500–511.
3. Jensen C. Vacuuming. In The TSQL2 Temporal Query Language, Chapter 23, R. Snodgrass (ed.). Kluwer, New York, 1995, pp. 451–462.
4. Jensen C.S. and Mark L. A framework for vacuuming temporal databases. Tech. Rep. CS-TR-2516, University of Maryland at College Park, 1990.
5. Schmidt A., Jensen C., and Saltenis S. Expiration times for data management. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 36.
6. Skyt J. Specification-Based Techniques for the Reduction of Temporal and Multidimensional Data. Ph.D thesis, Aalborg University, Aalborg, Denmark, 2001.
7. Skyt J. and Jensen C.S. Vacuuming temporal databases. Time-Center technical report TR-32, Aalborg University, 1998.
8. Skyt J., Jensen C.S., and Mark L. A foundation for vacuuming temporal databases. Data Knowl. Eng., 44(1):1–29, 2003.
9. Skyt J., Jensen C.S., and Pedersen T.B. Specification-based data reduction in dimensional data warehouses. In Proc. 18th Int. Conf. on Data Engineering, 2002, p. 278.
10. Stonebraker M. and Rowe L. The design of POSTGRES. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 340–355.
11. Toman D. Expiration of historical databases. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 128–135.
12. Toman D. Logical data expiration for fixpoint extensions of temporal logics. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 380–393.
13. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2003, pp. 99–106.
14. Toman D. On construction of holistic synopses under the duplicate semantics of streaming queries. In Proc. 14th Int. Symp. Temporal Representation and Reasoning, 2007, pp. 150–162.

Temporal Value

▶ History in Temporal Databases

Temporal Visual Interfaces

▶ Temporal Visual Languages

Temporal Visual Languages

ULRICH SCHIEL[1], SONIA FERNANDES SILVA[2]
[1]Federal University of Campina Grande, Campina Grande, Brazil
[2]Etruria Telematica Srl, Siena, Italy

Synonyms

Temporal visual queries; Temporal visual interfaces

Definition

Database technology has evolved in order to be typically oriented towards a large set of non-expert users. While attempting to meet this need, textual query languages, such as SQL, have been replaced by visual query languages, which are based on visual representations of the database and direct manipulation mechanisms. Moreover, data characterized by the temporal dimension play an important role in modern database applications. Temporal Visual Languages are user-oriented languages that meet the specific requirements of querying and visualizing temporal data in an interactive and easy-to-use visual form.

Historical Background

The availability of graphical devices at low cost and the advent of the direct manipulation paradigm [10] have given rise in the last years to a large diffusion of visual user interfaces. Regarding the database area, databases are designed, created, and possibly modified by experts, but there are different kinds of users whose job requires access to databases, specifically for extracting information. However, traditional query languages for databases, such as SQL, are not very approachable for these users, for both their intrinsic syntactical complexity and the lack of a global view of the data of interest together with their interrelationships. Thus, visual interfaces for databases, in particular, the so-called Visual Query Systems (VQS) [4], have arisen as an evolution of the traditional query languages.

VQS include both a language to express queries in a visual form and a query strategy. They are oriented to a wide spectrum of users who generally ignore the inner structure of the accessed database and are characterized by several notable features, such as the availability of interactive visual mechanisms that facilitate the typical process of query formulation and refinement, without requiring users to have a previous knowledge of the database schema and to learn the syntax and semantics of a query language.

Moreover, it has been pointed out that modern database applications deal with temporal data such as banking, medical records, airline reservations, financial data, decision support systems, etc. Several proposals of temporal query languages have been carried out in the past years, where special clauses and predicates are added to the original language in order to deal with the temporal aspects. These languages increase the usability problems of the originating query languages,

including quite complex syntax and a steep learning curve. For instance, considering specifically the temporal relational languages, the user must be familiar with concepts such as *tuple* and *attribute time-stamping*, temporal joins, as well as syntax and semantics of temporal predicates. As efforts were made to find new visual query mechanisms for accessing conventional databases, this should also be done for temporal databases.

In order to address this need, some proposals of temporal visual query languages have arisen in the last years. Initial proposals have mainly concentrated on the activity of query formulation, where temporal visual operators are interactively applied over diagrammatic representations of the database schemas exhibiting temporal features. However, most of these visual languages are not provided with formal syntax and semantics [13] and do not address the visual interaction with the query result. Trying to overcome this limitation, a variety of techniques for visualizing time-oriented data have been proposed [12], but they do not support query capabilities for extracting further information. In recent years, research efforts have been made in order to develop temporal visual interfaces in which the end-user could apply visual mechanisms for querying and visualizing temporal data in a single structure.

Foundations

Elements in temporal databases such as objects and their attributes may be temporal, meaning that the history of the successive values of a property is recorded or that the history of an object as a whole may be kept in the database. The valid time of a data element may be a single instant, a time interval or a set of disjoint time intervals.

The process of visual query formulation can be seen as constituted by three phases [13]: the user selects the part of the database he wants to operate on (location phase); then, he defines the relations/restrictions within the selected part in order to produce the query result (manipulation phase); finally, he operates on the query result (visualization phase). The same phases apply to a visual temporal query formulation.

In the *location phase* the goal is the precise definition of the fragment of the database schema involved in the query, known as *query subschema*. Many approaches of temporal visual languages adopt the visualization of the database schemas as Entity-Relationship (ER) diagrams extended with temporal entities

(classes) and temporal relationships. Following this approach, the selection of the subschema of interest may be done by selecting the classes, attributes, and relationships of interest [10], including the temporal ones. As an evolution of this ER-based approach a direct manipulation strategy has been proposed in [13], which adopts a "graphical notebook" metaphor for interacting with the database schema. Usability tests showed that this approach was enjoyed by the users, since they prefer to interact with something more familiar.
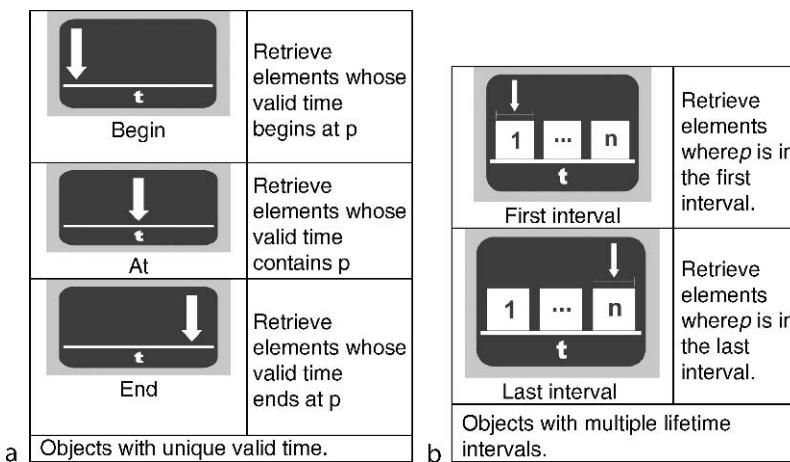
In the *manipulation phase*, the query subschema can be manipulated in several ways, according to the available query operators on which the detailed conditions of the query must be specified. A database query has two orthogonal components: *selection* and *projection*. In an analogy to a SQL SELECT-FROM-WHERE statement, the SELECT clause gives the data *projection* of the query result, the FROM clause states the *query subschema* of the location phase and the WHERE clause establishes the *selection* conditions on the database in order to retrieve the required data. Temporal queries encompass the possible combinations of current/temporal selection and current/temporal projection over time and data, resulting into nine different combinations beginning from data selection/data projection up to mixed selection/mixed projection [13]. For instance a query '*when did the employee Joseph move from department 1 to department 2, and what salary did he get at the new job?* ' is a data selection (Joseph and the two departments) and mixed projection (date of movement – time; new salary – data).
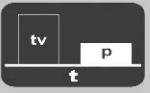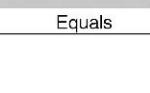
The question in temporal query processing is how to relate the (temporal-) data of the query to (temporal-) data of the database. This relationship is stated by comparing time-spans of the data lifetime that has been specified in the query with the corresponding lifetimes of objects or attributes in the database. This comparison may include special forms for expressing temporal restrictions or relations between the query and the database.

Temporal visual languages can use a set of visual representations for time and temporal relations, depending on the type of request: snapshot or slice [14]. Snapshot queries deal with facts that were valid at a particular time instant, whereas slices queries return facts that were valid over some period of time. Regarding snapshot queries, instants are visually represented as circles or small vertical bars. If p is a time instant (point) of the query and t is the valid time of an object (or attribute), the temporal relations between them can be visually represented as icons (Fig. 1).
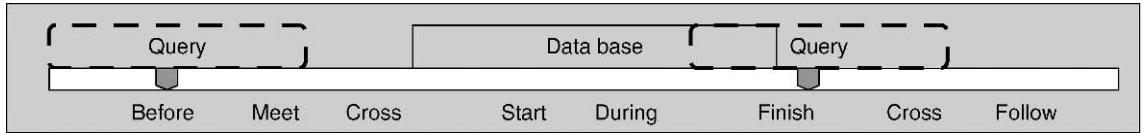
Regarding slice queries, the temporal relations between time intervals are based on Allen's Calculus of temporal intervals [3], which gives a complete set of possible relations between two (temporal) intervals. If both the database time tv and the query time p are intervals, the relational primitives of Allen can be visually represented as icons (Fig. 2). Instead of using icons for expressing temporal relations, a more intuitive

Temporal Visual Languages. Figure 1. Point/Interval relations as icons [5].

Temporal Visual Languages. Figure 2. Interval icons.

Temporal Visual Languages. Figure 3. Mobile slider [13].
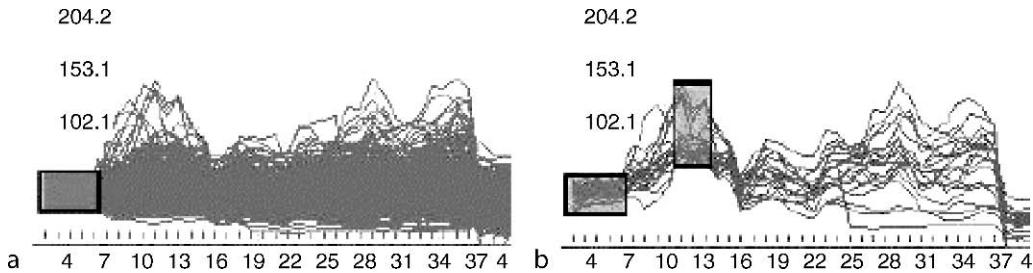
form of expressing such relations is to visually represent intervals as horizontal slide bars in order to dynamically specify the predicates of Allen. For instance, Silva et al. [13] have proposed a mobile slider, as illustrated in Fig. 3.

However, one of the most difficult problems in designing a temporal visual language is to achieve both high expressive power and ease-of-use. For example, the two representations illustrated above cannot deal with situations involving more than two intervals. In this case, the logical expressions between temporal relations (involving conjunctions and disjunctions) should be addressed in this manipulation phase. Some proposals address such a need by adopting visual metaphors for complex representation of temporal relations [6, 8] in a temporal query formulation. Considering that the endpoints may be imprecise or variable, Chittaro and Combi [6] propose three alternative

visual representations of flexible temporal intervals in order to deal with variable endpoints. Hibino and Rudensteiner [8] propose a bar with alternative endpoints, e.g., for specifying the temporal relation between two intervals starting at the same time but without restriction on the end, giving rise the logical expression *begins(A,B)* ∨ *equals(A,B)* ∨ *begins(B,A)*.

Moreover, other complex temporal relations such as temporal patterns should be also addressed. For instance, users might be interested in data related to events of arbitrary duration or events separated by arbitrary time gaps. The temporal visual language defined in [7] address such a need by presenting visual metaphors for expressing event-based queries, where constraints on events and inter-event time-spans are visually specified in different ways.

Finally in the *visualization phase,* the historical data retrieved from the query result is visualized for

Temporal Visual Languages. Figure 4. History with "TimeBox" [9].

interactive exploration and analysis. The most widely known visualization technique of time-oriented data are the interactive timeline [12], where the time is regarded as an ordinal axis in a bi-dimensional (2D) visualization and data are located at different positions along this time axis. Timelines and other visualization techniques can be categorized according to the generic criteria that address ontologies about the time, such as the temporal primitives that make up the time axis (time points and intervals), the temporal order (linear, cyclic or branching), etc. For instance, timeline visualization takes advantage of the linear ordered nature of time. An additional criterion is if time-oriented visualization supports the snapshot or slice views [12]. Other relevant criterion is the data tied to time axis [2], such as the data type which indicates if the data are abstract or spatial; the number of involved variables (univariate or multivariate) related to the data (temporal attributes and relationships); and its abstraction level (e.g., raw vs. aggregated data).

It is worth noting that for an effective exploration of data, visual techniques must be integrated with suitable interaction techniques, following the principle of visualization information mantra, defined in [11]: *overview first, zoom and filter, then details-on-demand*, where visual tools exploring the *dynamic query* approach [1] are well-known implementations of this principle. This means that starting from an overview of a large dataset, one may zoom and filter this overview to extract a data subset. Then, more details can be obtained from the selected data subset. In dynamic queries, the manipulation and visualization phases proceed iteratively in a visual query formulation. This means that, after visualization of a preliminary result, the user may interact with this result in order to refine the query.

Within this context, the visualization phase in temporal visual languages focuses on visual query and exploration of temporal trends and patterns within historical results from a preliminary query by using suitable interactive visualization techniques. When exploring such data, suitable interaction techniques such as the direct manipulation and brushing can be integrated with visual query capabilities. For instance, *TimeSearcher* [9] allow users to visually query and explore patterns in time-series data by using visual widgets for data filtering called "*TimeBoxes.*" *TimeBoxes* are rectangular query locators that specify the region(s) in which the users are interested. They are placed and directly manipulated on a 2D timeline, with the region boundaries providing the query parameters, as illustrated in Fig. 4. The extent of the *Timebox* on the time (x) axis specifies the time period of interest, while the extent on the value (y) axis specifies a constraint on the range of data values of interest. In this case, a query is dynamically created by drawing a box on the timeline. Multiple timeboxes can be combined to specify conjunctive queries. Only data sets that match all of the constraints implied by the timeboxes are visualized.

Key Applications

Temporal Visual Languages may be integrated with Spatial Visual Languages for Geographic Information Systems. Other typical applications are virtual reality, moving objects, or multimedia systems, such as video and audio data. It can be also an important concern in Visual Analytics.

In the medical field, there are many different applications needing temporality of patient records, images, examination events, and so on.

The classic application fields of Temporal Databases are systems of planning data, or systems of historic data, such as banking account, economic data, meteorological data, business histories, and many others. Also in Decision Support Systems, such as OLAP – Online Analytical Processing, time is the most important dimension.

Another promising application is related to the document management, which is based on time-changing texts, such as legal data or instructional texts. For instance, a judgment support system based on jurisprudence must consider the temporal context of past judgments.

Cross-references

▶ Data Visualization
▶ Lifespan
▶ Temporal Database
▶ Temporal Query Languages
▶ TSQL2
▶ Visual Interaction
▶ Visual Interfaces
▶ Visual Query Language

Recommended Reading

1. Ahlberg C. and Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1994, pp. 313–317.
2. Aigner W. et al. Visualizing time-oriented data – A systematic view. Comput. Graph., 31(3):401–409, 2007.
3. Allen J.F. Maintaining knowledge about temporal interval. Commun. ACM, 26(1):832–843, 1983.
4. Catarci T . et al. Visual query systems: analysis and comparison. J. Vis. Lang. Comput., 8(2):215–260, 1997.
5. Cavalcanti V.M.B., Schiel U., and Baptista C.S. Querying spatio-temporal databases using a visual environment. In Proc. Working Conf. on Advanced Visual Interfaces, 2006, pp. 412–419.
6. Chittaro L. and Combi C. Representation of temporal intervals and relations: information visualization aspects and their evaluation. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 13–20.
7. Fails J.A., Karlson A., and Shahamat L. Visual Query of Multi-Dimensional Temporal Data. http://www.cs.umd.edu/class/spring2005/cmsc838s/assignment-projects/visual-query-of-temporal-data/Final-Paper-06.pdf.
8. Hibino S. and Rundensteiner E.A. User interface evaluation of a direct manipulation temporal query language. In Proc. 5th ACM Int. Conf. on Multimedia, 1997, pp. 99–107.
9. Hochheiser H. and Shneiderman B. Dynamic query tools for time series data sets, timebox widgets for interactive exploration. Inf. Vis., 3(1):1–18, 2004.
10. Shneiderman B. Direct manipulation, a step beyond programming languages. IEEE Comput., 16(8):57–69, 1983.
11. Shneiderman B. The eyes have it: a task by data type taxonomy for information visualizations. In Proc. IEEE Symp. on Visual Languages, 1996, pp. 336–343.
12. Silva S.F. and Catarci T. Visualization of linear time-oriented data: a survey. In Proc. 1st Int. Conf. on Web Information Systems Eng., 2000, pp. 310–319.
13. Silva S.F., Catarci T., and Schiel U. Formalizing visual interaction with historical databases. Inf. Syst., 27(7):487–521, 2002.
14. Silva S.F., Schiel U., and Catarci T. Visual query operators for temporal databases. In Proc. 4th Int. Workshop Temporal Representation and Reasoning, 1997, pp. 46–53.

Temporal Visual Queries

▶ Temporal Visual Languages

Temporal XML

Curtis Dyreson[1], Fabio Grandi[2]
[1]Utah State University, Logan, UT, USA
[2]University of Bologna, Bologna, Italy
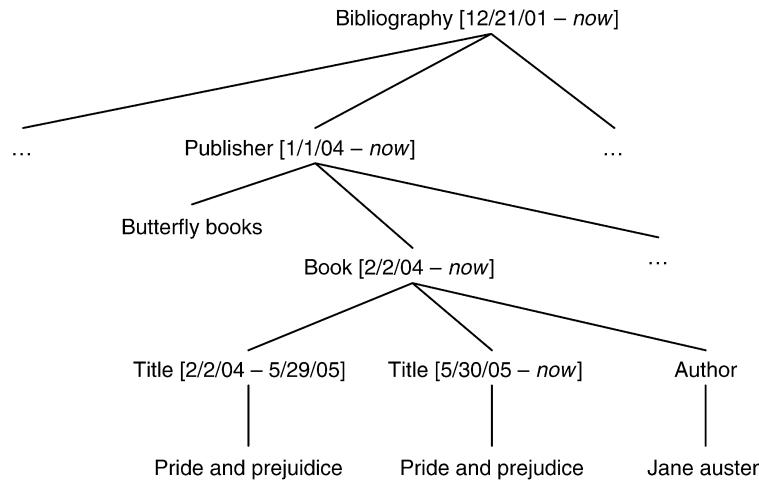
Synonyms
Temporal semi-structured data

Definition
Temporal XML is a *timestamped instance* of an XML data model or, more literally, an XML document in which *specially-interpreted* timestamps are present. In general, an XML data model instance is a tree or graph in which each node corresponds to an element, attribute, or value, and each edge represents the lexical nesting of the child in the parent's content. In temporal XML, a timestamp is added to some nodes or edges in the instance. The timestamp represents the lifetime of the node or edge in one or more temporal dimensions, usually valid time or transaction time. As an example, Fig. 1 shows a fragment of a temporal XML data model. The bibliographic data in the figure contains information about publishers, books, and authors. The figure also has timestamps that represent *when* each piece of data was entered into the data collection (i.e., the timestamps represent the *transaction-time* lifetime of each element). The bibliography began on Dec 21. 2001, and remains current (until *now*). Information about the Butterfly Books publisher was entered on Jan 1, 2004, and it started publishing a book by Jane Austen on Feb 2, 2004. The title of that book was originally misspelled, but was corrected on May 29, 2005. Alternatively, temporal XML is literally an XML document or data collection in which specially-interpreted timestamps, formatted

Temporal XML. Figure 1. A temporal XML fragment.

in XML, are included. Such a document yields a temporal XML data model instance when parsed.

Historical Background

XML is becoming an important language for data - representation and exchange, especially in web applications. XML is used to "mark-up" a data collection or document adding meaning and structure. The mark-up consists of elements inserted into the data. Usually an XML document is modeled as a tree in which each interior node corresponds to an element in the document and each leaf to a text value, attribute, or empty element. Temporal XML adds timestamps to the nodes and/or edges in the data model instance. The timestamps represent the lifetime of the nodes (edges).

Grandi has created a good bibliography of research in this area [8]. Chawathe et al. were the first to study time in an XML-like setting [2]. They encoded times in edge labels in a semi-structured database and extended the Lorel query language with temporal constructs. Dyreson et al. extended their research with collapsing and coalescing operators [5]. Grandi and Mandreoli presented techniques for adding explicit valid-time timestamps in an XML document [9]. Amagasa et al. next developed a temporal extension of the XML data model [1]. Following that, a range of temporal XML topics was investigated, from storage issues and indexing [3,11,12,13,14] to querying [6,7,12]. The timestamping of XML documents (or parts thereof) has also been considered in the more general context of

versioning of XML documents [11,15]. Finally, schemes for validating and representing times in XML documents have also been considered [4,10].

Foundations

It is important to distinguish between "the representation in XML of a time" and "temporal XML." Times are common in many XML documents, especially documents that record the history of an enterprise. There is nothing special about the representation or modeling of these times. They would be modeled just the same as any other snippet of XML, e.g., represented within a <time> element. Temporal XML, on the other hand, is different. It models both the components within a document or data collection and their lifetimes. An instructive way to think about the difference is that temporal XML weds metadata in the form of timestamps to data contained in a document, i.e., to the elements or parts of the document that are annotated by the timestamps. Research in temporal XML builds on earlier research in temporal (relational) databases. Though many of the concepts and ideas carry over to temporal XML research, the ideas have to be adapted to the tree-like model of XML.

Many temporal XML data models impose a transaction-time constraint on the times along every path in a model instance: the timestamp of a child must be during (inclusive) the timestamp of its parent [1,4]. Said differently, no child may outlive its parent in transaction time. The reason for this constraint is that every snapshot of a temporal data model instance

must be a single, complete, valid non-temporal XML data model instance. A non-temporal instance has a single root. But if in a temporal instance a child outlives its parent then, in some snapshot(s), the child represents a second root since it has no parent, thus violating a model property. In valid time it is more common to relax this constraint and model a temporal data collection as a sequence of forests where a child that outlives its parent is interpreted to mean that the child is the root in some snapshot(s) of some tree in the forest [10]. For instance, the valid time of Jane Austen's book Pride and Prejudice would extend from its time of publication (1813) to *now*, far exceeding the lifetime of its publication by Butterfly Books.

Another interesting situation is when a child moves among parents over time (for instance, in the data collection shown in Fig. 1 if the book Pride and Prejudice were published by two different publishers). A directed graph data model is better suited to modeling such movement as a node (e.g., the book) can have multiple incoming edges (e.g., an edge from each publisher) [5]. Various constraints have been proposed for relationships among the timestamps on nodes and edges in the graph.

Timestamps on nodes/edges in a data model instance changes query evaluation. At the core of all XML query languages (and different from SQL or relational query languages) are path expressions that navigate to nodes in a data model instance. In a temporal data model instance, a query has to account for the timestamps along each path that it explores. In general, a node is only available during the intersection of times on every node and edge in the path to it (though a node in a graph data model can be reached along multiple paths). Temporal XML queries can be evaluated using a sequenced semantics [7], that is, simultaneously evaluated in every snapshot or non-sequenced [6,14] where differences between versions can be extracted and paths between versions are directly supported by the data model.

Key Applications

Temporal XML can be used to model an evolving document or data collection. In many situations, "old" documents or document versions are still of use. For instance, in an industrial domain an airplane parts manufacturer has to retain part plan histories to produce parts for older planes, while in the legal domain a tax firm has to keep a complete history of tax laws for audits. Currently, the de facto method for storing old documents is an *archive*. An archive is a warehouse for deleted or modified documents. Archives can be site-specific or built for a number of sites, e.g., the Internet Archive. But the method to retrieve documents from an archive varies widely from site to site, which is problematic because then queries also have to vary. Moreover, archives typically only support retrievals of entire document versions, not a full range of temporal queries or version histories of individual elements. In contrast, temporal XML provides a basis for supporting a full range of temporal queries. Temporal XML can also be explicitly used to represent, store or view historical data, including structured data, or to encode multi-version documents. Multi-version documents are compact representations of XML documents which maintain their identity through modifications and amendments. A temporally consistent individual version or range of consecutive versions (timeslice) can be extracted by means of a temporal query. Temporal XML has also been proposed as a medium of communication with temporal relational databases in the context of traditional enterprise applications.

Future Directions

The future of temporal XML is tied to the continued growth of XML as an important medium for data storage and exchange. Currently, many sites promote XML by publishing data formatted in XML (e.g., genomic and proteomic data can be obtained in three, different XML formats from the National Center for Biotechnology Information (NCBI)). Building a temporal XML data collection by accumulating snapshots gathered from these sites is vital to answering queries such as "What new data has emerged over the past six months?" As search engines become more XML-aware, they could also benefit enormously from making time a relevant component in ranking resources, e.g., a search for "mp3 players" should lower the ranking of discontinued products. The growth of the Semantic Web may lead to XML being supplanted by new languages for knowledge representation such as the Ontology Web Language (OWL). Temporal extensions of these languages will not be far behind. OWL already has one such extension: the Time-determined Ontology Web Language (TOWL).

Cross-references

► Temporal Database
► Temporal Queries
► XML

Recommended Reading

1. Amagasa T., Yoshikawa M., and Uemura S. A data model for temporal XML documents. In Proc. 11th Int. Conf. Database and Expert Syst. Appl., 2000, pp. 334–344.
2. Chawathe S.S., Abiteboul S., and Widom J. Representing and querying changes in semistructured data. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 4–13.
3. Chien S.-Y., Tsotras V.J., and Zaniolo C. Efficient schemes for managing multiversion XML documents. VLDB J., 11(4):332–353, 2002s.
4. Currim F., Currim S., Dyreson C., and Snodgrass R.T. A tale of two schemas: creating a temporal XML schema from a snapshot schema with τ XSchema. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 348–365.
5. Dyreson C., Böhlen M.H., and Jensen C.S. Capturing and querying multiple aspects of semistructured data. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 290–301.
6. Dyreson C.E. Observing transaction-time semantics with TTXPath. In Proc. 2nd Int. Conf. on Web Information Systems Eng., 2001, pp. 193–202.
7. Gao D. and Snodgrass R.T. Temporal slicing in the evaluation of XML queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 632–643.
8. Grandi F. Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web. ACM SIGMOD Rec., 33(2):84–86, 2004.
9. Grandi F. and Mandreoli F. The valid web: an XML/XSL infrastructure for temporal management of web documents. In Proc. 1st Int. Conf. Advances in Information Systems, 2000, pp. 294–303.
10. Grandi F., Mandreoli F., and Tiberio P. Temporal modelling and management of normative documents in XML format. Data Knowl. Eng., 54(3):327–254, 2005.
11. Mitakos T., Gergatsoulis M., Stavrakas Y., and Ioannidis E.V. Representing time-dependent information in multi-dimensional XML. J. Comput. Inf. Technol., 9(3):233–238, 2001.
12. Rizzolo F. and Vaisman A.A. Temporal XML: modeling, indexing and query processing. VLDB J., 2007.
13. Wang F. and Zaniolo C. X-BiT: An XML-based Bitemporal Data Model. In Proc. 13th Int. Conf. on Entity-Relationship Approach, 2004, pp. 810–824.
14. Wang F. and Zaniolo C. An XML-based approach to publishing and querying the history of databases. World Wide Web, 8(3):233–259, 2005.
15. Wong R.K., Lam F., and Orgun M.A. Modelling and manipulating multidimensional data in semistructured databases. World Wide Web, 4(1–2):79–99, 2001.

Temporally Indeterminate Databases

► Probabilistic Temporal Databases

Temporally Uncertain Databases

► Probabilistic Temporal Databases

Temporally Weak

► Snapshot Equivalence
► Weak Equivalence

Term Expansion

► Query Expansion for Information Retrieval

Term Expansion Models

► Query Expansion Models

Term Frequency by Inverse Document Frequency

► TF*IDF

Term Frequency Normalization

► Document Length Normalization

Term Processing

► Lexical Analysis of Textual Data

Term Proximity

VASSILIS PLACHOURAS
Yahoo! Research Barcelona, Spain

Synonyms

Lexical affinities; Lexical relations

Definition

Term proximity is a form of term dependence based on the distance of terms in a document. A retrieval system using term proximity assigns a higher score to documents in which the query terms appear close to each other.

Key Points

Term proximity is a feature that partially captures the dependence of terms in documents. Information retrievals models are often based on the assumption that terms occur independently of other terms in a document. This assumption is only an approximation to allow the simple mathematical development of retrieval models. There have been, however, several efforts to introduce dependence of terms [4]. Most of the efforts to use term proximity in the past did not result in substantial improvements. Metzler and Croft [2] argued that this can be attributed to the small size of the test collections used in the past, as well as to the fact that previous models required estimating term dependencies for both the classes of relevant and non-relevant documents.

Metzler and Croft [2] proposed a model based on Markov Random Fields for term dependence using term proximity. They modeled full independence, sequential dependence that is equivalent to phrase search, and full dependence, where the dependence between any pair of query terms is computed. Mishne and de Rijke [3] also proposed a model in which every n-gram of the query is considered as a phrase, and it is evaluated on an index consisting of single terms. Their results show that improvements in early precision are obtained in the setting of Web search. In both models, term proximity is based on lexical relations [1]. Terms are said to be in a lexical relation if they appear often within a certain number of tokens of each other.

Cross-references

▶ Information Retrieval Models
▶ N-Gram Models

Recommended Reading

1. Maarek Y.S. and Smadja F.Z. Full text indexing based on lexical relations an application: software libraries. In Proc. 12th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1989, pp. 198–206.
2. Metzler D. and Croft B. A Markov random field model for term dependencies. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 472–479.
3. Mishne G. and de Rijke M. Boosting Web retrieval through query operations. In Proc. 27th European Conf. on IR Research, 2005, pp. 502–516.
4. Yu C.T., Buckley C., Lam K., and Salton G. A generalized term dependence model in information retrieval. Inform. Technol. R&D, 2:129–154, 1983.

Term Statistics for Structured Text Retrieval

MOUNIA LALMAS
Queen Mary, University of London, London, UK

Synonyms

Within-element term frequency; Inverse element frequency

Definition

Classical ranking algorithms in information retrieval make use of term statistics, the most common (and basic) ones being within-document term frequency, *tf*, and document frequency, *df*. *tf* is the number of occurrences of a term in a document and is used to reflect how well a term captures the topic of a document, whereas *df* is the number of documents in which a term appears and is used to reflect how well a term discriminates between relevant and non-relevant documents. *df* is also commonly referred to as inverse document frequency, *idf*, since it is inversely related to the importance of a term. Both *tf* and *idf* are obtained at indexing time. Ranking algorithms for structured text retrieval, and more precisely XML retrieval, require similar terms statistics, but with respect to elements.

Key Points

To calculate term statistics for elements, one could simply replace documents by elements and calculate so-called within-element term frequency, *etf*, and

inverse element frequency, *ief*. This however raises an issue because of the nested nature of XML documents in particular. For instance, suppose that a section element is composed of two paragraph elements. The fact that a term appears in the paragraph necessitates that it also appears in the section. This overlap can be taken into account when calculating the *ief* value of a term.

In structured retrieval, in contrast to "flat" document retrieval, there are no a priori fixed retrieval units. The whole document, a part of it (e.g., one of its section), or a part of a part (e.g., a paragraph in the section), all constitute potential answers to queries. The simplest approach to allow the retrieval of elements at any level of granularity is to index all elements. Each element thus corresponds to a document, and *etf* and *ief* for each element are calculated based on the concatenation of the text of the element and that of its descendants (e.g., [4]).

With respect to the calculation of the inverse element frequency, *ief*, the above approach ignores the issue of nested elements. Indeed, the *ief* value of a term will consider both the element that contains that term and all elements that do so in virtue of being ancestors of that element. Alternatively, *ief* can be estimated across elements of the same type (e.g., [3]) or across documents (e.g., [1]). The former greatly reduces the impact of nested elements on the *ief* value of a term, but does not eliminate it as elements of the same type can be nested within each other. This approach can be extended to consider the actual path of an element, leading to so-called inverted path frequency. For example, in [2], this is defined as the combination of the *ief* values (as above calculated) with respect to each of the element types forming the path. The latter case, i.e., calculating *ief* across documents, is the same as using inverse document frequency, which completely eliminates the effect of nested elements.

Cross-references
▶ XML Retrieval
▶ Indexing Units
▶ Structure Weight
▶ Relationships in Structured Text Retrieval

Recommended Reading

1. Clarke C.L.A. Controlling overlap in content-oriented XML retrieval. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 441–448.
2. Grabs G. and Schek H.-S. ETH Zürich at INEX: flexible information retrieval from XML with PowerDB-XML. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002, pp. 141–148.
3. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 73–84.
4. Sigurbjörnsson B., Kamps J., and de Rijke M. An element-based approach to XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 19–26.

Term Weighting

Ibrahim Abu El-Khair
Minia University, Minia, Egypt

Definition
Term weighting is a procedure that takes place during the text indexing process in order to assess the value of each term to the document. Term weighting is the assignment of numerical values to terms that represent their importance in a document in order to improve retrieval effectiveness [8]. Essentially it considers the relative importance of individual words in an information retrieval system, which can improve system effectiveness, since not all the terms in a given document collection are of equal importance. Weighing the terms is the means that enables the retrieval system to determine the importance of a given term in a certain document or a query. It is a crucial component of any information retrieval system, a component that has shown great potential for improving the retrieval effectiveness of an information retrieval system [7].

Historical Background
The use of word frequency dates back to G. K. Zipf and his well known law [14] for word distribution. The law indicates that there is a correlation between the frequency of a word and its rank, and their product is a constant.

$$r^*f = c$$

where: r is the rank of the word
f is the frequency of the word
c is a parameter/constant that depends on the text being analyzed.

Zipf indicates that this law is a way to express the within-document frequency weighting.

The use of word frequency as an indication of its significance in a given document was established almost 10 years later based on observations made by Luhn [4] when he was conducting an explanatory research on creating automatic abstracts in scientific documents. Based on these observations, Luhn proposed that the frequency of word occurrence in a document can be considered a useful measure of the word's significance in that document. The premise is that the author of a certain document repeats certain words as he/she presents his/her argument elaborating the subject of the document.

Looking at the word distribution in any given document shows that there are significant words and insignificant words. Luhn used Zipf's law as his null hypothesis [14] to enable him to specify two cut-off points to exclude all insignificant words, an upper point and a lower point. The upper cut-off eliminated the common words, and the lower point eliminated the rare words; both of which he considered extraneous in the document content. After Luhn excluded words above and below these two cut off points, the most useful range of words remained.

The 1960's saw several key developments in the field of information retrieval in general and the most notable were related to the development of the SMART system by Gerard Salton and his students, first at Harvard University and later at Cornell University. This system utilized weights for index terms based on their frequency [10]. The probabilistic approach to retrieval, with term weights based on probability of relevance, appeared in 1960 and since then it has been tested heavily with many variations [12].

The decade of the 1970's saw a breakthrough in the calculation of term weights used in retrieval systems. By then, it was confirmed that the significance of a certain term in a given document is determined using the term frequency of that term in the document. Sparck Jones [11] argued that the term frequency by itself is not sufficient enough to measure the importance of a term in a collection of documents. She suggested correlating the term frequency with its relative collection frequency, making the collection frequency a variable in retrieval. A significant development in the probabilistic based retrieval was also achieved in 1976 by Robertson and Sparck Jones [6].

Foundations

Storing, organizing, and retrieving information are the main functions of an information retrieval system. With the vast amounts of electronic information now available it is very hard to find the ideal information retrieval system that enables users to get what they want from a specific collection of documents. A considerable amount of research has addressed improvement in the effectiveness of retrieval systems, most of which is focused on finding appropriate indexing techniques. The indexing process in any retrieval system deals with assigning a set of index terms that represents the content of each document within a collection. Choosing the proper index terms is a primary issue in information retrieval systems, as they should be indicative of the content of a given document.

One of the most important procedures in the indexing process is assigning a value, or weight, to an index term in a document. It is a crucial component of any retrieval system, and one that has shown great potential for improving retrieval effectiveness [7]. By assigning a numerical value to a term representing its importance in the document, retrieval effectiveness can be improved [8]. Term weighting indicates how important each individual word is to the document and within the document collection.

The process of assigning term weights is affected by three major factors: term frequency, inverse document frequency, and document length.

Term Frequency

Following Luhn's observations [4], it is known that the significance of a certain term in a given document can be represented by the term frequency of that term in the document. Simply, if there is a document in which the word "database" occurs a hundred times, that document would potentially be more useful in response to a query containing "database" as a query term than a document in which the word appears only one or two times. Of course, the use of this factor alone in calculating the term weights in a collection of documents does not guarantee adequate retrieval performance. For example, there are very common words, sometimes referred to as stop words, which appear in the text but carry little meaning, serving only a syntactic function but not indicating subject matter [3]. They have a very high frequency and tend to diminish the impact of frequency differences among less common words, affecting the weighting process [2].

Inverse Document Frequency

The term frequency indicates the importance of the term in a given document, but knowing the term importance in a collection of documents is also significant. Term frequency was criticized as a method of determining term significance because in its simplest form, it treats all terms equally based on raw count, which does not take into account the term's discriminating power. To resolve this problem Sparck Jones [11] suggested the use of the relative collection frequency or inverse document frequency (IDF), making the frequency of the term in the collection as a whole a variable in retrieval. IDF places greater emphasis on the value of a term as a means of distinguishing one document from another than on its value as an indication of the content of the document itself.

Document Length

With the presence of long documents in the document collection handled by any retrieval system, it became harder to determine the importance of a term based only on the term frequency or the inverse document frequency or both. Even though the combination of them is a good weighting function, it overlooks the document length factor. Longer documents will have higher term frequencies because the terms tend to be repeated several times in the document, and thus will be high in the ranking during the retrieval process. Long documents are also likely to contain more unique terms which may affect the retrieval as well. More terms in a given document increases the possibility of matching between this document and multiple queries [7]. Applying a good normalization technique reduces the effect of long documents and makes the weighting function more effective. Another element to be taken into consideration with the document length factor is the removal of stop words, which changes the document length and subsequently affects the weighting process [2].

Term Weighting Schemes

The following is a brief and basic explanation of some of the major term weighting schemes available. It should be noted that each scheme has many variations and modifications that are not discussed.

TF*IDF A weighting function that depends on the term frequency (TF) in a given document calculated with its relative collection frequency or inverse document frequency (IDF). The term frequency emphasizes term significance in a given document, and inverse document frequency emphasizes term significance in the collection as a whole (TF*IDF).

BM25 A weighting function based on the traditional *Probabilistic Retrieval Model.* The basic principle is that a specific document could be judged relevant to a specific query, based on the assumption that the terms are distributed differently and independently in relevant and non relevant documents. The weight of a given term is calculated on the basis of the presence or absence of query terms in each document in the collection. Terms that have appeared in previously retrieved relevant documents for a given query should be given a higher weight than if they had not appeared in those relevant documents [12].

Language Modeling Language modeling (LM) is an extension of the probabilistic retrieval approach. It is a probabilistic mechanism for generating text, first applied by Andrei Markov at the beginning of the twentieth century to model letter sequences in works of Russian literature. It was also used by Claude Shannon in his models of letter sequences and word sequences, which he used to illustrate the implications of coding and information theory. At the end of the 1970's, LM was used successfully in speech recognition, which was its main application for many years [1]. In 1998 Ponte and Croft [5] were the first to apply language modeling to information retrieval. Their approach was to infer a language model for each document and estimate the probability of generating the query according to each of these models, and then rank the documents according to these probabilities. The results indicated an improvement in retrieval over the traditional TF*IDF, and there was further improvement when they used a smoothing function with their new approach.

Key Applications

Term weighting is a key process in any information retrieval system. It is the means that enables the system to determine the importance of any term in a certain document or a query.

Experimental Results

Experimentation in information retrieval has been an active area for over 40 years, and much of this research

has focused on term weighting. Different schemes and variations with different retrieval models have been tested in order to find weighting schemes that perform effectively. In general, the schemes above and their variations have been tested extensively and evaluated (see the corresponding references). Many other weighting schemes were developed and used but without becoming widely adopted, either because the results were not effective enough or because of the complexity of the calculations or both. The Term Discrimination Value (TDV) model of indexing [9] is an example which is now seldom used because of its complexity and weak results.

Until the 1990s, experiments in the field of information retrieval in general and term weighting in particular were conducted on relatively small collections. With the beginning of TREC (http://trec.nist.gov) (Text REtrieval Conference) in 1992, large test collections became available for use by the IR community, making the results of experiments more credible and generalizable. Research in term weighting benefited, establishing the effectiveness of term weighting schemes such as BM25.

Cross-references
► BM25
► Information Retrieval
► Language Models
► Lexical Analysis of Textual Data
► TF*IDF
► Text Indexing Techniques

Recommended Reading

1. Hiemstra D. and de Vries A. Relating the New Language Models of Information Retrieval to the Traditional Retrieval Models (No. TR-CTIT-00-09). Centre for Telematics and Information Technology (CTIT), University of Twente, Amsterdam, Netherlands, 2000.
2. Korfhage R.R. Information Storage and Retrieval. John Wiley, New York, 1997.
3. Lancaster F.W. Indexing and Abstracting in Theory and Practice (2nd edn.). University of Illinois, Graduate School of Library and Information Science, Champaign, IL, 1998.
4. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Dev., 2(2):159–165, 1958.
5. Ponte J.M. and Croft W.B. A language modeling approach to information retrieval. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 275–281.
6. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inf. Sci., 27(3):129–146, 1976.
7. Salton G. and Buckley C. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage., 24(4):513–523, 1988.
8. Salton G. and McGill M. Introduction to Modern Information Retrieval. McGraw-Hill Book Company, New York, NY, 1983.
9. Salton G., Yang, C.S., and Yu, C.T. A theory of term importance in automatic text analysis. J. Am. Soc. Inf. Sci. Technol., 26(1):33–44, 1975.
10. Singhal A. Modern information retrieval: a brief overview. Bull. IEEE Comput. Soc. Tech. Comm. Data Eng., 24(4):35–43, 2001.
11. Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval. J. Doc., 28:11–20, 1972.
12. Sparck Jones K., Walker S., and Robertson S.E. A probabilistic model of information retrieval: development and comparative experiments: Part I. Inf. Process. Manage., 36:779–808, 2000.
13. van Rijsbergen C.J. Information Retrieval (2nd edn.). Butterworths, London, 1979.
14. Zipf G.K. Human Behavior and Principle of Least Effort. Addison Wesley, Cambridge, MA, 1949.

Term-Document Matching Function

► Information Retrieval Models

Terminologic Languages

► Description Logics

Terminological Database

► Electronic Dictionary

Test Collection

BEN CARTERETTE
University of Massachusetts Amherst, Amherst, USA

Synonyms
Corpus

Definition
A test collection is a standard set of data used to measure search engine performance. It comprises a set of queries, ideally randomly sampled from some space, a set of documents to be searched, and a set of judgments indicating the relevance of each document to each query in the set.

Key Points

The use of test collections for performance evaluation began with Cleverdon and Mills [1] and is today known as the Cranfield methodology. Test collections today are much larger than Cleverdon's Cranfield collection, consisting of millions of documents and tens of thousands of relevance judgments. The advantage of having standardized test collections is that experimental results can be compared across research groups and over time.

The National Institute of Standards and Technology (NIST), through their annual Text REtrieval Conferences (TREC), has led the way in providing test collections for information retrieval research. NIST has assembled large-scale test collections for many different retrieval tasks and types of documents and made these available to researchers so that the state of the art in retrieval can be constantly improved.

Cross-references

▶ Document Databases

Recommended Reading

1. Voorhees E.M. and Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.

Text Analytics

▶ Web Information Extraction

Text Categorization

Dou Shen
Microsoft Corporation, Redmond, WA, USA

Synonyms

Text classification

Definition

Text classification is to automatically assign textual documents (such as documents in plain text and Web pages) into some predefined categories based their content. Formally speaking, text classification works on an instance space X where each instance is a document d and a fixed set of classes $C = \{C_1, C_2, ..., C_{|C|}\}$ where $|C|$ is the number of classes. Given a training set D_l of training documents $\langle d, C_i \rangle$ where $\langle d, C_i \rangle \in X \times C$, using a learning method or learning algorithm, the goal of document classification is to learn a classifier or classification function γ that maps instances to classes: $\gamma : X \rightarrow C$ [7].

Historical Background

Text classification, which is to classify documents into some predefined categories, provides an effective way to organize documents. Text classification dates back to the early 1960s, but only in the early 1990s did it become a major subfield of the information systems discipline. Recently, with the explosive growth of on-line textual data, text classification attracts more and more attention. A knowledge engineering approach is one of the most popular solutions before the late 1980s, which relies on manually defined rules encoding expert knowledge on how to classify documents under the given categories. After that, machine learning based methods became pervasive. These methods automatically build some automatic text classifiers by learning from a set of manually classified documents. The following sections will focus on machine learning based methods.

Foundations

Text categorization consists of several important components including document representation, dimensionality reduction, classification algorithms and performance evaluation, which are to be introduced in the followings sections. Readers are referred to [11] for more details.

Document Representation

Documents or Web pages cannot be directly interpreted by a classifier. Therefore, a proper representation approach is necessary to represent documents. Generally, a document d is usually represented as a vector of term weights $d = \langle w_1, w_2, ..., w_{|V|} \rangle$, where V is the set of terms (sometimes called features) that occur at least once in the training document set D_l. This way is known as Vector Space Model (VSM) [7]. Different representation approaches vary in two issues: (i) different ways of understanding what a term is; (ii) different ways of computing term weights. For issue (i), a straightforward way is to identify terms with words. This is often called either the set-of-words or the bag-of-words approach to document

representation, depending on whether weights are binary or not [11]. Although some previous work has found that representations more sophisticated than this are not significantly more effective [1], researchers still struggle to find better ways in the following four directions: (i) Represent a document by phrases [6]; (ii) Use the senses of words to represent a document [4]; (ii) Augment document representation by hidden concepts in a document; (iv) Employ language models, such as n-gram models [9], multigram models. For issue (ii), the weight can be binary (1 denoting presence and 0 absence of the term in the document) or nonbinary. For the nonbinary value of a term t, it can be either the Term Frequency (TF) of the term in a document or TFDIF as computed according to the following equation where $N(t,d)$ is the number of times the word t appears in d, $|D|$ is the size of the corpus, $n_{t,D}$ is the number of documents in D containing the word t:

$$w_t = N(t, d) * log(|D|/n_{t,D})$$

Sometimes, the document vectors are normalized by cosine normalization [11]. Using either binary or nonbinary values, either normalized or the original values depends on the classifier learning algorithm.

Generally, the bag-of-words representation is built based on the text in each document alone. However, it is not uncommon that some documents have certain relationships among them so that the text in one document can help enrich the text of its related documents to improve the classification results. The relationships among documents are quite obvious in the context of Web-page classification. For example, in [2], Glover et al. enrich Web pages by considering inbound links and words surrounding them. They come to the conclusion that the full-text of a Web page is not good enough for representing the Web pages for classification. They create virtual documents by incorporating anchor text and extended anchor text. The experimental results demonstrate that the virtual documents, especially when constructed through extended anchor text are of great help. In [13], the authors enhance the notion of virtual documents by complementing hyperlinks with implicit links which are extracted from query logs. Besides utilizing the links among documents, there are also works enriching documents by inserting features extracted from an existing knowledge base.

Dimensionality Reduction

In document classification, it is unavoidable to face the problem of high dimensionality. The original feature space consisting of the unique terms that occur in a moderate-sized document collection can reach hundreds of thousands of terms. Such high dimensionality prohibits the application of many classification algorithms such as neural networks and Bayes belief models [15]. Furthermore, some features in the native space do not contribute much to the document classification. Therefore, it is beneficial to conduct dimensionality reduction (DR). DR techniques consist of two groups, term selection and term generation. Term selection methods select a subset of terms from the native space while term generation methods obtain new features by combining or transforming the original ones. The methods belonging to the former group include Document Frequency (DF), Mutual Information (MI), Chi-Square and so on. The latter group of methods include Term Clustering, and Latent Semantic Indexing (LSI). A detailed analysis and comparison of these methods is presented in [11,15].

Related to feature selection, some works have tried to remove noise from documents. It is easy to imagine that some text in a document, especially in a Web page, is not related to the main topic of the documents. When judging the category of a document, only the main topic of the document should be considered and the irrelevant text should be removed. These works are different from conventional feature selection in that they process each document independently and the resultant feature space can still have high dimensionality. For example, in [5], Kolcz et al. use summarization as a feature selection method and apply a simple extraction-based technique with several heuristic rules. Different from Kolcz et al's work on pure-text document classification, [12] proposes to improve the Web-page classification performance by removing the noise through some summarization techniques.

Classification Algorithms

During the past few decades, a large number of categorization algorithms have been proposed for document classification such as naïve bayes [8], k-nearest neighbor, decision trees, regression models, neural networks, support vector machines [3], boosting and rule learning algorithms. The authors of [14] made a thorough comparison among these classifiers. In this

section, two widely used text classification algorithms, Naive Bayes (NB) and Support Vector Machine (SVM) are briefly introduced.

Naïve Bayesian Classifier (NB) The Naïve Bayesian Classifier (NB) is a simple but effective text classification algorithm which has been shown to perform very well in practice [8]. The basic idea of NB is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. As described in [8], most researchers employ NB method by applying Bayes' rule:

$$P(C_j|d_i; \hat{\theta}) = \frac{P(C_j|\hat{\theta}) \prod_{k=1}^{|V|} P(w_k|C_j; \hat{\theta})^{N(w_k, d_i)}}{\sum_{r=1}^{|C|} P(C_r|\hat{\theta}) \prod_{k=1}^{|V|} P(w_k|C_r; \hat{\theta})^{N(w_k, d_i)}}$$

where $P(C_j|\hat{\theta})$ can be calculated by counting the frequency with each category C_j occurring in the training data; $|C|$ is the number of categories; $p(w_i|C_j)$ stands for probability that word w_i occurs in class C_j which may be small in training data, so the Laplace smoothing is chosen to estimate it; $N(w_k, d_i)$ is the number of occurrences of a word w_k in d_i; $|V|$ is the number of words in the training data.

Support Vector Machine (SVM) SVM is well founded in terms of computational learning theory and has been successfully applied to text categorization [3]. SVM operates by finding a hyper-surface in the space of possible inputs. The hyper-surface attempts to split the positive examples from the negative examples by maximizing the distance between the nearest of the positive and negative examples to the hyper-surface. Intuitively, this makes the classification correct for testing data that is near but not identical to the training data. There are various ways to train SVMs. The SVM^{light} system provided by Joachims [3] is one of the widely adopted and efficient implementations.

Performance Measures
Precision, recall and F1-measure are the most popular measures to evaluate the performance of document classification [10]. Precision (P) is the proportion of actual positive class members returned by the system among all predicted positive class members returned by the system. Recall (R) is the proportion of predicted positive members among all actual positive class members in the data. F1 is the harmonic average of precision and recall as shown below:

$$F1 = 2 \times P \times R / (P + R)$$

To evaluate the average performance across multiple categories, there are two conventional methods: micro-average and macro-average. Micro-average gives equal weight to every document; while macro-average gives equal weight to every category, regardless of its frequency [14].

Key Applications
Text classification has many applications [9], including email classification, text genre classification, topic identification, subjective sentiment classification and Web query classification.

Data Sets
There are several open data sets for text categorization. See the following for details:

20 Newsgroups: http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html

Reuters-21578: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

RCV1: http://www.daviddlewis.com/resources/testcollections/rcv1

Enron Email Dataset: http://www.cs.cmu.edu/ enron/

Query Classification Dataset: http://www.sigkdd.org/kddcup/index.php?section=2005&method=data

Cross-references
► Classification
► Information Retrieval (IR)
► Text Clustering

Recommended Reading
1. Dumais S., Platt J., Heckerman D., and Sahami M. Inductive learning algorithms and representations for text categorization. In Proc. Int. Conf. on Information and Knowledge Mangement, 1998, pp. 148–155.
2. Glover E.J., Tsioutsiouliklis K., Lawrence S., Pennock D.M., and Flake G.W. Using web structure for classifying and describing web pages. In Proc. 11th Int. Conf. World Wide Web Conference. 2002, pp. 562–569.
3. Joachims T. Text categorization with support vector machines: learning with many relevant features. In Proc. 10th European Conf. on Machine Learning, 1998, pp. 137–142.
4. Kehagias A., Petridis V., Kaburlasos V.G., and Fragkou P. A comparison of word- and sense-based text categorization using several classification algorithms. J. Intell. Inf. Syst., 21(3):227–247, 2003.
5. Kolcz A., Prabakarmurthi V., and Kalita J.K. String match and text extraction: summarization as feature selection for

text categorization, In CIKM'01: Proc. 10th ACM Int. Conf. on Information and Knowledge Management, 2001, pp. 365–370.

6. Lewis D.D. Representation quality in text classification: An introduction and experiment. In Proc. Workshop on Speech and Natural Language, 1990, pp. 288–295.

7. Manning C.D., Raghavan P., and SchÜZe H. Introduction To Information Retrieval. Cambridge University Press, 2007.

8. Mccallum A. and Nigam K. A comparison of event models for naive bayes text classication. In Proc. AAAI-98 Workshop on Learning for Text Categorization, 1998.

9. Peng F., Schuurmans D., and Wang S. Augmenting naive bayes classifiers with statistical language models. Inf. Retr., 7(3–4):317–345, 2004.

10. Rijsbergen C.V. Information Retrieval, 2nd edn. Butterworths, London, 1979.

11. Sebastiani F. Machine learning in automated text categorization ACM Comput. Surv., 34(1):1–47, 2002.

12. Shen D., Chen Z., Yang Q., Zeng H.-J., Zhang B., Lu Y., and Ma W.-Y. Web-page classification through summarization. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 242–249.

13. Shen D., Sun J.-T., Yang Q., and Chen Z. A comparison of implicit and explicit links for web page classification. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 643–650.

14. Yang Y. An evaluation of statistical approaches to text categorization. Inf. Retr., 1(1–2):69–90,1999.

15. Yang Y. and Pedersen J.O. A comparative study on feature selection in text categorization. In Proc. 14th Int. Conf. on Machine Learning, 1997, pp. 412–420.

Text Classification

▶ Text Categorization

Text Clustering

Hua Li
Microsoft Research Asia, Beijing, China

Definition

Text clustering is to automatically group textual documents (for example, documents in plain text, web pages, emails and etc) into clusters based on their content similarity. The problem of text clustering can be defined as follows. Given a set of n documents noted as DS and a pre-defined cluster number K (usually set by users), DS is clustered into K document clusters $DS_1, DS_2,...,DS_k$, ($i.e,\{DS_1, DS_2,...,DS_k\} = DS$) so that the documents in a same document cluster are similar to one another while documents from different clusters are dissimilar [14].

Historical Background

Text clustering was initially developed to improve the performance of search engines through pre-clustering the entire corpus [2]. Text clustering later has also been investigated as a post-retrieval document browsing technique [1,2,7].

Foundations

Text clustering consists of several important components including document representation, text clustering algorithms and performance measurements. The readers should refer to [6,8,13] for more details.

Document Representation

The original representation of textual documents (like plain texts, web pages, emails and etc) could not be interpreted by text clustering algorithms directly. A proper document representation method is necessary for any text clustering algorithms. Vector Space Model [6] is generally used to represent a document d as a vector of term weights $d =< w_1, w_2,...,w_{|V|} >$, where V is the set of terms (also named as features sometimes) that occur at least once in the document set DS. Different representation approaches vary in two issues: (i) different ways of understanding what a term is; (ii) different ways of computing term weights. For issue (i), a straightforward way is to identify terms with words. This is often called either the set-of-words or the bag-of-words approach to document representation, depending on whether weights are binary or not [11]. Some previous work has found that representations more sophisticated than this are not significantly more effective [5]. For issue (ii), the weight can be binary (1 denoting the presence and 0 absence of the term in the document) or non-binary. For the non-binary value, it can be either the Term Frequency (TF) of the term in a document or $TFIDF$ as computed according to the following equation where $N(t, d)$ is the number of the times the word t appears in d, $|D|$ is the size of the document corpus, $n_{t,D}$ is the number of documents in D containing the term t:

$$w_t = N(t, d) * \log(|D|/n_{t,D})$$

Cosine normalization is sometimes used to normalize the document vectors [11]. It would depend on the

text clustering algorithms to choose proper term weight strategies.

Text Clustering Algorithms

Two categories can be used to organize all various clustering algorithms (most of the general clustering algorithms could be applied to text clustering tasks) developed in the past a few years: hierarchical and parititional approaches. The hierarchical algorithms generate successive clusters in a nested sequence. The partitional ones produce all clusters at one time.

In the following section, three popular clustering algorithms would be briefly introduced for readers to get primary impressions of basic clustering algorithms. Single-Link clustering [3] is one basic approach among hierarchical clustering algorithms category (http://en.wikipedia.org/wiki/Cluster_analysis). K-Means clustering [9] is one of the typical paritional algorithms which minimizes square error to generate clusters. Co-clustering [4] is a graphic theory based partitional clustering approach which is very popular in recent years. For more clustering algorithms, the readers can refer to [6].

Single-Link Clustering In the Single-Link clustering, the distance between two clusters is defined as the minimum of the distances of all linkages drawn from the two clusters, where the linkage is the criterion to determine the distance of pairs of patterns/points between two clusters while patterns/points are associated with them. One shortcoming of the Single-Link clustering is that it would suffer from a chaining effect [10] which has a tendency to produce clusters that are straggly or elongated [6].

The three main steps of Single-Link Clustering algorithm are as follows [6]:

1. With each pattern/point in its own cluster, construct a list of inter-pattern/point distances for all distinct N ordered pairs of patterns/points, and sort this list in ascending order.
2. Step through the sorted list of distances, forming for each distinct dissimilarity value d_k a graph on the patterns where pairs of patterns closer than d_k are connected by a graph edge.
   a If all the patterns are members of a connected graph, stop.
   b Otherwise, repeat 2.

3. The output of the algorithm is a nested hierarchy of graphs which can be cut at a desired dissimilarity level to form a clustering. The clusters would be identified by simply connected components in the corresponding graph.

K-Means Clustering K-Means clustering algorithm is one of the simple but very efficient clustering algorithms, which allows it to run through large datasets. The main advantages of K-Means are (i) simplicity and efficiency; (ii) does not yield the same result with different run as the resulting clusters depend on the initial random assignments. The main disadvantage is that as it minimizes intra-cluster variance, K-means does not ensure the result has a global minimum of variance (http://en.wikipedia.org/wiki/Cluster_analysis).

K-Means algorithm is to cluster n objects (here textual documents) based on attributes (the document representation as vector space model) into K ($K < n$) partitions. It assigns each object to the cluster which has the nearest center. The center is defined as the average of all the objects in the cluster, which starts from a set of random initial centers. It assumes that the object attributes form a vector space and the objective for the algorithm to achieve is to minimize total intra-cluster variance or, the squared error function (http://en.wikipedia.org/wiki/Cluster_analysis):

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

where S_i, $i = 1, 2, \dots, k$ are K clusters and μ_i is the center of cluster S_i.

The main steps of K-Means clustering algorithm are as follows [9]:

1. Setup the cluster number K;
2. Randomly generate K clusters and calculate the cluster centers, or directly generate K random points as cluster centers;
3. Assign each other points to the nearest cluster center;
4. Recalculate the new cluster centers after new points are clustered into the clusters;
5. Repeat 3 and 4 until some convergence criterion is met;

Co-Clustering In Co-Clustering method, the document collection would be modeled as a bipartite graph between document and words. That makes the

clustering problem could be posed as a graph partitioning problem. Then Co-Clustering is developed as a spectral algorithm which could simultaneously yield a clustering of documents and words based on this document and word graph. The Co-Clustering algorithm uses the second left and right singular vectors of an appropriately scaled word-document matrix to yield good bipartitionings [4].

Performance Measurements

There are generally two types of measurements used to evaluate the performance of different text clustering algorithms. One is internal quality measure and the other is external quality measure. The authors of [12] had made a thorough introduction of various clustering algorithms measurements. The readers could refer to their work for more details. Here a brief introduction for both internal and external quality measurements would be introduced in the following.

Internal Quality Measure The internal quality measure is used to compare different sets of clusters without referring to external knowledge (like human labeled/known classes/categories). One approach of this kind of internal quality measurement is to calculate the "overall similarity" based on the pair-wise similarity of documents in a cluster [12].

External Quality Measure The external quality measure as naming is to leverage external knowledge as known classes (categories) to make comparisons with the generated clusters from the clustering algorithms. Entropy [12] is one external measure which provides a measure of "goodness" for un-nested clusters or for the clusters at one level of a hierarchical clustering. F-measure is another good example of external quality measure, which is more oriented toward measuring the effectiveness of a hierarchical clustering.

The readers should be aware that there are still many other different quality measures than those ones introduced here. The more important thing is that the performance of different clustering algorithms could vary substantially depending on which measure is applied [12].

Key Applications

Text clustering has many applications, including search results clustering, topic detection and tracking, email clustering, and etc.

Cross-references

► Document Clustering
► Information Retrieval
► Text Classification

Recommended Reading

1. Croft W.B. Organizing and Searching Large Files of Documents. Ph.D. Thesis, University of Cambridge, 1978.
2. Cutting D.R., Karger D.R., Pedersen J.O., and Tukey J.W. Scatter/gather: a cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.
3. Day W.H. and Edelsbrunner H. Efficient algorithms for agglomerative hierarchical clustering methods. J. Classification, 1:1–24, 1984.
4. Dhillon I.S. Co-clustering documents and words using bipartite spectral graph partitioning, UT CS Technical Report #TR. Department of Computer Sciences, University of Texas, Austin, TX, 2001.
5. Dumais S., Platt J., Heckerman D. and Sahami M. Inductive learning algorithms and representations for text categorization. In Proc. 7th Int. Conf. on Information and Knowledge Management, 1998, pp. 148–155.
6. Jain A.K., Murty M.N., and Flynn P.J. Data clustering: a review. ACM Comput. Surv., 31(3):264–323, 1999.
7. Leouski A.V., and Croft W.B. An evaluation of techniques for clustering search results. Technical Report IR-76. Department of Computer Science, University of Massachusetts, Amherst, 1996.
8. Lewis D.D. Representation quality in text classification: an introduction and experiment. In Proc. Workshop on Speech and Natural Language, 1990, pp. 288–295.
9. MacQueen J.B. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability, 1967, pp. 281–297.
10. NAGY G. State of the art in pattern recognition. Proc. IEEE., 56:836–862, 1968.
11. Sebastiani F. Machine learning in automated text categorization. ACM Comput Surv., 34(1):147, 2002.
12. Steinbach M., Karypis G., and Kumar V. A comparison of document clustering techniques. Technique Report, University of Minnesota – Computer Science and Engineering, 2000.
13. van Rijsbergen C.J. Information Retrieval, 2nd edn. Butterworths, London, 1979.
14. Yoo I. and Hu X.H. A comprehensive comparison study of document clustering for a biomedical distal library Medline. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2006, pp. 220–229.

Text Compression

Paolo Ferragina, Igor Nitto
University of Pisa, Pisa, Italy

Synonyms
Lossless data compression

Definition

Text Compression involves changing the representation of a file so that the (binary) compressed output takes less space to store, or less time to transmit, but still the original file can be reconstructed *exactly* from its compressed representation.

Key Points

The benefit of compressing texts in computer applications is threefold: it reduces the amount of memory to store a text, it reduces the time for transmitting the text over a computer network and, recently, it has been deployed to speed up algorithmic computations because they can better exploit the memory hierarchy available in modern PCs by reducing the disk access time, by increasing virtually the bandwidth and size of disk (or memory, cache), and by coming at a negligible cost because of the significant speed of current CPUs.

A text in uncompressed format, also called *raw* or *plain* text, is a sequence of symbols drawn from an alphabet Σ and represented in $\lceil \log_2 |\Sigma| \rceil$ bits each. Text compressors aim at storing a text in space less than its raw encoding by exploiting the redundancies possibly contained in it. Such redundancies might occur in the form of repetitions or frequently occurring patterns, which are not unlikely in texts generated by humans. Text compression is a *lossless* process because it allows restoring the original text from its compressed form by means of a proper *decompression* algorithm. Most of current approaches in text compression [4] can be classified into: *symbolwise*, *dictionary-based* and *transform-based*.

Symbolwise compressors encode the text one-symbol at time, by emitting a (variable length) codeword per individual symbol. They are further divided into two sub-families: *statistical* and *symbol-ranking*. Statistical compressors include some of the best compressors currently known, like PPM and DMC. They rely on two basic tools for the encoding task: a statistical model of the text and a statistical encoder. The statistical model serves to predict the probability of the next symbol given the previous portion of the text. If the prediction is conditioned on the last k occurred symbols, for some fixed constant k (typically ranging from 5 to 10), the model is said of *order k*. The model can be *static*, *semi-static*, or *dynamic*, according to the way the symbol probabilities are estimated from the scanning of the input text. The static construction assumes a fixed probability distribution for every input text. This is the simplest form of model construction but, since the model is independent of the input, it may poorly predict its symbols. The semi-static construction avoids this limitation by building the model via a preliminary scan of the input text. Unfortunately, a semi-static model must be passed to the decompressor as a part of the compressed file, thus increasing its size. As a result, statistical models are typically dynamic in that they are constructed *adaptively* as the input text is processed. This shares with the semi-static approach the advantage of being tailored on the input, but it additionally avoids the need of passing the model to the decompressor. The probability distribution provided by the model, whichever construction process is adopted, is eventually passed to the statistical encoder that determines the bit codeword associated to the next symbol. The principle used by all statistical encoding methods is to assign shorter codewords (even fraction of bits) to most probable symbols in order to minimize the average length of a codeword (this is typically called the *golden rule*). This is the essence of Huffman and Arithmetic encodings, also known as entropy encoders for their dependence on the entropy of symbols frequencies, a theoretical lower-bound on the number of bits emitted by any statistical encoder. The concatenation of all generated codewords gives the final compressed file of a statistical compressor.

Symbol-ranking compressors are still based on statistical prediction but, rather than estimating the probability distribution of the next symbol, they maintain the alphabet in a dynamic list, where symbols are sorted by decreasing likelihood of occurrence. Each symbol of the text is encoded with its rank in the list. If the prediction process is accurate enough, the distribution of ranks will be skewed around small values, and the resulting sequence will be well compressible through a statistical encoder. Symbol-ranking techniques have been rarely used as "stand-alone" compression methods but, like the well-known *Move-To-Front* (MTF) and *Inversion Frequency* encodings, they are often employed as a fundamental stage of more sophisticated compressors like the *Wheeler-Burrows* Transform (BWT), below.

While symbolwise compressors can encode only one symbol at a time, dictionary-based compressors can represent a *group* of symbols with one unique codeword. These compressors maintain a dictionary of strings, called *phrases*, each one identified by a distinct codeword. The input text is parsed into dictionary phrases which are then replaced by their

corresponding (shorter) codewords. As for statistical models, dictionary construction schemes can also be classified into static, semi-static and dynamic; but the most significant difference is that, in the semi-static case, deciding which phrases to include in the dictionary is computationally difficult: in fact, computing the dictionary that maximizes compression is an NP-hard problem. An example of static dictionary is the Run-Length-Encoding method (RLE), in which the phrases are all possible runs of equal symbols, typically adopted in FAX transmissions and as a fundamental stage of the BWT-based compressors. An example of semi-static dictionary is the Huffword compressor, in which the dictionary is formed by all tokens extracted from an input text and phrases are Huffman-encoded according to their frequency of occurrence. Examples of dynamic-dictionary methods are the well-known LZ77 and LZ78 compressors which are implemented in many commercial softwares like winzip, pkzip, ARJ, the .GIF image format, etc..

The last class of compression algorithms considered is the one that *transforms* the input text in order to make it easier to compress by simpler coding schemes. The transformation must be reversible, lossless (e.g., a permutation of the input symbols), and efficiently computable and invertible. One notable example is the *Burrows-Wheeler* transform (BWT), which can be built and inverted in time linear in the length of the input text. The output of the BWT is a string in which symbols following the same context are grouped together, giving raise to clusters of nearly identical symbols. This feature makes redundancy in the input more accessible to simple coding schemes. The famous compression utility bzip2, currently available on most Linux distributions, is indeed based on the BWT and uses a proper combination of MTF, RLE and a statistical encoder to significantly squeeze the BWT-output. Besides its usage in pure text compression, the BWT has three other remarkable properties: it can be used to design a *compression booster* [1], that is, a tool for improving the performance of other compressors in a well-defined and measurable way; it can be used to derive novel and powerful transform-based compressors for various other data types [2,3], like XML, dictionaries and graphs (just to cite a few); and it is at the core of modern Compressed Full-text Indexes [3], that is, compressed representations of the input string which are efficiently searchable via arbitrary patterns.

Cross-references

► Data Compression in Sensor Networks
► Indexing Compressed Text
► XML Compression

Recommended Reading

1. Ferragina P., Giancarlo R., Manzini G., Sciortino M. Boosting textual compression in optimal linear time. J. ACM, 52(4): 688–713, 2005.
2. Ferragina P., Luccio F., Manzini G., Muthukrishnan S. Compressing and searching XML data via two zips. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 751–760.
3. Navarro G., Mäkinen V. Compressed full-text indexes. ACM Comput. Surv., 39(1), Article no. 2, 2007.
4. Salomon D., Data Compression: The Complete Reference, 4th edn., Springer, London 2007.

Text Data Mining

► Data, Text, and Web Mining in Healthcare
► Text Mining

Text Databases

► Document Databases

Text Extraction

► Biomedical Scientific Textual Data Types and Processing

Text Generation

LI ZHANG[1], JIAN-TAO SUN[2]
[1]Peking University, Beijing, China
[2]Microsoft Research Asia, Beijing, China

Synonyms

Natural language generation (NLG)

Definition

Text generation is a subfield of natural language processing. It leverages knowledge in computational linguistics and artificial intelligence to automatically generate natural language texts, which can satisfy certain communicative requirements.

Historical Background

Research work in the text generation field first appeared in the 1970s. Goldman's work on natural language generation from a deep conceptual base appeared in [2]. In the 1980s, more significant work was contributed in this field: McDonald saw text generation as a decision making problem [6], Appelt on language planning (1981), McKeown [8]. In the 1990s, a generic architecture for text generation was discussed, Reiter [10], Hovy [3]. Still today, variations on the generic architecture is a still a widely discussed question, Mellish et al. [9].

Foundations
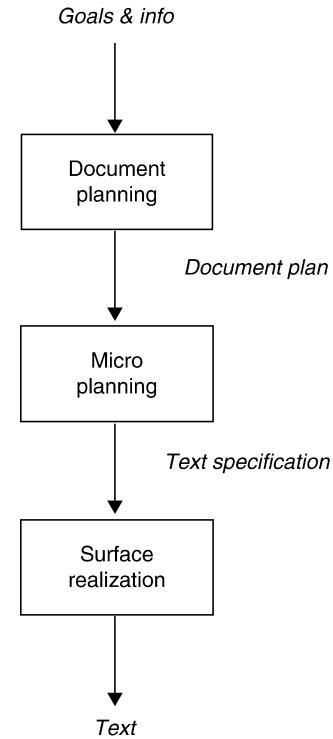
Text Generation, or Natural language generation (NLG), is usually compared with another subfield of natural language processing – natural language understanding (NLU), which is generally considered as the inverse process of the former. Because in a highly abstract level, NLG task synthesizes machine representation of information into natural language texts, while NLU task parses and maps natural language texts into machine representations. However, upon inspection at a more concrete level, they can hardly be seen as "opposite," because they are very different in problem sets, and by internal representations.

Text Generation System Architecture

Input and Output   The input of text generation system is information represented in non-linguistic format, such as numerical, symbolical, graphical, etc. The output is understandable natural language in text format, such as messages, documents, reports, etc.

Architectures

The Generic Architecture   Despite difference in application backgrounds and realization details, many of the current text generation systems followed a general architecture, which is known as the *Pipelined Architecture* or *Consensus Architecture*, usually described as in Fig. 1([11]; Edward Hovy also had a similar representation for this architecture).

Text Generation. Figure 1. A classical architecture for text generation.

As seen in the Fig. 1, the "Pipelined Architecture" describes a general strategy of tackling text generation problem from macro to micro, from inner structure organization to outer surface realization. Thus, language components such as paragraphs, sentences, and words will be coherently arranged together to meet certain communicative requirements.

The following are the detailed descriptions of the above stages:

Stage 1: Document Planning
Also known as Text Planning, Discourse Planning or Macro Planning). This includes:

- Content determination: Also know as content selection and organization, which is to discover and determine the major topics the text should cover, given a set of communicative goals and representations of information or knowledge.
- Document structuring: Determining the overall structure of the text/document. This structure categorizes and organizes sentence-leveled language components into clusters. The relationship

between different components inside a cluster can be explanatory, descriptive, comparative, causal, sequential, etc.

Stage 2: Micro Planning

Also know as Sentence Planning. This is to convert a document plan into a sequence of sentence or phrase specifications, including:

- Aggregation: To combine several linguistic structures (e.g., sentences, paragraphs) into a single and coherent structure. An example: Tomorrow will be cold. Tomorrow will be windy.
  →Tomorrow will be cold and windy.
- Lexicalization: To choose appropriate words from possible lexicalizations based on the communicative background. Examples: (i) buy, purchase, take, etc, (ii) a lot of, large amounts of, etc.
- Referring expression generation: To choose or introduce different means of reference for sentences, such as pronouns (*pronominalization*). There is usually more than one way to identify a specific object, for example: "Shakespeare, " "the poet and playwright, " "the Englishman, "and "he/him" can all point to the same object. Example: Andrew wanted to sing at the birthday party.
  →He wanted to sing at the birthday party.
  →The boy wanted to sing at the birthday party.

Stage 3: Surface realization

Also know as Speech Synthesis. This is to finally synthesize the text according the text specifications made in the previous stages.

- Structure realization: To mark up the text's surface structure, such as an empty line, or the boundaries between paragraphs, etc.
- Linguistic realization: To smooth the text by inserting function words, reorder word sequences, and select appropriate inflections and tenses of words, etc.

Other Architectures: Although the *Pipelined Architecture* provides a considerably articulate routine for text generation, it also provides predetermined restrictions for each stage in the process. Thus, the flexibility it can provide is limited, and is especially true for those sub-tasks in micro planning and surface realization stages. For example, the need for lexical selection can happen at any stage of the process. Thus, variations of the generic architecture and other methodologies have

been discussed by many researchers (a recent discussion, Chris Mellish et al. [9]).

Key Applications

1. Routine documentation or information generation: examples of information are weather forecast descriptions, transportation schedules, accounting spreadsheets, expert system knowledge bases, etc. Examples of documentation are technical reports and manuals, business letters, medical records, doctor prescriptions, etc.
2. Literary writing: such as stories, poems, lyrics, couplets, etc. (Chinese couplet writer: generating a couplet sentence according to a given one. http://duilian.msra.cn).

Cross-references

▶ Text Summarization
▶ Text Representation
▶ Text Normalization
▶ Text Segmentation
▶ Text Analytics
▶ Text semantic Explanation

Recommended Reading

1. Dale R. Introduction to the special issue on natural language generation. Comput. Linguistics, 24 (3):346–353, 1998.
2. Goldman N.M. Computer Generation of Natural Language from a Deep Conceptual Base. Ph.D. thesis, Stanford University, CA, 1974.
3. Hovy E.H. Language generation, Chapter 4. In Survey of the State of the Art in Human Language Technology, G.B.Varile, A. Zampolli (eds.). Cambridge University Press, Cambridge, 1997, pp. 139–163.
4. Hovy E.H. Natural language generation. Entry for MIT Encyclopedia of Computer Science. MIT Press, Cambridge, MA, 1998, pp.585–588
5. Hovy E.H. Language generation. Entry for Encyclopedia of Cognitive Science, article 86. McMillan, London, 2000.
6. McDonald D.D. Natural Language Production as a Process of Decision Making Under Constraint. Ph.D. thesis, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1980.
7. McDonald D.D. *I*Natural language generation, Chapter 7. In Handbook of Natural Language Processing, Dale, R. H. Moisl, H. (eds.). Somers Marcel Dekker, New York, NY, 2000, pp. 147–180.
8. McKeown K.R. Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text. Cambridge University Press, Cambridge, 1985.
9. Mellish C, et al. A reference architecture for natural language generation systems. Nat. Lang. Eng., 12(1):1–34, 2006.

10. Reiter E. Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? In Proc. 7th Int. Conf. on Natural Language Generation, 1994, pp. 163–170.

11. Reiter E. and Dale R. Building Natural Language Generation Systems. Cambridge University Press, Cambridge, 2000.

Text Index Compression

Gonzalo Navarro
University of Chile, Santiago, Chile

Synonyms

Inverted index; List; File compression

Definition

Text index compression is the problem of designing a reduced-space data structure that provides fast search of a text collection, seen as a set of documents. In Information Retrieval (IR) the searches to support are usually for whole words or phrases, either to retrieve the list of all documents where they appear (full-text searching) or to retrieve a ranked list of the documents where those words or phrases are most relevant according to some criterion (relevance ranking). As inverted indexes (sometimes also called inverted lists or inverted files) are by far the most popular type of text index in IR, this entry focuses on different techniques to compress inverted indexes, depending on whether they are oriented to full-text searching or to relevance ranking.

Historical Background

Text indexing techniques have been known at least since the 1960's (see, for example, the book *Automatic Information Organization and Retrieval*, 1968, by Gerard Salton, one of the pioneers in the area). Initially departing from the analog manual indexing process, where a short list of keywords was associated to each document from a collection, the increase in computational and storage capabilities quickly led to the so-called "full-text model", where every text word would be searchable (except for a few so-called "stopwords", which are too frequent and do not carry any meaning nor discriminative power, e.g., articles and prepositions). The so-called "inverted indexes" (or inverted lists, or inverted files), which are also modeled upon the traditional inverted index found at the end of books, have been since then the canonical model for indexing text collections. Most of the indexes used nowadays in Information Retrieval (IR) are variants of the inverted index. These mainly differ depending on the precise type of task that is to be carried out: Sometimes the application needs to find all the documents (and even exact positions within them) where some search terms appear; sometimes only a few "good" documents are wanted because the end user is a human.

Depending on the type of inverted index, it might take as little as 10% of extra space over the text size, or as much as 100% and even more. Nowadays the space, at least in secondary storage media, is extremely cheap and virtually unlimited. However, there are also extremely large text collections (for example the Web) where paying a significant amount of extra space for the index is not irrelevant. However, the most compelling reason to reduce the index size is the large gaps in the memory hierarchy: It is several orders of magnitude faster to transfer data from the main memory than from secondary storage. Hence, a reduction in space translates into an almost proportional increase in throughput at query processing, as the extra processing time for decompression is almost negligible. In networked environments, which are also becoming common to cope with the large demands in storage space and processing power, the same considerations apply with respect to network transfer time. For decades, CPU speeds have been increasing at an exponential rate, whereas disk speeds have not improved much. Moreover, new memory levels (caches) have appeared between the CPU and the main computer memory. This has only made it more and more attractive to design text indexes that fit in little space, even at the expense of needing more sophisticated decompression mechanisms. See [14,15] for a recent exhaustive coverage of the topic.

Foundations

An *inverted index* is formed by two main parts:

1. *Vocabulary.* Is the set of all the different *words* in the collection. The definition of what is a word may depend on the application. Not only does it involve delimiting them, but also determining which normalization processes will be carried out on them, e.g., upper/lower case conversion, removal of stopwords, stemming, etc. Once defined, the word becomes the unit of retrieval: one can search for words or for sequences of words (*phrases*), but not

for, say, a part of the word (although there are some exceptions in text retrieval systems offering extended functionalities which are not covered here, see e.g., [3]).

2. *Postings.* Is a list of "occurrences" of each vocabulary word in the collection. Depending on the application, the index might store the list of document identifiers where the word appears, or the list of exact positions (byte offsets or word offsets), or the list of document identifiers plus a "weight" associated to it, which computes according to some formula the importance of the word in that document, or the list of word positions plus data on the field where the word appears, or even color or font size of each occurrence.

For concreteness, this entry will assume that a policy for defining and normalizing words has been fixed, and will focus on the two most important type of posting lists: one storing exact word positions within documents (useful for full-text retrieval), and another storing document identifiers and weights (useful for relevance ranking). Because of the different processing needs, these lists might be stored in different orders. This is relevant because ordering is the key to inverted list compression.

A widely accepted statistical rule in IR [6] establishes that the vocabulary grows sublinearly with the collection size, more precisely as $v = O(n^\beta)$ for some $0 < \beta < 1$ that depends on the collection. Different experiments show that β is actually around 0.5 [3], which means in practice that, even for very large collections, it is feasible to maintain the vocabulary in the main memory of the computer. For this reason, most of the efforts in index compression have focused on compressing the postings.

Inverted Indexes for Full-Text Retrieval

In full-text retrieval, queries are words or phrases and the task is to find *all* the documents where the word or phrase appears, possibly giving also the position(s) of the occurrences within each retrieved document. The most convenient index organization for word queries is a scheme storing basically all the answers to all the v possible queries: For each vocabulary word, the list of all the documents where it appears is stored in increasing order of document identifier. If exact occurrence positions are desired, and one does not want to sequentially scan the retrieved documents in

order to find them, then the index must also store all the occurrence positions of each word within each document, also in increasing order. Because of the other types of queries that are usually necessary to support, it is usually advantageous to store the list of document identifiers contiguously and separated from the list of positions, so that the document identifiers alone can be retrieved with fewer disk accesses.

The important point is that the postings consist of *lists of increasing numbers* (document identifiers or occurrence positions). Moreover, for word queries, those lists are accessed from the beginning to the end. An obvious idea is to encode the *differences* between consecutive entries in the list. The longer the list, the smallest the differences between consecutive values. Because the number of occurrences of vocabulary words is usually highly skewed [3] (following a Zipf–Mandelbrot distribution [8]), there will be very long (and thus very compressible) lists and many short lists, where the long lists will contain a significant fraction of the whole data. Hence, a technique that represents those differences in a way that smaller numbers use a shorter representation should achieve compression. This requires that a different amount of bits is used to represent different numbers, as opposed to the classical data representation where a fixed number of bits is allocated for every possible number.

A first idea would be to use exactly the bits that the number uses, for example the binary representation of 9 is 1001_2, and hence one would like to represent it using 4 bits. The problem, of course, is that one cannot decode the individual numbers from the concatenation of their representations if one uses this method. For example, 10011101 could be 9,1,5 or 9,13. A large part of the research in inverted index compression refers to the design of *self-delimiting codes*, which (i) can be uniquely decoded from the concatenation of the codes, (ii) give shorter codes to smaller numbers, (iii) preferably can be efficiently decoded.

Several self-delimiting coding schemes are described in the book *Managing Gigabytes* [14], one of the best references on text index compression. Let $x > 0$ be a number to encode and $|x|$ the number of bits needed to code it (i.e., the highest bit set in the binary representation of x is at position $|x|$). Some of the most famous codes are described next.

1. Elias' γ-code emits $|x| - 1$ 0's followed by the $|x|$ bits representing x. For example, the code for

$x = 23 = 10111_2$ is 0000 10111 (which are here artificially separated in two parts). To decode the first number from a bit stream, one finds the next 1, and if one skipped $\ell - 1$ 0's before it, on reads the next ℓ bits starting from that 1. Those ℓ bits form x. This works because the representation of x always starts with a 1 (i.e., its most significant bit). Elias' γ representation of x takes $2|x| - 1 = 1 + 2\lfloor \log_2 x \rfloor$ bits. The representations of the first numbers are as follows: $1 = 1_2 \rightarrow 1, 2 = 10_2 \rightarrow 010, 3 = 11_2 \rightarrow 011, 4 = 100_2 \rightarrow 00100,...$

2. Elias' δ-code first codes $|x|$ using Elias' γ-code, and then emits the $|x| - 1$ least significant bits of x (as one knows that the most significant bit is a 1). For example, the code for $x = 23 = 10111_2$, $|x| = 5 = 101_2$, is 00 101 0111. The length of the δ-code of x is $2||x|| - 1 + |x| - 1 = 1 + 2\lfloor \log_2(1 + \lfloor \log_2 x \rfloor) \rfloor + \lfloor \log_2 x \rfloor$. These codes are theoretically appealing because the space they require is $\log_2 x + O(\log \log x)$, which is asymptotically the same as just writing x. For example, it is not hard to prove that, using this scheme to encode all the word offsets of the occurrences, the whole index needs $nH_0 + O(n \log \log n)$ bits, where n is the number of words in the text collection and H_0 is its zero-order entropy seen as a sequence of words [9]. Thus the space is similar to that obtained when compressing the text using a good word-based zero-order compressor [4].

3. Rice codes are parameterized by a value r. The lowest r bits of x are represented in binary form (even if $|x| < r$), preceded by the unary representation of $\lfloor x/2^r \rfloor$. A different r value (the one yielding least space) can be chosen for each inverted list. An extension are Golomb codes, where instead of the quotient and remainder of the division by 2^r, any divisor d can be used. These are usually reported to achieve the least space consumption [14].

There are many other self-delimiting codes, such as Fibonacci codes [7], variable-byte codes [13], Dense codes [4], Simple codes [2], and many others. Some trade a slight loss in space for a large gain in decoding time.

Searching for Phrases Using Inverted Indexes

The described encodings of inverted lists must be decoded from the beginning, which as explained is appropriate to answer queries consisting of a single word. Phrase queries, however, are more complicated to deal with. If the index stores only document identifiers, all it can do is to intersect the list of documents of the different words, and then it is necessary to sequentially search the candidate documents for the exact phrases. If the index, instead, stores occurrence positions, it can look for the consecutive occurrences of the involved words within the same document. (This is especially convenient if the index stores word offsets rather than byte offsets of occurrences. Storing byte offsets, on the other hand, is more convenient to highlight the occurrences in the text.)

In both cases, a list intersection-like capability is necessary. Incidentally, note that another popular operation in inverted indexes is the conjunctive query, that is, find the documents where all the query words appear. Those are obviously solved by intersecting the lists of document identifiers as explained. Disjunctive queries (find the documents where some of the words appear) are solved by merging lists of document identifiers. Although there is not an especially clever way of merging lists other than traversing them all sequentially, much research has been done on the problem of efficiently intersection increasing lists of numbers. Although sequential intersection is an option as well, one can do better when one list is much shorter than the other, as it is frequently the case of the highly skewed length distributions that arise in natural language text collections [8].

Describing those techniques is not in the scope of this entry. A recent example of this research, which indeed takes compression into account, can be seen in [12]. In all cases the main idea is that one can search the longer list for the elements in the shorter list. This implies that some degree of random access is necessary in the encoded lists. For this sake, the natural choice [14] is to insert some absolute list values at regular intervals in the compressed list data, so that the search can be first carried out on those sampled values and only one chunk between consecutive samples must be decompressed. There is also some recent research on how those absolute values should be physically organized [5], as it turns out to be more convenient to store all the samples in contiguous form.

Note that, independently of the encoding technique used, the success of the compression depends on what is indexed. If one indexes the occurrences of all the text words, the space used by the index will be at best around 40% of that of the original (uncompressed) text, which can be reduced to around 20% by removing the stopwords. If only document

identifiers are recorded, the space can be typically 10–20% depending on the size of the documents. A technique to achieve even less space, at the expense of higher time to solve queries, is called *block addressing* [10]. The idea is to cut the text into blocks, so that the index stores the blocks where each word appears. The index is used to filter out some blocks upon queries, and the others have to be sequentially traversed. Hence, the block size yields a space/time trade-off, where reasonable performance on moderate-sized collections can be achieved with as little as 5% of index space overhead.

Finally, it is interesting to mention that there exist compressed indexes for what is (also) called "full-text searching", in the sense that the text is seen as a sequence of symbols and the index is able of retrieving any text substring (not only words or phrases) [9]. Those indexes were classically much larger than the text, and recent developments have shown that they can be compressed to sizes competitive to those of inverted indexes for word and phrase retrieval. Their search times are still larger than those for inverted indexes, although research is being carried out on applying them over a text regarded as a sequence of words (not letters). In this case they can also search only for words and phrases, but their space becomes even better than using inverted indexes, and performance for phrase searching is competitive.

Inverted Indexes for Relevance Ranking

Inverted indexes are also used to rank documents by "relevance" to a given query, so as to return a small set of those ranking higher. There are many formulas for computing relevance [3], yet the most popular ones build on two components: one is the *term frequency* $tf_{w,d}$, which is the number of times word w appears in document d, and the other is the *inverse document frequency* idf_w, which is the logarithm of the inverse of the fraction of the documents where word w appears. Note that, while idf_w is just one value per vocabulary word (and hence easy to maintain in main memory with the vocabulary), there is one $tf_{w,d}$ value per entry in the inverted index: For each word, the index stores the documents where it appears and the associated term frequency for each.

Because the relevance formula gives more weight to terms with higher term frequency, it is sensible to store the document identifiers of each word by decreasing term frequency value, rather than by increasing document identifier. This enables efficient algorithms that examine only a short prefix of the inverted lists [11]. The problem is that, although the decreasing term frequencies can be stored differentially, this is not possible anymore with the document identifiers accompanying them. Fortunately, because of Zipf–Mandelbrot law [8], it turns out that many of the term frequencies are small values, and therefore there are long runs of equal term frequencies, especially at the tail of the lists. Within those runs, one can still reorder the documents by increasing document identifier and encode them differentially. Recently, it has been shown that reducing the precision of the exact term frequency values is not only advantageous for compression purposes (which is obvious) but also for retrieval effectiveness [1].

Key Applications

Any application managing natural language text collections that have to be searched, and which are massive enough to discard sequential search as a solution, needs some kind of index. Index compression not only saves space, but more importantly, disk and network transfer time. A canonical example application are Web search engines.

Future Directions

Inverted index technology, even including compression, is rather mature. Still, it faces enormous efficiency challenges, especially those coming from Web search engines. The most active research fields related to inverted index compression are on encodings that permit fast decoding (e.g., [2]), and the interactions between information discarded to boost compression and the resulting retrieval quality of the index (e.g., [1]). The possibility of applying "true full-text indexes" [9] to natural language text is also extremely interesting, as it brings in radically new compression methods as well as algorithms for solving phrase queries. Yet, this trend is rather preliminary, and much research is needed to compete with inverted indexes in secondary memory scenarios.

Other challenges in inverted indexes, only mildly related to compression, are distributed indexes (how to split the collection and/or the indexes across multiple machines to boost performance), dynamic indexes (how to efficiently update the index when the

collection changes), and extensions (how to cope with more complex queries, for example allowing approximate searches).

Experimental Results

Experiments can be found in the most recent cited papers, as most of them are of practical nature.

URL to Code

Probably the best known public domain implementation of compressed indexes was the *MG System* (http://www.cs.mu.oz.au/mg), closely related to the book *Managing Gigabytes* [14]. This system is now almost 10 years old, and is being replaced by its successor, *Zettair* (http://www.seg.rmit.edu.au/zettair).

Cross-references

▶ Inverted Indexes

Recommended Reading

1. Anh V. and Moffat A. Simplified similarity scoring using term ranks. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 226–233.
2. Anh V. and Moffat A. Improved word-aligned binary compression for text indexing., IEEE Trans. Knowl. Data Eng., 18(6): 857–861, 2006.
3. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley, Reading, MA, 1999.
4. Brisaboa N., Fariña A., Navarro G., and Paramá J. Lightweight natural language text compression., Inf. Retriev., 10:1–33, 2007.
5. Culpepper S. and Moffat A. Compact set representation for information retrieval. In Proc. 14th Int. Symp. String Processing and Information Retrieval, 2007, pp. 137–148.
6. Heaps H. Information Retrieval – Computational and Theoretical Aspects. Academic Press, New York, 1978.
7. Kautz W. Fibonacci codes for synchronization control. IEEE Trans. Inf. Theor., 11:284–292, 1965.
8. Mandelbrot B. An informational theory of the statistical structure of language. In Proc. Symp. on Applications of Communication Theory, 1952, pp. 486–500.
9. Navarro G. and Mäkinen V. Compressed full-text indexes. ACM Comput. Surv., 39(1):article 2, 2007.
10. Navarro G., Moura E., Neubert M., Ziviani N., and Baeza-Yates R. Adding compression to block addressing inverted indexes.. Inf. Retriev., 3(1):49–77, 2000.
11. Persin M., Zobel J., and Sacks-Davis R. Filtered document retrieval with frequency-sorted indexes. J. Am. Soc. Inf. Sci., 47 (10):749–764, 1996.
12. Sanders P. and Transier F. Intersection in integer inverted indices. In Proc. Workshop on Algorithm Engineering and Experiments, 2007.
13. Scholer F., Williams H., Yiannis J., and Zobel J. Compression of inverted indexes for fast query evaluation. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 222–229.
14. Witten I., Moffat A., and Bell T. Managing Gigabytes. 2nd edn. Van Nostrand Reinhold, New York, 1999.
15. Zobel J. and Moffat A. Inverted files for text search engines. ACM Comput. Surv., 38(2), 2006.

Text Indexing and Retrieval

HAODA HUANG, BENYU ZHANG
Microsoft Research Asia, Beijing, China

Synonyms

Document index and retrieval

Definition

Text indexing is a preprocessing step for text retrieval. During the text indexing process, texts are collected, parsed and stored to facilitate fast and accurate text retrieval. Text retrieval (also called document retrieval) is a branch of information retrieval in which the information is stored primarily in the form of text. Text retrieval is defined as the matching of some stated user query against a set of texts. As the result of text retrieval, texts are ranked and presented to the user according to their relevance with user query. User queries can range from a few words to multi-sentence full descriptions, which represent the user's information need.

Historical Background

Text indexing is the most fundamental part of a retrieval system. Over the past two decades, the corpus size of typical retrieval system has increased dramatically. The Text REtrieval Conference (TREC) (http://trec.nist.gov/) that started in 1992 only provides document collection consisting of less than 1 million texts in the 1990s. Today, the largest TREC test collection, named GOV2, is a 2004 crawl of the.gov top-level domain. GOV2 consists of approximately 25 million web pages (428GB of text). Moreover, a web search engine is believed to have far more items in its index. For example, in 2005, Yahoo claimed to have more than 20 billion items in its index, which is several

orders larger than GOV2. The retrieval system is often required to return relevant document/text for a user query in a few seconds. Without an index, it is impossible for a retrieval system to achieve this task.

There are many indexing techniques. Among them, inverted index, suffix array, and signature are three typical examples. Signature files were popular in the 1980s, but since this scheme is inferior to inverted files in terms of speed, size and functionality, it is used less nowadays. Suffix trees are faster for phrase searches, but they are not easy to build and maintain. Finally, due to its simplicity and efficiency, inverted index is currently the most popular indexing scheme and is used for many applications.

Text retrieval is the core part of a retrieval system. There have been several decades of research on text retrieval, which has resulted in a fruitful range of beautiful, effective text retrieval models, such as Boolean model, vector model, and probabilistic model.

The Boolean model is one of the earliest models. Despite its simplicity, the Boolean retrieval model was popular among many large commercial information providers until the early 1990s. A noticeable advantage of the Boolean model is that in Boolean models, queries are represented in the form of a Boolean expression of terms, that is, an expression in which terms are combined with the operators AND, OR, and NOT. Many users prefer the Boolean query model because Boolean queries are precise and offer users greater control and transparency over the retrieved text. But the Boolean model also suffers from some major drawbacks. For example, the Boolean model judges a document to be either relevant or non-relevant in respect to a given query without any notion of grading scale, which does not achieve good performance in practice.

Different from the Boolean model, the vector model largely uses free text queries, which consist of one or more words but do not use operators to build up the query expressions. Query and documents are presented as a weighted vector over a fixed dictionary, and the documents are ranked by their Cos similarity with query. The vector model is better than Boolean model in that it is able to deal with cases in which documents only partially match the query. The famous tf-idf scheme for the vector model is very influential in the research of text retrieval.

Besides the simple Boolean model and vector model, probabilistic models have achieved great success regarding the text retrieval task. Probabilistic models are mainly guided by Probability Ranking Principle (PRP): "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

The Binary Independence Retrieval (BIR) model is one of the earliest probabilistic models for text retrieval. The model has other names, such as the Okapi model and the classical probabilistic model. Similar to the Boolean model, the BIR model documents are represented as binary vectors indexed over a fixed vocabulary. The representation ignores the number of times the term occurs, and is only able to capture whether or not a term occurs in a document. The 2-Poisson model was proposed to overcome this limitation. Under this model, documents are represented as vectors of term frequencies. Probabilistic models are not only solid from theoretical perspective, but also perform well in practice. An empirical, hand-crafted approximation of the 2-Poisson model, called BM25, showed good performance over many data collections. It was first introduced at TREC in 1995, and is still a strong baseline for current text retrieval research.

The probabilistic language model, which was first applied to speech recognition and machine translation, has also been successfully applied to information retrieval in 1998 [4]. Since then, research to improving the language model has been an active area. Also, the probabilistic language model is a state-of-the-art text retrieval models for its robust, highly effective performance in the practice.

In all these models, documents are represented as a bag of words, where the exact ordering of the terms in a document is ignored but the number of occurrences of each term is material.

There are other models that go beyond the bag of words assumption, such as the Markov Random Field model, n-Gram language models, and the Indri Inference network model. The Markov Random Field model [2,3] for information retrieval, which was proposed in 2005 and improved in 2007, is demonstrated to be consistently and significantly better than

probabilistic language model in several document collections, which shows that it is beneficial to incorporate dependency and features into text retrieval model.

Foundations

Among many text indexing schemes, the inverted index is the most popular. It could well support bag of words based text retrieval methods, like the tf-idf vector model, the BM25 model, and probabilistic language model. The process of building an inverted index is briefly described here. The major steps include:

1. Collecting the interested texts to form the text collection. For example, if the interested texts are the web pages on the internet, web pages should be crawled to form the text collection.
2. Tokenizing the text, which turns each document into a list of tokens. Generally, a text is represented as a continuous character stream. In this step, text is parsed and segmented into terms, digits, and email address and punctuations, such as ",", ".", "!", "?" and "-", are removed.
3. Doing linguistic preprocessing, which typically includes removing stemming and stop words, and then producing a list of indexing terms. Stemming refers to the process of reducing terms to their stems or root variants. For example, "computer," "computing," "compute" are reduced to "comput." For English, the most popular stemmer is Martin Porter's stemming algorithm. Stop words removal eliminates stop words from the documents. Common stop words include "a," "the," "of," "is" etc.
4. Creating the inverted index, which mainly consists of a dictionary and postings. The dictionary includes all the terms and the number of times they appear in the text collection. The postings include the text in which the terms appear. Thus the postings of a query could be obtained through the intersection operator of the query terms' postings.

With the support of the inverted index, many text retrieval models could be implemented efficiently. Here, the tf-idf vector model and probabilistic language model are taken as examples:

In the tf-idf vector model, either the query or document is represented as the weighted vector, with each component calculated as the product of term frequency (TF) and inverse document frequency

(IDF) of the corresponding term. Term frequency is the number of times the term appears in a query or a document. The inverse document frequency of the term t is defined as

$$idf(t) = \log\left(\frac{N}{n_t}\right)$$

where N is the total number of texts in the collection and n_t is the number of documents with term t. The query vector is represented as $q = (tf_a(t_1) \cdot idf(t_1), tf_a(t_2) \cdot idf(t_2),..., tf_a(t_n) \cdot idf(t_n))$, and the document vector is represented as $d = (tf_d(t_1) \cdot idf(t_1), tf_d(t_2) \cdot idf(t_2),..., tf_d(t_n) \cdot idf(t_n))$. Then, documents are ranked by their Cos similarity with the query. The Cos similarity between the query and document is calculated as below:

$$sim(q, d) = \frac{\sum_{i=1}^{n} q_i \times d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \times \sqrt{\sum_{i=1}^{n} d_i^2}}$$

The TF and IDF together discriminate along two dimensions – informativeness (IDF) and aboutness (TF). The IDF component is used to discriminate between informative and non-informative query terms. Terms with high IDF occur rarely in the collection, and thus are considered more informative. The incorporation of TF component is based on the intuition that documents that contain more mentions of a term are more "about" this term.

The probabilistic language model for information retrieval is based upon the idea that a document is a good match to a query if the document model is likely to generate the query. Therefore, language model documents are ranked by query generation probability $p(q|d)$. There are several variant realizations for the language model. Among them, the query likelihood model [4] is the original and basic method for using language models in IR. The most common way to evaluate query generation probability $p(q|d)$ is using the multinomial unigram language model. Under this model, $p(q|d)$ is approximated as

$$p(q|d) = \prod_{w \in q} p(w|d)$$

Due to the data sparseness, the generation probability of query term absent in the document will be zero. Generally a smoothing technique is applied to overcome this problem. Please refer [6] for more details.

Key Applications

Text retrieval has many applications. It is used in digital libraries to help people quickly find desired books or articles. It could also be used in desktop search to help people instantly find documents, such as e-mail or other files, in a computer. Moreover, it is the fundamental basis of all internet search engines.

Future Directions

Most past research on text retrieval is based on the bag of words assumption, which has resulted in very fruitful models. These models have achieved rather good performance in the past, but appear to have reached a plateau; thus, their improvement has dwindled for several years. Recently, some text retrieval models (for example, Markov Random Field model for information retrieval) have tried to go beyond the bag of words assumption, and have achieved consistent and significant improvement. This indicates that it would be beneficial to incorporate dependency and features of documents into the text retrieval model; thus, more dedicated models may be developed in the future to improve retrieval performance further.

Data Sets

For testing the effectiveness of text indexing and retrieval strategies, TREC text datasets (http://trec.nist.gov/) are commonly used in the research community.

Cross-references

▶ Indexing
▶ Information Retrieval
▶ Inverse Document Frequency
▶ Term Frequency

Recommended Reading

1. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, MA, 2008.
2. Metzler D. and Croft W.B. A Markov random field model for term dependencies. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 472–479.
3. Metzler D.A. Beyond bags of words: effectively modeling dependence and features in information retrieval, Ph.D. thesis, University of Massachussetts, 2007.
4. Ponte J. and Croft W.B. A language modeling approach to information retrieval. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 275–281.
5. Ricardo B.-Y. and Berthier R.-N. Modern Information Retrieval. Addison Wesley Longman, New York, NY, 1999.
6. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to ad hoc information retrieval. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 334–342.

Text Indexing Techniques

EDLENO SILVA DE MOURA
Federal University of Amazonas, Manaus, Brazil

Definition

Text indexing is the act of processing a text in order to extract statistics considered important for representing the information available and/or to allow fast search on its content. Text indexing operations can be performed not only on natural language texts, but virtually on any type of textual information, such as source code of computer programs, DNA or protein databases and textual data stored in traditional database systems.

Historical Background

Efforts for indexing electronic texts are found in literature since the beginning of computational systems. For example, descriptions of Electronic Information Search Systems that are able to index and search text can be found in the early of 1950s [3].

In a seminal work, Gerard Salton wrote, in 1968, a book containing the basis for the modern information retrieval systems [5], including a description of a model largely adopted up to now for indexing texts, known as Vector Space Model. Other successful models for indexing texts were proposed since then, such as, for instance, Probabilistic Models and Language Models, which are discussed in detail in [2].

Text indexing operations have also received attention in the last decades as a basic operation in a large variety of applications. They are used in several basic algorithms related to data management, such as data integration and data disambiguation. Bioinformatics is another area where text indexing has been largely applied. In these cases, biological databases containing data such as DNA or protein information are indexed as texts in order to provide fast access to their content.

Foundations

The implementation of text indexing operations requires data structures for allowing fast access to the indexed information. The most used data structure for this purpose is known as *inverted index* or *inverted file*. An inverted index is a data structure composed of: (i) a vocabulary that contains all the distinct words indexed found in the text and (ii), for each word t of the vocabulary, a list that contains statistics about the occurrences of t in the text. Such list is known as the inverted list of t.

Inverted indexes allow fast search for statistics related to the distinct words found in a text. They are designed for using words as the search unit, which restricts their use in applications where words are not clearly defined or in applications where the system does not use words as the search unit. The statistics stored in an inverted index may vary according to the target application. Two non-exclusive alternatives usually found in literature are to record the position of all word occurrences in the text and to record general statistics about the word occurences in text units. Indexing all the word occurrences is useful in applications where positional information should be taken into account, such as when it is necessary to allow search for phrases or proximity queries. Inverted indexes that store word statistics are usually deployed in systems that adopt information retrieval models, such as the Vector Space Model, and in this case the text is divided into units of information, usually doucments. For instance, in web search engines, these units are the pages crawled from the web, while the whole set of pages compose the indexed text.

A third alternative to produce inverted indexes with intermediate space overhead and allow search for phrases and positional queries is to use a scheme known as block addressing [4]. Block addressing is a technique to reduce the space requirements of an inverted index. It was first proposed in a system called *Glimpse* [4]. The idea is that the text is logically divided into blocks, and the occurrences do not point to exact word positions but only to the blocks where the word appears. Space is saved because there are less blocks than text positions (and hence the pointers are shorter), and also because all the occurrences of a given word in a single text block are referenced only once.

Searching in a block addressing index is similar to search in a full inverted one. The pattern is searched in the vocabulary and a list of blocks where the pattern appears is retrieved. However, to obtain the exact pattern positions in the text, a sequential search over the qualifying blocks becomes necessary. The index is therefore used as a filter to avoid a sequential search over some blocks, while the others need to be checked. Hence, the reduction in space requirements is obtained at the expense of higher search costs.

The use of a block addressing scheme when indexing texts results in a tradeoff between index size and the computational costs for processing queries. The larger are the block units, the smaller is the final index overhead in terms of space. On the other hand, the larger are the block units, the larger are the portions of text that should be sequentially traversed when searching for a pattern in the text.

Although this combination of sequential search and index seems to produce less efficient search systems at first, block addressing indexes may achieve interesting combinations of cost for search time and space overhead. Block addressing was analyzed in [1], where it is analytically proved and experimentally verified that a block addressing index may yield sub-linear space overhead and, at the same time, sub-linear query time. Traditional inverted indexes pointing to words or documents achieve only the second goal. Empirical experiments performed in [?] indicate that in practice, $O(n^{0.85})$ space and query time can be obtained for exact queries in natural language texts.

Other data structures also adopted for indexing texts with the goal of allowing fast access to them are *signature files* and *suffix trees* [2,7]. Signature files are indexes that use a hash function to map the words found in a text to bit masks. The text is divided into blocks, and each block b is indexed by storing the result of a bit-wise OR operation over all the masks of the words that occur in b. This final mask represents the block in the signature file index. The search operation is performed through a bit-wise AND between the searched word and each block mask in the collection. Potential matches are reported whenever the bit-wise AND operation results in a number equal to the mask of the searched word. Note that false matches may arise, and thus the matched blocks should be inspected for confirming the occurrence of the word or not. Therefore, the signature file works as a filter for reducing the amount of text traversed in search operations.

The search in signature files is usually less efficient than the search when using inverted indexes. Further, inverted indexes support a larger set of search operations than signature files. For typical document indexing applications, signature files do not perform well compared to inverted indexes, being much larger and they are more expensive to build and update [8]. Thus, the use of inverted indexes in text indexing operations is more frequent than the use of signature files.
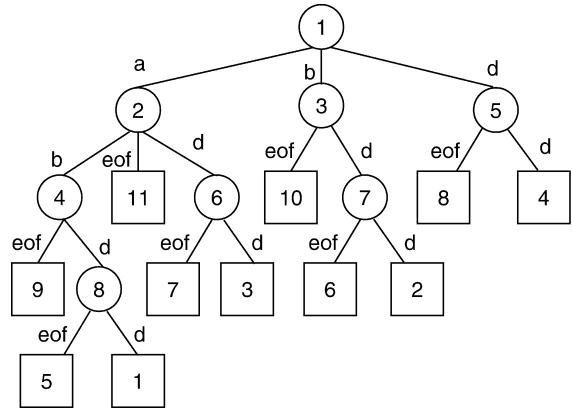
Another approach for indexing text databases regards the text as a long string. Suffix trees can index the text as a set of symbols according to the desired granularity of the search operations. For instance, by using suffix trees, it is possible to take all the characters of the text as index entries. This flexibility allows indexing texts that are written in languages where the words are not clearly separated from each other, such as it happens in some Asian languages, or even in texts where the concept of word does not exist, such as in DNA or protein databases.

Each position in the text is called a *suffix*. A suffix is defined by a starting position and extends to the right as far as needed or to the end of the text. Depending on the character sequences that are to be searched, the user must define the *index points*, i.e., which positions of the text will be indexed. In text databases, it is customary to index only word beginnings, but in many applications for text indexing it is necessary to index all the characters of the text. Suffix trees are data structures designed for indexing text using the suffix model, thus considering each entry point indexed in the text as a suffix of the whole text. In the suffix trees, each distinct path from the root to a leaf represents a unique suffix. For instance, consider the text fragment: "abadabadaba", which contains the suffixes described in Table 1.

Each suffix found in this example finishes with an implicit special symbol, marking the end of the text. Figure 1 describes the suffix tree for storing all the eleven suffixes of this text fragment. The circles in the trees contain internal nodes and their numbers indicate the position in the text where their children differ from each other. The square nodes are the leaves of the suffix tree and their numbers represent suffix ids (suffixes are usually represented by their position in the text). The letters marking each node in the tree indicate the value found on the positions where the node differ from their siblings nodes. The "eof" represents the end of the text is this example.

Text Indexing Techniques. Table 1. Suffixes found in the text fragment "abadabadaba"

Suffix	Suffix ID
"abadabadaba"	1
"badabadaba"	2
"adabadaba"	3
"dabadaba"	4
"abadaba"	5
"badaba"	6
"adaba"	7
"daba"	8
"aba"	9
"ba"	10
"a"	11

Text Indexing Techniques. Figure 1. Suffix tree for the text fragment "abadabadaba".

For instance, in the root node, the number 1 indicates that the suffixes are different in their first position. At this position, the suffixes may have value 'a', 'b' or 'c'. When searching for a suffix starting with an 'a', then the left most sub-tree should be taken in order to continue the search. In this case, the next comparison position to continue the search is position 2. The search continues until the current search position is longer than the search pattern, which means all the patterns in the sub-tree contain the search key; or until the current position in the search key does not match any child of the current node, which means the search key is not in the text.

Using this structure, it is possible to perform a variety of complex search operations at a relatively

low computational cost. One of the drawbacks of using a suffix tree is the space required to store the index. If the space is a restriction, a more compact structure known as suffix array can be adopted. A suffix array is an array of pointers to each suffix indexed in the text. This array is sorted in the alphabetical order of the suffixes it represents and can then be used to perform search tasks. The use of suffix arrays increases the cost of search operations when compared to suffix trees. For instance, a given search pattern can be found in a suffix array at cost $O(log(n))$, where n is the number of suffixes in the text, while the cost for searching a simple pattern in a suffix tree is $O(m)$, where m is the size of the searched pattern, and is not affected by the number of suffix in the text.

A practical problem when using suffix trees or suffix arrays is the cost to build and maintain the indexes. Further, when searching for words inverted indexes are usually faster than suffix trees and suffix arrays. Exceptions may occur when it is necessary to process complex queries, such as regular expression patterns and search for word fragments.

Key Applications

Text indexing techniques have important practical applications, being of great importance in the construction of Web Search Engines, Web Directories and Enterprise Search systems. These techniques are also useful for allowing fast search on DNA or protein databases, which can also be treated as texts. In these cases, the text indexing techniques are adopted to accelerate more complex operations performed over the databases, such as the comparison of DNA fragments allowing small differences between them. Text indexing operations have also played an important role in several algorithms related to data management, such as data integration and data disambiguation.

Cross-references
▶ Inverted Files
▶ IR Retrieval Models
▶ Suffix Trees
▶ Text Retrieval

Recommended Reading

1. Baeza-Yates R. and Navarro G. Block-addressing indices for approximate text retrieval. J. American Soc. for Inf. Sci., 51(1): 69–82, 2000.
2. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, Reading, MA, 1999.
3. Luhn H.P. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1(4):309–317, October 1957.
4. Manber U. and Wu S. GLIMPSE: A tool to search through entire file systems. In Proc. USENIX Winter 1994 Technical Conf., 1994, pp. 23–32.
5. Salton G. Automatic Information Organization and Retrieval. McGraw-Hill, New York, NY, 1968.
6. Salton G., Won A., and Yang C.S. A vector space model for automatic indexing. Inf. Retriev. Lang. Process., 18(11):613–620, November 1975.
7. Witten I., Moffat A., and Bell T. Managing Gigabytes, 2nd edn. Morgan Kaufmann, Los Altos, CA, 1999.
8. Zobel J., Moffat A., and Ramamohanarao K. Inverted files versus signature files for text indexing ACM Trans. Database Syst., 23(4):453–490, December 1998.

Text Mining

YANLI CAI[1], JIAN-TAO SUN[2]
[1]Shanghai Jiao Tong University, Shanghai, China
[2]Microsoft Research Asia, Beijing, China

Synonyms
Knowledge discovery in text (KDT)

Definition
Text mining is the art of data mining from text data collections. The goal is to discover knowledge (or information, patterns) from text data, which are unstructured or semi-structured. It is a subfield of Data Mining (DM), which is also known as Knowledge Discovery in Databases (KDD). KDD is to discover knowledge from various data sources, including text data, relational databases, Web data, user log data, etc. Text Mining is also related to other research fields, including Machine Learning (ML), Information Retrieval (IR), Natural Language Processing (NLP), Information Extraction (IE), Statistics, Pattern Recognition (PR), Artificial Intelligence (AI), etc.
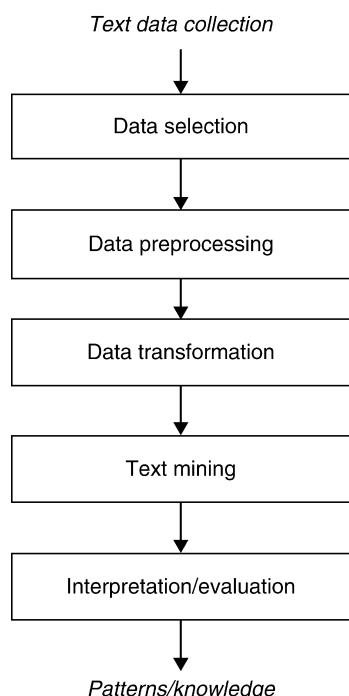
Historical Background
The phrase of *Knowledge Discovery in Databases (KDD)* was first used at 1st KDD workshop in 1989. Marti Hearst [4] first used the term of text data

mining (TDM) and differentiated it with other concepts such as information retrieval and natural language processing.

Foundations

Typical steps of knowledge discovery in databases can be found in [7] by Usama Fayyad et al. The process of knowledge discovery in text data collections is similar, as shown in Fig. 1. Given one collection of text data as input, the goal is to extract patterns/knowledge from them. As the first step, data selection is to select proper data that will be processed and analyzed in the following steps. In the data preprocessing step, the task is to filter out noisy information and do some preliminary processes to facilitate the following steps, e.g., extracting name entities from the text data or part-of-speech tagging of the text data. In the data transformation step, the text data are converted to the format that is easy to be processed by mining algorithms, e.g., the format of vectors, sequences or inverted index tables. Text mining is to apply mining algorithms to find candidate patterns. In the interpretation/evaluation step, the candidate patterns produced in the former step are evaluated and the interesting ones are outputted as final knowledge.

Text data collection

Text Mining. Figure 1. Steps of knowledge discovery from text data.

Text preprocessing strongly affects the success of the outcome of text mining. Tokenization, or splitting the input into words, is an important first step that seems easy but is fraught with small decisions: how to deal with apostrophes and hyphens, capitalization, punctuation, numbers, alphanumeric strings, whether the amount of white space is significant, whether to impose a maximum length on tokens, what to do with non-printing characters, and so on [9]. It may be beneficial to perform some rudimentary morphological analysis on the tokens such as removing suffixes or representing them as words separate from the stem. Tokens may be standardized by using a dictionary to map different, but equivalent, variants of a term into a single canonical form. Once the input is tokenized, some level of syntactic processing is usually required. One operation is to remove stop words. Another is to identify common phrases and map them into single features. Tokenizing a document and discarding all sequential information yields the "bag of words." Additional linguistic preprocessing may also be needed [1]. Part-of-speech tagging (POS) determines the part of speech tag for each word, e.g., noun, verb, adjective. Text chunking aims at grouping adjacent words of a sentence. Word Sense Disambiguation (WSD) tries to resolve the ambiguity for individual words or phrases. Parsing produces a full parse tree for a sentence, which can identify the relation of each word in the sentence to all the others, and typically also its function in the sentence, e.g., subject, object.

Various statistical techniques and learning methods have been employed in text mining, including Naive-Bayes methods, support vector machines (SVM), decision trees, hidden Markov models (HMM), neural networks, etc. Major ways in which text is mined [1,9] are discussed in the following subsections.

Text Classification

Text classification is the assignment of text documents to predefined categories according to their contents. The pre-defined categories are symbolic labels with no additional semantics. When classifying a document, no information is used except for the document's content itself. A large number of feature selection and machine learning techniques have been applied to text classification. Typical approaches extract features from each document, and use the feature vectors as input to a scheme that learns how to classify documents. Using words as features and word occurrence

frequencies as feature values, a model is built for each category. The documents in that category are positive examples and the remaining documents are negative ones. The model predicts whether or not that category is assigned to a new document based on the words in it, and their occurrence counts. Given a new document, each model is applied to determine which categories to assign. Automatic text classification has many practical applications, including indexing for document retrieval, automatically extracting metadata, word sense disambiguation by detecting the topics a document covers, and organizing and maintaining large catalogues of Web resources.

Text Clustering

Text clustering is unsupervised learning in which there is no predefined category or class, but groups of documents that belong together are sought. Clustering schemes not require training data to be pre-classified, and the algorithms themselves are generally far more computation-intensive than supervised schemes. Usually the quality of clustering is considered better if the contents of the documents within one cluster are more similar and between the clusters more dissimilar.

Information Extraction

Information extraction is used to refer to the task of filling templates from natural language input, one of the principal subfields of text mining. A template is a composite structure with slots that are filled by individual pieces of structured information. A commonly-cited domain is that of terrorist events, where the template may include slots for the perpetrator, the victim, type of event, where and when it occurred, etc.

Machine learning has been applied to the information extraction task by seeking pattern-match rules that extract fillers for slots in the template. The rules can be expressed in pattern-action form, and the patterns comprise constraints on words in the surrounding context and the slot-filler itself. These constraints involve the words included, their part-of speech tags, and their semantic classes. An application of information extraction is extracting information from job ads such as those posted on Internet newsgroups.

The extracted information by information extraction can also be used in a subsequent step to learn rules that characterize the content of the text itself.

Document Summarization

A text summarizer strives to produce a condensed representation of its input, intended for human consumption. Useful distinctions can be made between different kinds of summaries: an extract picks certain key sentences scattered throughout the document. In contrast, an abstract contains material that is not present in the input, or at least expresses it in a different way. An indicative summary's main purpose is to suggest the contents of the article without giving away details of the article content. It can serve to entice the user into retrieving the full form [5]. Book jackets, card catalog entries and movie trailers are examples of indicative summaries. An informative summary is meant to represent the original document. Therefore it must contain all the pertinent information necessary to convey the core information and omit ancillary information. Another distinction is between a generic summary, aimed at a broad readership, and a topic-focused one, tailored to the requirements of a particular group of users. Summaries may also be produced from a single document or multiple documents [3]. In single-document summarization, features like word frequency, key phrases, sentence position, sentence length and uppercase words are used. Multi-document summarization extracts a single summary from multiple documents, and is used in the domain of news articles. It departs from single-document summarization since the problem involves multiple sources of information that overlap and supplement each other, being contradictory at occasions. So the key tasks are not only identifying and coping with redundancy across documents, but also recognizing novelty and ensuring that the final summary is both coherent and complete.

Key Phrase Extraction

Keywords and key phrases are attached to documents to give a brief indication of what they are about.

Key phrases are a useful form of metadata because they condense documents into a few pithy phrases that can be interpreted individually and independently of each other. In key phase extraction, all the phrases that occur in the document are listed and information retrieval heuristics are used to select those that seem to characterize it best. Most key phrases are noun phrases, and syntactic techniques may be used to identify these and ensure that the set of candidates contains only noun phrases. The heuristics used for selection range from simple ones such

as the position of the phrase's first occurrence in the document to more complex ones such as the occurrence frequency of the phrase in the document versus its occurrence frequency in a corpus of other documents in the subject area.

Topic Detection and Tracking (TDT)

Topic Detection and Tracking (TDT) [8] refers to a variety of automatic techniques for discovering and threading together topically related material in streams of data such as newswire and broadcast news. The TDT research applications keep track of topics or events of interest, in a constantly expanding collection of multimedia stories. There are five research applications defined in the TDT Program. Story Segmentation detects changes between topically cohesive sections. Topic Tracking keeps track of stories similar to a set of example stories. Topic Detection builds clusters of stories that discuss the same topic. First Story Detection detects if a story is the first story of a new, unknown topic. Link Detection detects whether or not two stories are topically linked. Shared resources, such as TDT corpora, language resources and evaluation software, provide the necessary tools to build a TDT application. The TDT corpora consist of broadcast news and newswire texts sampled daily during most of 1998. The Linguistic Data Consortium (LDC) exhaustively annotated the corpora by identifying which stories discuss a predefined set of topics.

Opinion Mining

Opinion mining [8] refers to the research work of mining user opinion data. It is also known as sentiment analysis. Typical research problems include: (i) subjectivity/objectivity classification is to identify if the text data contains user opinion; (ii) sentiment classification is to predict the polarity of user sentiment (e.g., positive or negative); (iii) opinion summarization is to provide a condensed representation for a set of user opinion texts; (iv) opinion anti-spamming is to detect if the opinion data are written by review spammers.

Key Applications

Bioinformatics

Bioinformatics is the study of the information content and information flow in biological systems and processes [6]. Text mining can be applied to bioinformatics literatures for named entity recognition and relationship extraction. Named entity recognition identifies entities such as drugs, receptors, enzymes, toxins, genes and their features (box, chain, sequence, subunit, etc.). Relation extraction detects the relationship between a pair of entities such as the relationship between genes, protein, or other biological entities.

Email Spam Filtering

The explosive growth of unsolicited emails, more commonly known as spam, has been undermining constantly the usability of emails. One solution is provided by spam filters. Some spam filters use black lists and hand-crafted rules, which are not easy to adapt to new types of spam. On the other hand, the success of machine learning methods in text classification provides the possibility to compete with rule-based filters and quickly adapt to new types of spam. Many spam filters based on machine learning are using Naive-Bayes classifiers. A prominent example is Mozilla's email client. Different classifier methods such as SVM are also used.

Business Intelligence

Business intelligence (BI) is a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions. BI applications include decision support systems, query and reporting, online analytical processing, statistical analysis, forecasting, and data mining. People express their opinions and post reviews of products on the Web. Opinion mining [3] identifies whether users like or dislike a product and products summary from the reviews. Opinion mining can help manufactures to identify problems in their products, and also provides valuable information for placing advertisements in Web pages.

URL to Code

Tools for basic processes of text mining: http://nlp.stanford.edu/links/statnlp.html

Cross-references

► Biomedical Scientific Textual Data Types and Processing
► Data Cleaning
► Data Mining

Recommended Reading

1. Andreas H., Andreas N., and Gerhard P. A brief survey of text mining. J. Computat. Linguistics Lang. Technol., 20(1):19–62, 2005.
2. Bing L. Web Data Mining: Exploring Hyperlinks, Contents and Usage Data. Springer, Berlin, 2007, pp. 411–447.
3. Dipanjan D. and Martins A.F.T. A Survey on Automatic Text Summarization. Literature Survey for the Language and Statistics II course at Carnegie Mellon University, November, 2007.
4. Hearst M. Untangling text data mining. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999.
5. Informative and indicative summarization. Available at: http://www1.cs.columbia.edu/~min/papers/sigirDuc01/node2.html
6. Liebman M. Bioinformatics: an editorial perspective. Available at: (http://www.netsci.org/Science/Bioinform/feature01.html)
7. Usama F., Gregory P.-S., and Padhraic S. From data mining to knowledge discovery in databases. AI Mag., 17(3):37–54, 1996.
8. Wayne C.L. Multilingual topic detection and tracking: successful research enabled by corpora and evaluation. In Proc. Conf. on Language Resources and Evaluation, 2000.
9. Witten I.H. Text mining. In Practical Handbook of Internet Computing, M.P. Singh (eds.). Chapman and Hall/CRC Press, Boca Raton, FL, 2005, pp. 14-1–14-22.

Text Mining of Biological Resources

PADMINI SRINIVASAN
The University of Iowa, Iowa City, IA, USA

Synonyms

Knowledge discovery from biological resources; Hypothesis generation and exploration from biological resources; Literature-based discovery from biological resources

Definition

Text mining is about automatically or semi-automatically exploring hypotheses or new ideas from a set of resources. The mined hypotheses require further tests with methods native to the discipline, in this case with scientific methods in biomedicine. An overall goal in text mining is to support the intellectual activities of biomedical scientists as they explore new ideas using a collection of resources. Text mining is similar to data mining. But instead of mining a collection of well-structured data, text mining operates off semi-structured text collections. Current text mining efforts in biomedicine increasingly involve more structured data sources such as the Entrez Gene database maintained by the National Library of Medicine (NLM).

There is some diversity of opinion on the *kinds* of research that fall within the realm of text mining. As an example, some include text classification, which is about building models to predict one or more topical categories for a text. Still others include information extraction goals such as in research on identifying named entities, definitions and expansions of abbreviations in texts. Some go so far as to include text retrieval, i.e., research on finding documents relevant to a user query. Despite these diverse viewpoints, there is wide agreement that a core focus of text mining is on hypothesis generation and exploration based upon *implicit* connections present in the sources. Consistent with this viewpoint, Blagosklonny and Pardee [1] refer to text mining as conceptual biology, a field that fuels hypothesis-driven biomedical exploration. These hypotheses typically postulate relationships between at least two entities (or more generally concepts). The emphasis is also on novelty at least in the context of the mined sources. Note also that novelty itself eludes definition as it is difficult to pinpoint when and how an idea acquires the status of being *known*.

Historical Background

The motivation underlying biomedical text mining with its focus on hypothesis generation and exploration is that it is difficult for any one scientist or even a group of collaborators to keep abreast with research developments. This difficulty becomes compounded many times given the increasing importance of interdisciplinary perspectives to solve biomedical problems. Another motivation comes from the serendipitous nature of many scientific discoveries. Serendipity may be influenced by several intangibles, such as researcher intuition, prior experience and knowledge, including also the ability to creatively scan and combine the literature of multiple disciplines. With the biomedical literature growing at an astounding pace, there is a definite need for text mining tools to assist bioscientists gauge new ideas against prior research.

Text mining has its origins in the work of Swanson conducted during the mid 1980's. In 1986, Swanson mined the MEDLINE bibliographic database and proposed that fish oils may be used to treat Raynaud's disease [13]. Swanson observed from the literature that Raynauds is exacerbated by platelet aggregability, vaso-constriction, and blood viscosity. He also observed that fish oils reduce these phenomena. Combining these observations he postulated that fish oils may be beneficial for persons with Raynauds which was later corroborated by other scientists. The decade from the mid-1980's to the mid-1990's is marked by a series of papers by Swanson and his collaborator Smalheiser on using text mining to propose a variety of other hypotheses such as connections between estrogen and Alzheimer's disease [10]. These papers created a fertile for text mining research, especially in the context of MEDLINE. Remarkably, it was only in the mid-1990's that other researchers became seriously attracted to biomedical text mining. The first few papers were on automating several of the text mining steps in Swanson and Smalheiser's methodology [5,11]. Since then researchers have proposed other hypotheses such as viruses that may be used as bioweapons [14], possible therapeutic uses for substances such as thalidomide [16] and for turmeric (*Curcumin Longa*) [12], possible functional connections between genes as well as between genes and diseases [9]. Now past the 20 year mark, the text mining field is a fertile ground for research and development. It is mature to the point that several annual workshops affiliated with major conferences have been established. There is also a Journal of Biomedical Discovery and Collaboration that is dedicated to the field and finally there are recent reviews as for example on bio-medical text mining tools [15].

Foundations

Resources

A central aspect to text mining in biomedicine is the use of MEDLINE, the bibliographic database produced by the National Library of Medicine. MEDLINE records contain a variety of fixed and variable length fields such as publication date, title, abstract, authors and author affiliations, chemical terms and subject representations in the form of assigned phrases. Phrases assigned to a record are selected from the MeSH (Medical Subject Headings) controlled vocabu-lary by trained indexers. Text mining systems

sometimes differ in the MEDLINE field(s) used. Some methods use the free-text fields such as title and abstract [5] while others are based on controlled vocabulary fields such as MeSH and chemical terms (e.g., [11]). Systems using controlled vocabularies have the advantage of more precise vocabulary. But they take the risk of missing important information that may be present in the free-text alone. Systems using the free-text fields usually involve procedures for informa-tion extraction to identify key concepts that occur in the texts such as gene, protein, disease and drug name. In this regard research on natural language processing in the biomedical domain has had a huge influence on text mining. This direction is of increasing importance as other forms of textual collections such as patient records are brought into the fold of text mining.

Many text mining approaches also avail of allied vocabulary resources such as the UMLS (Unified Med-ical Language System) also produced by the NLM. The UMLS offers a rich variety of conceptual and linguistic details and it also offers interconnections between the many general and specialized vocabularies arising from different subfields of biomedicine. Other core resources seen, especially in applications targeting bio-informatics, are Entrez Gene [3] and Gene Ontology [4]. Information about annotation links between GO terms and genes frequently accompanied by the MED-LINE records providing supporting evidence, are uti-lized in text mining. Beyond these core resources several others have been used for text mining such as DIP, the Database of Interacting Proteins [2].

Methods

There is a growing variety of text mining methods, orientations and applications. In general, methods ap-pear to be selected or designed to fit the problem at hand. And since the space of text mining problems is broad (and growing), there is an almost bewildering array of text mining solutions. While this situation offers almost free rein to researchers and developers, it also makes it challenging to determine what methods (or aspects about methods) are most successful or most appropriate for a given problem or in a specific domain. Seemingly similar problems are sometimes addressed using significantly different approaches while certain approaches exhibit broader appeal.

Two established text mining approaches derive from Swanson and Smalheiser's early research. These
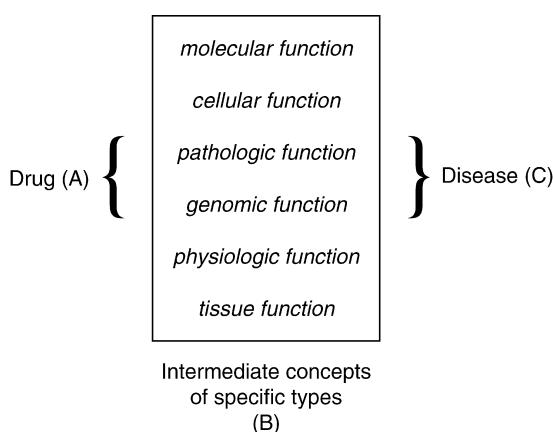
are generally referred to as *open* discovery and *closed* discovery. The open discovery process is initiated with a single concept (A) and the goal is to identify one or more concepts (C) that are connected but only indirectly with A, i.e., through other intermediate concepts. In closed discovery, two concepts A and C are provided as input and the idea is to seek out novel connections between them.

Figure 1 displays these strategies. An example of open discovery represented by the figure is where the starting A concept is a specific drug and the user is interested in novel connections with C concepts representing diseases. Moreover, the intermediate (B) concepts could be constrained to concepts of particular types such as functions of different kinds, molecular, tissue etc. Or, they may be constrained to concepts designating genes already known to be associated with the disease. The novelty aspect is satisfied by ensuring that the A drug and the identified C disease have not yet been studied together. A typical approach implemented to satisfy this requirement is that the A-B concept(s) and B concept(s)-C connections should be found in disjoint portions of the literature collection or that the A-C connection is not recorded in an appropriate knowledge source. For closed discovery an example illustrated by the figure is where the user inputs both a specific drug (A) and a disease (C). The algorithm then looks for new connections, i.e., B concepts that tie the two together. Now it may be the case that no connections between the drug and the disease are known thus far or it may be that some connections are known and other novel ones are being sought. In all of these it can be seen that investigator participation is of value not only to specify the inputs but also to specify the kinds of outputs and to constrain the intermediate connection types of interest. Typically when multiple novel C concepts are discovered through open discovery or multiple B concepts discovered with closed these are ranked by some estimate of confidence.

Many variations of these two basic strategies have been explored. One type of variation extends the transitive nature of these methods to allow for implicit connections ranging over longer distances, i.e., with longer connecting path lengths. For example in G2D [7] researchers start by connecting a disease to its pathological conditions and then to their co-occurring chemical terms in MEDLINE. These are in turn connected through co-occurrence with terms representing protein function from Gene Ontology (GO). These GO terms are then used to identify RefSeq [8] sequences through annotation links. Homology is then used to connect these with candidate sequences that are then constrained to those mapping to the same chromosomal region where the disease is mapped. Another key distinguishing feature across implementations of these text mining algorithms is that confidence estimates may be made in different ways. There is certainly variability in the particular confidence estimation functions used. But more generally, some utilize weights along the single best path while others utilize some function (such as a mean or median) computed over all the connecting paths. Still others, inspired by association rules, adopt the dual notions of confidence and support while several utilize symmetric measures exploiting concept co-occurrence based statistics.

Besides open and closed discovery, there are other text mining methods that rely on exploring graph properties. A graph may be constructed where the nodes represent biomedical objects and the links represent their interconnections. For example, gene networks have been studied by many researchers. Here links between pairs of genes may be directed to indicate influence of one gene over another and weighted by some estimate of degree of influence. Or, these may be undirected with link weight being a function of co-appearance in MEDLINE records. Such graphs may be studied for their structural properties including to identify core groups of objects (here genes).

molecular function

cellular function

pathologic function

genomic function

physiologic function

tissue function

Drug (A) { ... } Disease (C)

Intermediate concepts
of specific types
(B)

Text Mining of Biological Resources. Figure 1. Open and closed discovery.

Unexpected members of such groups may suggest ideas and lead to specific hypotheses and further research.

A recent trend is to embed text mining functions in systems designed with larger scope. iHOP is an example of such a system [6]. In such systems hypothesis generation becomes one of several sub goals. Wide ranging functions are offered by such systems such as easy connections between records of different databases and knowledge sources, literature retrieval and ranking as well as syntactic analysis of the text sentences to extract entities and relationships.

Key Applications

For pointers to current text mining applications the reader may explore sources such as the *Application Notes* sections of Bioinformatics (*Data and Text Mining* subsection); the *Software* and *Database* sections of BMC Bioinformatics; the *Web Server* issue of Nucleic Acid Research and similar sections of other biomedical journals.

Future Directions

Biomedical text mining is at a highly creative age with its scientific basis still in infancy. The area has been heavily influenced by research and development in several fields such as text retrieval, machine learning and computational linguistics. The rapid growth in terms of research papers is a strong indicator of its perceived potential.

There are several open problems in text mining. Certainly the relative merits of alternative methods, their generalizability and criteria for gauging relevance to specific application contexts are still open. There is also a need to obtain a deeper understanding of what is meant by a *novel* idea or hypothesis, as this is a key motivation for text mining research. Methods for evaluating text mining systems is also an open problem. Evaluations tend to be somewhat subjective and non standardized. A common strategy is to see if the algorithms can discover knowledge that is already recorded in sources such as the Database of Interacting Proteins [2]. Another strategy is to obtain the opinion of domain specialists on the connections found. Both strategies have their limitations. Also important is research on how to successfully integrate text mining systems into the work patterns of bioscientists. This will require

a better understanding of research strategies used by bioscientists. Overall it remains to be seen how well text mining will address the needs of the biomedical research community. For the moment, at the very least, it is clear that text mining has significantly extended the frontiers of text based applications in biomedicine.

Cross-references

▶ Data Mining
▶ NLP Techniques for Biomedical Text Analysis and Mining
▶ Text Mining

Recommended Reading

1. Blagosklonny M.V. and Pardee A.B. Unearthing the gems. Nature, 416, 373, 2002.
2. Database of Interacting Proteins: http://dip.doe-mbi.ucla.edu/
3. Entrez Gene: http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene
4. Gene Ontology: http://www.geneontology.org/
5. Gordon M.D. and Lindsay R.K. Toward discovery support systems: A replication, reexamination, and extension of Swansons work on literature-based discovery of a connection between Raynauds and fish oil. J. Am. Soc. Inf. Sci., 47, 116–128, 1996.
6. iHOP: http://www.ihop-net.org/UniPub/iHOP/
7. Perez-Iratxeta C., Bork P., and Andrade M.A. Association of genes to genetically inherited diseases using data mining. Nat. Gene., 31(3):316–319, 2002.
8. RefSeq, http://www.ncbi.nlm.nih.gov/RefSeq/
9. Seki K. and Mostafa J. Discovering implicit associations between genes and hereditary diseases. Pacific Symp. Biocomput., 12:316–327, 2007.
10. Smalheiser N.R. and Swanson D.R. Linking estrogen to Alzheimers disease: an informatics approach. Neurology, 47:809–810, 1996.
11. Srinivasan P. Text mining: generating hypotheses from MEDLINE. J. Am. Soc. Inf. Sci. Technol., 55:396–413, 2004.
12. Srinivasan P. and Libbus B. Mining MEDLINE for Implicit Links between Dietary Substances and Diseases. Bioinformatics, 20 (Suppl 1):I290–I296, August 2004.
13. Swanson D.R. Fish oil, Raynauds syndrome, and undiscovered public knowledge. Persp. Biol. Med., 30:7–18, 1986.
14. Swanson D.R., Smalheiser N.R., and Bookstein A. Information discovery from complementary literatures: categorizing viruses as potential weapons. J. Am. Soc. Inf. Sci. Technol., 52:797–812, 2001.
15. Weeber M., Kors J.A., and Mons B.Online tools to support literature-based discovery in the life sciences. Brief. Bioinform., 6(3):277–286, 2005; doi:10.1093/bib/6.3.277
16. Weeber M., Vos R., Klein H., de Jong-Van den Berg L.T.W., Aronson A., and Molema G. Generating hypotheses by discovering implicit associations in the literature: a case report for new potential therapeutic uses for Thalidomide. J. Am. Med. Inform. Assoc., 10:252–259, 2003.

Text Representation

Jun Yan
Microsoft Research Asia, Haidian, China

Definition

Text representation is one of the fundamental problems in text mining and Information Retrieval (IR). It aims to numerically represent the unstructured text documents to make them mathematically computable. For a given set of text documents $D = \{d_i, i=1, 2,...,n\}$, where each d_i stands for a document, the problem of text representation is to represent each d_i of D as a point s_i in a numerical space S, where the distance/similarity between each pair of points in space S is well defined.

Historical Background

Mining the unstructured text data has attracted much attention of researchers in different areas due to its great industrial and commercial application potentials. A fundamental problem of text mining is how to represent the text documents to make them mathematically computable. Various text representation strategies have been proposed in the past decades for different application purposes such as text categorization, novelty detection and Information Retrieval (IR) [5]. This entry focuses on the text representation strategies specifically for IR applications.

Nowadays, the most commonly used text representation model in the area of Information retrieval is called as the Vector Space Model (VSM) [4,5]. It aims representing each text document by a numerical vector such that the similarity between vectors (documents) can be computed by different kernels. A simple and commonly used kernel is their normalized inner product, which is also known as the Cosine similarity. One of the commonly used VSM is the Bag of Words model (BOW). It uses all words appeared in the given document set D as the index of the document vectors. Under the BOW model, different term weighting schema give different text representation results. The simplest case of BOW is the Boolean model. It utilizes the binary vectors to represent text documents. In other words, if a term appears in a document, there has a "1" in the position, which corresponds to this term, in the document vector.

Otherwise, the term weight is "0". As an extension of the Boolean model, Term Frequency Inversed Document Frequency (TFIDF) model was proposed. It uses real values which capture the term distribution among documents to weight terms in each document vector. However, there are many limitations in the traditional BOW text representation model. For example, (i) BOW ignores the within document term correlation such as the order of terms in a given document; (ii) the polysemy and synonymy problems can greatly decrease the IR performance in the TFIDF text representation model; and (iii) the TFIDF model cannot capture the semantics of documents for IR.

To solve the limitations of BOW model, various advanced text representation strategies have been proposed. The N-gram statistical language models [2] were proposed to capture the term correlation within document. However, the exponentially increasing data dimension with the increase of N limits the application of N-gram models. The Latent Semantic Indexing (LSI) [3] was proposed to reduce the polysemy and synonym problems. At the same time, LSI can also represent the semantics of text documents through the linear combination of terms, which is computed by the Singular Value Decomposition (SVD). However, the high complexity of SVD [1] make LSI seldom used in real IR tasks. In addition, some external resources such as the Wordnet and Wikipedia are recently used for solving the polysemy and synonym problems in text representation. Since the effectiveness of these external resources for text representation can only be learned in research papers and there still has no evidence to show their power in real IR applications, this article will not introduce their details. Motivated by the LSI, the Probabilistic Latent Semantic Indexing (PLSI) [6] is also proposed for representing the semantics of text documents. However, it is still limited by the computation complexity due to the increasing scale of Web data. As a summary, though various approaches have been proposed for solving the limitations of BOW, the BOW with TFIDF term weighting schema is still one of the most commonly used text representation strategies in real IR applications.

Beyond VSM, many propose to represent text documents in other formats instead of vectors or represent text documents through their meta-information. For instance, the text documents can be represented through neural network, through graphs and in tensor

space model. The text documents in Web pages can be semantically represented by the search queries which have clicked these pages, the social bookmarks which have annotated these pages. However, most of them are used for solving specific text mining problems or used to enhance performance of some special IR tasks. In the next Section of this article, the details for text representation in two parts will be given, which are the traditional BOW text representation model with TFIDF term weighting schema and the semantic text representation through LSI.

Foundations

In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection $D=\{d_i, i=1, 2,...,n\}$, suppose there are m unique terms appeared in this collection (The stop words removal and stemming will be introduced later). Mathematically, this corpus of documents can be represented by a m by n matrix $S \in R^{m \times n}$. Each text document is denoted by a column vector $s_i, i = 1, 2,...,n$ and each term is denoted by a row vector. The jth entry of s_i is denoted by $s_{ji}, j = 1, 2,...,m$. As an example, suppose the document collection D implies two documents,

d_1: He investigates the text representation approaches.
d_2: What is the meaning of text representation approach for text documents?

There are a list of 13 unique terms, which are,

"He, investigates, the, text, representation, approaches, What, is, meaning, of, approach, for, documents".

The list of terms roughly represents the two documents by a 13 by 2 matrix. The problem is how to weight each entry of this matrix. Considering the simple Boolean model first. If a term appears in a document, its corresponding weight is 1; otherwise, it is 0. The transform of the matrix for the collection D is,

$$S^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

where the order of term index is the same as the term list given above, i.e., the first term is "He" and the last term is "documents." There are three obvious problems in this text representation strategy: (i) not all terms have the physical meaning in representing text documents such as "the," "is" etc; (ii) some terms such as "approach" and "approaches" are actually the same;

and (iii) the importance of all the terms is treated as the same in this strategy. Can the difference of the term importance be reflected in text representation? The answers for these three questions correspond to three key steps in text representation, which are *stop words removal*, *stemming* and *TFIDF indexing*.

The stop words (or stopwords) is the name of the terms that should be filtered out before text documents indexing or natural language processing. There has no fixed stop words list for all text processing applications. Generally the stop words list will include the terms like "a," "the," "an," etc. After removing the stop words, a stemming procedure is applied before the TFIDF indexing. The stemming aims at reducing inflected words to their stem. For example, "approaches" is stemmed to "approach," "investigates" is stemmed to "investigate" and "representation" is stemmed to "represent." Thus in the above example, the list of terms is reduced to,

"He, investigate, text, represent, approach, what, mean, document."

Thus the two documents d_1 and d_2: can be represented by an 8 by 2 matrix. The transpose of it is,

$$S^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Until now, the problems (i) and (ii) have been solved. The last problem is how to tell the importance of different terms in text representation. One of the commonly used approaches is called as the TFIDF indexing. The TFIDF aims to assign a weight to each term according to a document in a collection or corpus. In other words, TFIDF aims to assign different weights for all entries s_{ji} in matrix S. An intuition is that the more times a term appears in a document, the more important is this term to this document. Thus the weight should increase proportionally to the number of times a term appears in the document. On the other hand, if a word appears in many documents in the corpus, the discriminative power of the term will be weak. Thus the weight is offset by the frequency of the word in the corpus. The former step is called the Term Frequency (TF). It can be counted directly from the documents. As an example, the TF of term stext" in d_2 is 2. A normalizing factor is always used for calculating the TF of a term in a document. Since in d_2 there are 7 terms after stop words removal and stemming. Among

them, 2 of them are "text." Thus the TF for term "text" in d_2 is 2/7. The latter step is called the Inverse Document Frequency (IDF). It is a log function. The IDF for term t is,

$$\log \frac{n}{\#\text{document involve term } t}$$

Thus the TFIDF weighting schema can be as simple as TF*IDF. There are various variations of TFIDF text indexing. The major differences are how to normalize and smooth the weighting equation.

In the BOW model, the TFIDF indexing gives a way to weight the terms for text documents. However, this kind of term frequency based approaches cannot discover the semantics of the text documents. In the following several paragraphs, the Latent Semantic Indexing (LSI) is briefly introduced, which aims to discover the text semantics through linear combination of term weights. As introduced above, the term by document matrix is a sparse matrix whose rows correspond to terms and columns correspond to documents. LSI aims to transform the matrix S into a reduced matrix which can reflect the relation between the documents and some *concepts*. Thus the terms and documents are indirectly related through the concepts.

Mathematically, LSI aims to find a projection matrix $W \in R^{m \times p}$ such that the linear projection $y_i = W^T s_i \in R^p, i = 1, 2, ..., n$ can reflect the p semantic concepts implied by document d_i, where $p << m$. In other words, LSI assumes that the linear combination of terms can reflect the concepts in text documents. $W \in R^{m \times p}$ can give p different linear combinations for term weights vector s_i. Through the linear projection matrix W, the text document d_i which is represented by m terms' weights in vector s_i is represented by p concepts' weights in vector y_i. The problem left is how to get the projection matrix W, i.e., how to get the weights for the linear combination of terms, from the matrix S. The calculating of W can be formulated from different perspectives. For example, it can be formulated as optimization problem through the essential relationship between LSI and Principal Component Analysis (PCA). It can also be formulated as the best low rank matrix approximation problem. Intuitively, LSI is computed through the matrix decomposition. Given the term by document matrix S, where s_{ji} stands for the weight of term j in document i. Assume that there exists a

decomposition of matrix $S = U \Sigma V^T$, where U and V are orthogonal matrices and S is a diagonal matrix. This matrix decomposition is called as the Singular Value Decomposition (SVD). If preserve only the first p columns of U, it is the projection matrix $W \in R^{m \times p}$ for LSI. For details for LSI please refer to [3].

Key Applications

The text representation is the fundamental work for IR and text mining. Besides IR, it can also be used for text categorization, text clustering, topic detection and novelty detection etc.

Future Directions

The future directions of text representation problem can be roughly classified into two categories. The first is the large scale computation and the second is the semantic text representation. For the former, the goal is to make the text representation strategies which have high cost to be usable in real IR tasks. For example, one can develop the distributed infrastructure for N-gram model which can be used to enrich the BOW model if the computation of N-gram model is efficient enough. One can develop the incremental approximation or distributed computation algorithms for large scale LSI which can discover the semantic of documents in large scale IR systems. For the latter, besides LSI and PLSI, some semantic Web related research works give good information sources about how to discover the semantics of text documents.

Data Sets

For testing the effectiveness of text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578 (http://www.daviddlewis.com/resources/testcollections/reuters21578/), RCV1 (http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lewis04a.pdf), 20 Newsgroup (http://people.csail.mit.edu/jrennie/20Newsgroups/), Open Directory Project (ODP) (http://rdf.dmoz.org/) and the TREC text datasets (http://trec.nist.gov/data.html) are all commonly used for text mining or IR research.

Cross-references

► Stemming Algorithms
► Term Statistics
► Term Weighting
► Text Analytics

Recommended Reading

1. Alter O., Brown PO., and Botstein D. Singular value decomposition for genome-wide expression data processing and modeling. In Proc. Natl. Acad. Sci. USA., 97:10101–10106.
2. Daniel J. and James H.M. Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Processing. Prentice-Hall, Englewood Cliffs, NJ, 2000.
3. Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., and Harshman R.A. Indexing by latent semantic analysis. J. Soc. Inf. Sci., 41(6):391–407.
4. Gerard S.A. Theory of Indexing. Society for Industrial Mathematics, Philadelphia, PA, 1987.
5. Gerard S. and Michael J. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.
6. Thomas H. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.

Text Retrieval

► Information Retrieval

Text Segmentation

HAODA HUANG, BENYU ZHANG
Microsoft Research Asia, Beijing, China

Synonyms
Document segmentation

Definition
Text segmentation is a precursor to text retrieval, automatic summarization, information retrieval (IR); language modeling (LM) and natural language processing (NLP). In written texts, text segmentation is the process of identifying the boundaries between words, phrases, or some other linguistic meaningful units, such as sentences or topics. The term separated from such processing is useful to help humans reading texts, and are mainly used to assist computers to do some artificial processes as fundamental units, such as NLP, and IR.

Historical Background
Natural language processing (NLP) is an important research field. Its primary problem is how to segment text correctly. Various segmentation methods have emerged in the past decades for different kinds of language and applications. Text segmentation is language dependent (different language has its own special problems, which would be introduced later), corpus dependent, character-set dependent, and application dependent. Most existing text segmentation systems are language specific and corpus dependent.

This entry mainly focuses on traditional English and some Chinese text segmentation strategies. To do such segmentations, there are two major approaches: (i) Manually analyzing the characters of text to get some heuristic approaches; (ii) Doing annotations for the sample corpus with boundary information, then adopting some machine learning (ML) methods to learn from annotated corpus, and finally doing automatic text segmentation.

The problem of word segmentation (also known as tokenization), which is the process of dividing the sequence of characters into words by locating words boundaries, does not appear to be difficult for written languages that have explicit word boundary markers, such as English in which words are separated by white spaces. When such clues are not consistently retained in written languages, such as Chinese, in which sentences can be considered to be a character string, doing word segmentation should be the most important and essential part of text segmentation. On the contrary, sentence segmentation (which also can be referred to as sentence boundary detection, sentence boundary disambiguation (SBD), or sentence boundary recognition) is more difficult in English text segmentation, which must disambiguate punctuations that might denote sentence boundaries.

Based on previous analysis, for English text segmentation, words can be isolated by white spaces. So, the main problem of this kind of segmentation is how to recognize the boundaries of sentences, which involves resolving the use of ambiguous punctuation, such as periods, commas, and semicolons. Some recognizable tokens contain ambiguous punctuation,

such as numbers (e.g., 1,236.78), dates (e.g., 02/05/96), acronyms (e.g., AT&T), and abbreviations (e.g., U.S.). Some example sentences containing ambiguous punctuations follow:

1. *Clairson International Corp. said it expects to report a net loss for its second quarter ended March 26 and doesn't expect to meet analysts' profit estimates of $3.0 to $4 million, or 1,276 cents a share to 1,279 cents a share, for its year ending Sept. 24.* (From the Wall Street Journal (1988))
2. *The contemporary viewer may simply ogle the vast wooded vistas rising up from Saguenay River and Lac St. Jean, standing in for the St. Lawrence River.* (From the Wall Street Journal (1991))
3. *The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.* (From the Wall Street Journal (1987))

As can be seen, periods have been used three different ways in the first example - within a decimal *($3.0)*, in abbreviations (*Corp.* and *Sept.*), and at the end of the sentence. A comma is used in a number (*1,276 and 1,279*). Consider the second and third examples in which "*St.*" appears three times. The first two instances of "*St.*" do not denote the boundary of the sentence, whereas the last one delimits the sentence. At the same time, it is also essential to do word segmentation in English text segmentation. It is necessary to recognize whether a period within a number *($3.0)* should be a part of decimal or the end of a sentence – that is, should *$3.0* be parsed as a word "*/$3.0/*" or be parsed as "*/$3/. /0/*". The comma used in the number (*1,276 and 1,279*) also should be recognized as a part of number rather than a break of a sentence – that is, whether "*1,276*" should be broken to "*/1,276/*" rather than "*/1/, /276/*". Taking the third and first example sentences above, the first two "*St.*" should be segmented to "*/St./*". Though the last "*St.*" is also supposed to be segmented to "*/St./*", one must always realize that this period also stands for the end of sentence.

If there were a dictionary containing all kinds of words and including all the acronyms and abbreviations, it would be much easier to recognize words and do tokenization. In actuality, however, acronyms and abbreviations are emerging continuously and will never stop. In order to solve this problem, many approaches have been proposed. There are mainly two approaches: First, manual analysis often

uses regular expression grammars and lexicon to solve those problems. The systems based on such methods are always called rule-based systems. In such systems, Lexicon is used to obtain the information of words, such as capitalization, spelling, and suffixes, to determine whether a word followed with a period should be a common word or a boundary signal of a sentence. Regular expressions are established for detecting special kinds of words, such as numbers and dates. Another way of performing sentence boundary disambiguation (SBD) is to use machine learning (ML) techniques, such as decision tree classifier, maximum entropy modeling, and neural network. The most essential information for Machine Learning systems to obtain are good features, such as word spelling, suffix, capitalization, and word classes. The difficulty is how to obtain a well annotated sample corpus with labels of word boundaries and sentence boundaries.

For Chinese text segmentation, sentences are always isolated by non-ambiguous punctuations, but words are character strings without explicit boundary markers. So, the main problem of Chinese text segmentation is how to recognize the boundaries of words. All the same, Word Sense Disambiguation is the primary issue in word segmentation. Three types of ambiguity string are present in Chinese text segmentation: overlapping ambiguity string, combinatorial ambiguity string, and hybrid ambiguity string. To solve those problems, various approaches have been proposed. The most typical solution is using forwards maximum matching (FMM) and backwards maximum matching (BMM) based on a dictionary.

Except for these two typical languages, there are still other types of languages with many different features of written text. Many more difficulties for each language have appeared.

In addition to language-dependent word segmentation and sentence segmentation, there is another segmentation that has emerged: topic segmentation, since a document always contains several topics and even one topic may have different aspects. Topic segmentation is much harder than word and sentence segmentation due to difficulties in detecting semantic units and dividing them into topics. Many approaches have been tried to deal with such problems [1,4]. Applications based on topic segmentation are

abundant, such as document summarization, text processing, and NLP.

Foundations

Using regular expression grammars and lexicon to solve ambiguous punctuation is classic and to some extend useful. Different regular expressions have been defined for all kinds of tokens containing ambiguous punctuation. First, lexicon is used to find if a token exists in the dictionary or not. If it does not exist, a regular expression is used.

1. Numbers. Take "123,456.789%" as an example. The regular expression should be ((0–9]+,)*[0–9] +(.(0–9]+)*%? (ab means matching a, and then matching b.
   a + means one or more a would be matched.
   a* means zero or more a would be matched.
   a? means zero or one a would be matched.). Once this expression is used to recognize numbers, periods, commas and percent within a number, those ambiguous punctuations will not be parsed falsely as a boundary of a sentence.
2. Dates, such as "05/11/95". Heuristic regular expression would be [0–9][0–9]/[0–9][0–9]/[0–9][0–9].
3. Acronyms and Abbreviations, typically "U.S., St., Ash., and Corp." Regular Expression [A–Z] +[A–Za-z0–9]*.([A–Za–z0–9].)* will be used to find those acronyms and abbreviations. It should be noted that when an acronym or abbreviation is located at the end of a sentence, after the expression mentioned above has been used to parse such kind of sentences, an error would occur. For example, the sentence "*The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.*" would be parsed to one sentence, although it should actually be split into two sentences "/*The firm said it plans to sublease its current headquarters at 55 Water St./*" and "/*A spokesman declined to elaborate./*".

Three typical kinds of tokens have been presented above. There are still many other types of words containing ambiguous punctuations. Many more regular expressions are required to deal with all of them. The set of expressions would be extremely huge. It is important to mention that these typical kinds of tokens are language dependent. This implies that different regular expression sets for each language would need to be established, which is a labor consuming enterprise. Also, the expressions are usually corpus related, which makes those rules unable to be ported across domains.

Because there are so many limitations in using regular expressions to segment words and sentences, as mentioned above, machine learning (ML) techniques have been taken into this field. These techniques can be retrained quickly for a new domain, a new corpus, a new language only if a well-annotated sample corpus has been given. Some systems also use part-of-speech (POS) information to improve performance [3].

In order to discuss how to deal with Chinese text segmentation, first, three kinds of ambiguity string should be introduced. (i) Overlapping Ambiguity String. It is defined as follows: ABC can be segmented into A/BC and AB/C, such as "和/平等" could be segmented into"和/平等" and "和平等" in different situations. In sentence "独立/自主/和/平等/独立/的/原则" and "讨论/战争/与/和平/等/问题", the segmentation of "和平等" is different. (ii) Combinatorial Ambiguity String. It is defined as follows: AB can be segmented into AB and A/B, such as "马上" should be segmented into "马上" in sentence "马上/过来" and should be "马/上" in sentence "他/骑/在/马/上". (iii) Hybrid Ambiguity String. It combines overlapping and combinatorial ambiguity. To figure out the first kind of ambiguity string problem, forwards maximum matching (FMM) and backwards maximum matching (BMM) are used to segment sentence separately. Such as sentence "独立/自主/和/平等/独立/的/原则", with FMM, it should be segmented into "独立/自主/和平/等/独立/的/原则"; with BMM, it should be segmented into "独立/自主/和/平等/独立/的/原则". When the result of segmentation with FMM and BMM is different, syntax, semantic, pragmatic information will be added to determine which segmentation should be chosen. To solve the combinatorial ambiguity string problem, FMM and Backwards Minimum Matching are adopted. For example with FMM, the sentence "他骑在马上" should be segmented into "他/骑/在/马上"; with BMM, it should be segmented into "他/骑/在/马/上". The following processing is the same as overlapping ambiguity string segmentation.

Key Applications

Text segmentation is an important aspect in developing text processing applications, such as Information Extraction, Document Summarization, Machine Translation (MT), Natural Language Processing,

Information Retrieval, Language Modeling, and Speech Recognition.

Future Directions

There are roughly two directions for text segmentation in the future. One is text segmentation based on specific language. Another is its use in other fields, such as the rich transcription field. For the former, there are more than 200 languages in the world. Each has its own difficulties in doing text segmentation, such as word segmentation in Chinese text segmentation. These problems are do not have ideal solutions. For the latter, sentence boundary disambiguation (SBD) has attracted increased attention recently as a way to improve speech recognition output for better readability and downstream natural language processing and some subsequent tasks, such as speech translation and speech summarization.

Data Sets

Two main data sets are typically used for testing, evaluation and development in large amount of text segmentation and text processing tasks: Brown Corpus and the Wall Street Journal (WSJ) corpus – containing the Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993). Texts in those corpora are all split into documents, paragraphs, and sentences and are annotated with POS information. The above information is necessary to develop and evaluate text segmentation systems.

Other data sets on Rich Transcription can be used to perform sentence boundary disambiguation, such as the Rich Transcription data sets provided by National Institute of Standards and Technology (NIST).

Cross-references
► Column Segmentation
► Text Retrieval
► Text Summarization
► Text Representation

Recommended Reading

1. Beeferman D., Berger A., and Lafferty J. Statistical models for text segmentation. Mach. Learn., 34(1–3):177–210, 1999.
2. Grefenstette G. and Tapanainen P. What is a word, what is a sentence? Problems of tokenization. In Proc. 3rd Conf. on Computational Lexicography and Text Research, 1994, pp. 7–10.
3. Mikheev A. Tagging sentence boundaries. In Proc. 1st Conf. on North American Chapter of the Association for Computational Linguistics, 2000, pp. 264–271.
4. Reynar J.C. and Marcus M.P. Topic segmentation: algorithms and applications. Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA, 1998.

Text Semantic Representation

Jun Yan, Jian Hu
Microsoft Research Asia, Haidian, China

Definition

The classical text representation strategies aim to numerically represent the unstructured text documents to make them mathematically computable. With the rapid growth of information retrieval and text data mining research, the semantic text representation is attracting more and more attention. The problem is how to represent the text documents by explicit or implicit semantics instead of word occurrence in the document. The goals of semantic text representation are to improve the text clustering, classification, information retrieval and other text mining problems' performance.

Historical Background

In the past decades, semantic text representation has attracted much attention in the area of information retrieval and text data mining research. There have different ways for categorizing various semantic text representation strategies. This entry generally classifies the previous efforts for this problem into two categories: explicit semantic text representation and implicit semantic text representation.

The explicit semantic text representation aims to represent text documents by explicit readable sentences, key phrases or keywords, which can semantically describe the main topic of the given text documents. The related approaches can be further classified into automatic approaches and manual approaches. From the automatic approaches' perspective, the Text Summarization technologies aim at learning one or more sentences to represent a given text document; the Information Extraction technologies aim at extracting one or more key phrases for describing the semantic of a given text document. In addition, many previous works propose to annotate the Web pages, which is a special type of text documents, by search engine click through

logs. With the rapid growth of Semantic Web (http://infomesh.net/2001/swintro/#furtherReading) in recent years, the automatic web page annotation and ontology learning, etc can all be utilized for automatically represent Web pages semantically. As some examples, the WordNet and Wikipedia have both been utilized for semantically enhance the text representation for various applications. From the manual approaches' perspective, there are many commercial systems target at semantic text representation by key phrases or key words. Two of the most representative examples of these commercial systems are delicious and flicker. Both of them aim to semantically represent the Web pages through manually assigned social annotation to a large number of Web pages.

To date, the work on integrating semantic background knowledge into text representation is quite few and the results are not good enough. Buenaga Rodriguez et al. and Urena Loez et al. successfully integrated the WordNet resource for a document categorization task. They improved classification results of Rocchio and Widrow-Hoff algorithms on Reuters corpus. In contrast some work utilized WordNet in a supervised scenario without employing WordNet relations such as hypernyms and associative relations. Meanwhile, they built the term vectors manually. Dave et al. has utilized WordNet synsets as features for document representation and subsequent clustering. That work did not perform word sense disambiguation and found that WordNet synsets decreased clustering performance in the experiments. Hotho et al. integrated WordNet knowledge into text clustering, and investigated word sense disambiguation strategies and feature weighting schema through considering the hypernym relations from WordNet. The experimental results on Reuters corpus show improvements compared with the best baseline. However, considering the few word usage contexts provided by WordNet, the word sense disambiguation effect is quite limited. Meanwhile, the enrichment strategy which appends or replaces document terms with their hypernym and synonym is overly simple. To solve the limitations of many previous work, Hu et al. proposed to enhance the text representation by Wikipedia.

On the other side, one of the most classical implicit semantic text representation approaches is known as the Latent Semantic Indexing (LSI) [2]. Nowadays, the most commonly used text representation model is called as the Vector Space Model (VSM) [3]. It aims to represent each text document by a numerical vector such that the similarity between vectors (documents) can be computed by different kernels. Latent Semantic Indexing (LSI) was originally proposed for dealing with the problem of *synonymy* and *polysemy* in the vector space model. In VSM, LSI represents the semantics of text documents through the linear combination of terms, which is computed by the Singular Value Decomposition (SVD) [1]. The reason why LSI is known as the implicit semantic text representation strategy is that the linear combination of keywords, which are semantics of the text documents, cannot be explained intuitively. A variety of tests and applications have been developed to validate its power in text representation. Special interests have been paid to investigate its ability in improving the IR performance. Besides general IR tasks, LSI has also been successfully applied for the cross-language retrieval and distributed information retrieval tasks. However, classical LSI suffers from the high computational cost involved in the Singular Value Decomposition (SVD), especially when applied to large scale text corpus. To avoid the costly computation, it has been proposed to use other strategies such as Semi-Discrete matrix Decomposition (SDD) and Concept Indexing (CI) instead of LSI for implicitly and semantically representing text documents. There are many variances of the classical LSI, the Probabilistic Latent Semantic Indexing (PLSI) [4] and the Supervised LSI are some of the examples.

Foundations

This entry mainly introduces one classical algorithm in each algorithm category. In the explicit semantic text representation category, the Wikipedia for text enrichment is introduced. On the other hand, in the implicit semantic text representation category, the traditional latent semantic indexing is introduced.

Wikipedia is a dynamic and fast growing resource – articles about newsworthy events are often added within few days of their occurrence. Each article in Wikipedia describes a single topic; its title is a succinct, well-formed phrase that resembles a term in a conventional thesaurus. Meanwhile, each article must belong to at least one category of Wikipedia. Hyperlinks between articles keep many of the same semantic relations as defined in international standard for thesauri, such as equivalence relation (synonymy), hierarchical relation (hypernym) and associative relation. However, as an open resource, it inevitable includes much noise. To make it a clean

and easy-to-use as a thesaurus, a recent work proposed by Hu et al. first preprocess the Wikipedia data to collect Wikipedia concepts, and then explicitly derive relationships between Wikipedia based on the structural knowledge of Wikipedia.

Each title of a Wikipedia article describes a topic, and they are denoted as a concept. After process the Synonymy, Polysemy and Hypernymy, since each Wikipedia article contains a lot of hyperlinks, which express relatedness between them, The cosine similarity of article pairs in Wikipedia may reflect the relatedness between the two concepts. However the drawback of this measurement is the same as that of *BOW* approach, since it only considers terms appeared in text documents which have no semantic information. Another method to measure the relatedness between a pair of Wikipedia articles is to compare the similarity between outlinked categories of the two articles. It can be observed that if two articles share some out-linked categories, the concepts described in these two articles are most likely related. To get an overall relatedness of two Wikipedia concepts, the above two measures are linearly combined.

As introduced above, to represent text documents semantically, many previous approaches enriched text representation with external resources such as WordNet and ODP. The same to them, for the Wikipedia, first, the algorithm generates new features for each document in the dataset. The features can be synonym or hypernym for document terms or expanded features for terms, sentences and documents. Second, the generated new features replace or append to original document representation and construct new vector representation.

On the other hand, the classical LSI due to its effectiveness in the area of text data mining and information retrieval research. More details about other related approaches please refer to the recommended readings. In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection $D = \{d_i, i = 1, 2,...,n\}$, suppose there are m unique terms appeared in this collection. Mathematically, this corpus of documents can be represented by a m by n matrix. $S \in R^{m \times n}$ Each text document is denoted by a column vector $s_i, i = 1, 2,...,n$ and each term is denoted by a row vector. The jth entry of s_i is denoted by $s_{ji}, j = 1, 2,...,m$.

Mathematically, LSI aims to find a projection matrix $W \in R^{m \times p}$ such that the linear projection

$y_i = W^T s_i \in R^p, i = 1, 2,...,n$ can reflect the p semantic concepts implied by document d_i, where $p << m$. In other words, LSI assumes that the linear combination of terms can reflect the concepts in text documents. $W \in R^{m \times p}$ can give p different linear combinations for term weights vector s_i. Through the linear projection matrix W, the text document d_i which is represented by m terms?' weights in vector s_i is represented by p concepts?' weights in vector y_i. The problem left is how to get the projection matrix W, i.e., how to get the weights for the linear combination of terms, from the matrix S. The calculating of W can be formulated from different perspectives. For example, it can be formulated as optimization problem through the essential relationship between LSI and Principal Component Analysis (PCA). It can also be formulated as the best low rank matrix approximation problem. Intuitively, LSI is computed through the matrix decomposition. Given the term by document matrix S, where s_{ji} stands for the weight of term j in document i. Assume that there exists a decomposition of matrix $S = U \Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix. This matrix decomposition is called as the Singular Value Decomposition (SVD). If preserve only the first p columns of U, it is the projection matrix $W \in R^{m \times p}$ for LSI.

Key Applications

The semantic text representation has various different applications. Generally speaking, it is very important for the traditional text clustering, text categorization and information retrieval tasks. As some detailed examples, the LSI is generally used for relevance based information retrieval and text clustering. Documents summarization can be used for search results snippet generation. Key word extraction can be used as a component of the ontology learning of semantic Web. The social annotation is a fundamental work for semantic Web. As a summary, the semantic text representation is a fundamental problem for text mining and analysis.

Future Directions

The future directions of semantic text representation problem can be roughly classified into threefold. The first is how to develop novel semantic text representation approaches by designing novel algorithms and leveraging other meta-information. In more details,

the Wordnet was first proposed to enhance the semantic text representation. Simultaneously, the search engine click-through log was proposed to enhance the text representation. Recently, the Wikipedia and social annotation were widely studied as the meta-information of text documents for semantic text representation. The remaining problem is whether other better data sources for text enrichment will be found. The second is the scalability issue for current strategies. For example, the SVD computation for LSI is highly expensive. A big problem is how to design scalable algorithm for LSI to deal with large scale data. The advanced algorithm could be approximated LSI, parallel computation and incremental computation etc. The third direction is how to apply the semantic text representation strategies for pushing the progress of semantic Web and other applications of semantic text representation. For example, one problem is how to enhance the text clustering or classification by semantic text representation.

Data Sets

For testing the effectiveness of semantic text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578 (http://www.daviddlewis.com/resources/testcollections/reuters21578/), RCV1 (http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lewis04a.pdf), 20 Newsgroup (http://people.csail.mit.edu/jrennie/20News,groups/), Open Directory Project (ODP) (http://rdf.dmoz.org/) and the TREC text datasets (http://trec.nist.gov/data.html) are all commonly used for text mining or IR research.

Cross-references

▶ Semantic Web
▶ Text Categorization
▶ Text Retrieval

Recommended Reading

1. Alter O., Brown PO., and Botstein D. Singular value decomposition for genome-wide expression data processing and modeling. In Proc. Natl. Acad. Sci. USA., 97:10101–10106.
2. Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., and Harshman R.A. Indexing by latent semantic analysis. J. Soc. Inf. Sci., 41(6):391–407.
3. Gerard S. and Michael J. Introduction to Modern Information Retrieval. McGraw-Hill Companies, 1983.
4. Thomas H. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.

Text Streaming Model

Ning Liu
Microsoft Research Asia, Beijing, China

Definition

Text streaming model (TSM) is one of the fundamental problems in streaming model and text mining. It aims to process a sequence of text data that comes at a rate. In this model, text data does not take the form of arriving in multiple, continuous, rapid, time-varying data streams. Input text streaming D_1, D_2, ... arrives sequentially, one by one, and the aim of TSM is to analysis the streaming text data.

Historical Background

Mining the streaming data has attracted much attention of researchers in different areas due to its great industrial and commercial application potentials. Specifically, text streaming data models are of interest to the machine learning and data mining community. The world wide web has many text data, such as web pages, news-feeds, emails and blogs. And most of them are classical text streaming data. Then, how to model text streaming data is important.

Traditional text data model is Vector Space Model (VSM). One of the commonly used VSM is the Bag of Words model (BOW), which index the document as a set of terms. This set of terms defines a space such that each distinct term represents the entries in that space. Since VSM represents the documents as a set of terms, this space can be viewed as a "document space." A numeric weight can then be assigned to each term in a given document, representing an estimate of the usefulness of the given term as a descriptor of the given documents. The weights assigned to the terms in a given documents can then be interpreted as the coordinates of the document in the documents space. Then, Term Frequency Inversed Document Frequency (TFIDF) model was proposed. It uses real values, which capture the term distribution among documents to weight terms in each document vector. Moreover, the N-gram statistical language model was proposed to model text data with VSM.

However, there are two challenges in the traditional VSM for text streaming data.

1. Update the data automatically
2. Do analyses of updated streaming data in acceptable time

Foundations

In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection $D = \{d_i, i = 1, 2,...,n\}$, suppose there are m unique terms appeared in this collection (stop words removal and stemming are introduced later). Mathematically, this corpus of documents can be represented by a m by n matrix $S \in R^{m \times n}$. Each text document is denoted by a column vector $s_i, i = 1, 2,...,n$ and each term is denoted by a row vector. The jth entry of s_i is denoted by $s_{ji}, j = 1, 2,...,m$. As an example, suppose the document collection D implies two documents,

– d_1: He investigates the text representation approaches.
– d_2: What is the meaning of text representation approach for text documents?

There is a list of 13 unique terms, which are,

▶ "*He, investigates, the, text, representation, approaches, What, is, meaning, of, approach, for, documents.*"

Thus, one roughly represents the two documents by a 13 by 2 matrix. Consider the simple Boolean model first. In other words, if a term appears in a document, its corresponding weight is 1; otherwise, it is 0. The transform of the matrix for collection D is,

$$S_2^T = \begin{pmatrix} 1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1 \end{pmatrix}$$

where the order of term index is the same as the term list given above, i.e., the first term is "He" and the last term is "documents."

For text streaming data, a new text is coming,

– d_3: Text mining is important in IR.

Now, the term list of d_1 and d_2 should be changed. The problem is how the model streaming text data, which is how to update the indexing matrix.

$$S_s^T = \begin{bmatrix} S_2^T & 0 & 0 \\ 0 & s_{3i} & 0 \end{bmatrix}$$

Key Applications

The streaming text model is the fundamental work for IR and text mining. Besides IR, it can also be used for news categorization and RSS clustering etc.

Data Sets

For testing the effectiveness of text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578, RCV1, 20 Newsgroup, Open Directory Project (ODP) and the TREC text datasets are all commonly used for streaming text model.

Recommended Reading

1. Abadi D., Carney D., Çetintemel U., Cherniack M., Convey C., Erwin C., Galvez E., Hatoun M., Maskey A., Rasin A., Singer A., Stonebraker M., Tatbul N., Xing Y., Yan R., and Zdonik S. Aurora: a data stream management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 666.
2. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.

Text Summarization

DOU SHEN
Microsoft Corporation, Redmond, WA, USA

Synonyms

Document summarization

Definition

Text summarization is the process of distilling the most important information from a text to produce an abridged version for a particular task and user [9].

Historical Background

With more and more digitalized text being available, especially with the development of the Internet, people are being overwhelmed with data. How to help people effectively and efficiently capture the information from the data becomes extremely important. Many techniques have been proposed for this goal and text summarization is one of them.

Text summarization in some form has been in existence since the 1950s [8]. Two main influences have dominated the research in this area, as summarized by Mani in [10]. Work in library science, office automation, and information retrieval has resulted in a focus on methods for producing extracts from scientific papers, including the use of "shallow" linguistic analysis and the use of term statistics. The other

influence has been research in artificial intelligence, which has explored "deeper" knowledge-based methods for condensing information. While there are a number of problems remaining to be solved, the field has seen quite a lot of progress, especially in the last decade, on extraction-based methods. This progress has been greatly accelerated by the rather spectacular advances in shallow natural language processing, and the use of machine learning methods which train summarization systems from text corpora consisting of source documents and their summaries. In the following sections, a brief introduction of different kinds of text summarization will be introduced, which is followed by an overview of the existing algorithms.

Foundations

Text summarization can be categorized along two different dimensions: abstract-based and extract-based. An extract-summary consists of sentences extracted from the document while an abstract-summary may employ words and phrases that do not appear in the original document but that are semantically meaningful [9]. The summarization task can also be categorized as either generic or query-oriented. A query-oriented summary presents the information that is most relevant to the given queries, while a generic summary gives an overall sense of the documents content [4]. In addition to single document summarization, which has been first studied in this field for years, researchers have started to work on multi-document summarization whose goal is to generate a summary from multiple documents that cover related information.

Most current text summarizers are extractive, since extraction is widely used, such as the snippets generated by search engines, while it is much easier than abstracting. Therefore, this entry focuses on extraction. For extractive summarizers (either for single-document summarization or multi-document summarization), they usually need to solve three problems: (i) Content selection, that is what should be selected from the text to form the summaries, which are most in the form of sentences or phrases; (ii) Information ordering, that is how to order the extracted sentences or phrases; (iii) Sentence realization, that is what kind of clean up to perform on the extracted sentences or phrases so they form a coherent summary. It is clear that the first problem is the critical one for extractive summarizers.

The algorithms for this problem can be categorized as either unsupervised methods, or supervised methods. Unsupervised methods do not need any training data (pairs of a texts and the corresponding summaries). They calculate the importance of a sentence by considering the position of the sentence, the contained terms or phrases, the discourse centrality and so on. Supervised methods rely on a set of training data, which designs some features to capture the importance of sentences and a model can be learned from the training data to predict the importance of sentences. With this model, the most important sentences can be selected to form a summary.

Extractive Summarization Algorithms

Unsupervised Methods The simplest and straightforward way to select sentences is based on the sentences' locations. Generally speaking, certain locations of the text (titles, headings, the first sentence in each paragraph, etc.) tend to contain important information. Therefore, by simply taking sentences in these locations, a summary can be constructed, which forms a strong baseline, even better than other methods [2]. Besides locations of sentences, the importance of a sentence is indicated by cue phrases. For example, once a sentence contains the phrases like "To sum up," "In summary," it is more likely to be extracted as an summary sentence. On the contrary, some other phrases like "specifically," "in details" indicate that the sentence is not "abstract" enough to be a summary sentence. In [15], Teufel and Moens manually built a list of 1,423 cure phrases in a genre of scientific texts and each cue phrase has a (positive or negative) "goodness score," also assigned manually.

There are also some methods calculating the importance of sentences based term importance and the term importance can be estimated by its frequency. The system of Luhn [8] is an typical example of this kind. In Luhn's method, every sentence is assigned with a significance factor, and the sentences with the highest significance factor are selected to form the summary. In order to compute the significance factor of a sentence, it is necessary to build a "significant words pool" which is defined as those words whose frequency is between high-frequency cutoff and low-frequency cutoff that can be tuned to alter the characteristics of the summarization system. After this is done, the significant factor of a sentence can be

computed in the following way: (i) set a limit L for the distance at which any two significant words could be considered as being significantly related. (ii) find out a portion in the sentence that is bracketed by significant words not more than L non-significant words apart. (iii) count the number of significant words contained in the portion and divide the square of this number by the total number of words within the portion.

In the above mentioned unsupervised methods, the sentences in a text are treated independently. In the following part, several more unsupervised methods are introduced, which exploit the relationship among sentences to some extent. The first one, as presented in [5], organizes the sentences in a text into a matrix and then apply a technique named Latent Semantic Analysis (LSA) [1] to derive the importance of sentences. A brief overview of LSA can make it easier to understand the LSA-based summarization method. LSA is based on singular value decomposition (SVD), a mathematical matrix decomposition technique that is applicable to text corpora as known by people. Given an m*n matrix $A = [A_1, A_2,...,A_n]$, with each column vector A_i representing the weighted term-frequency vector of sentence i in the document under consideration, the SVD is defined as:

$$A = U\Sigma V^T$$

where $U = [u_{ij}]$ is an m*n column-orthonormal matrix whose columns are called left singular vectors; $\Sigma = diag\ (\sigma_1,\sigma_1,...,\sigma_n)$ is an $n \times n$ diagonal matrix whose diagonal elements are non-negative singular values sorted in descending order. $V = [v_{ij}]$ is an $n \times n$ othonormal matrix whose columns are called right singular vectors.

As noted in [1], LSA is applicable in summarization for two reasons. First, LSA is capable of capturing and modeling interrelationships among terms by semantically clustering terms and sentences. Second, LSA can capture the salient and recurring word combination pattern in a document which describes a certain topic or concept. In LSA, concepts are represented by one of the singular vectors where the magnitude of the corresponding singular value indicates the importance of this pattern within the document. Any sentence containing this word combination pattern will be projected along this singular vector. The sentence that best represents this pattern will have the largest index value with this vector. Therefore, a summary can be constructed by collecting the sentences having largest index values over all concepts.

In [12], Mihalcea explicitly models the relationship among sentences by building a graph. In the graph, each node corresponds to a sentence and the weight of the edge linking two nodes is the similarity between the corresponding sentences. The direction of the edges can be decided by the appearance order of the sentences. After constructing the graph, Mihalcea employed some graph-based ranking algorithms like HITS and PageRank to decide the importance of a vertex (sentence) which can take into account the global information recursively computed from the entire graph. Finally, the sentences with highest ranking scores are selected to form a summary.

The third kind of summarization methods of exploiting sentence relationship is based on coherence relations. One example for the coherence relations is RST (rhetorical structure theory) relations. The RST relations are often expressed in terms of a satellite and a nucleus and nucleus sentences are more likely to be a summary sentence. More details of using RST for summarization can be found in [11].

Supervised Methods In the above mentioned unsupervised methods, each method estimates the importance of sentences based on some evidences from a certain aspect and then generate summaries based on the estimated importance. Therefore, a proper combination of these evidences may improve the generated summaries. Some supervised machine learning methods have been exploited for this goal. Among these methods, most of them treat the summarization task as a two-class classification problem at the sentence level, where the summary sentences are positive samples while the non-summary sentences are negative samples. After representing each sentence by a vector of features, a classification function such as Naive Bayes and Support Vector Machine can be trained [6]. Then for a new text, the trained classification function can be applied on each sentence in the text and decide whether it is a summary sentence. Although such methods are effective in most cases, they assume that the sentences are independent and classify each sentence individually without leveraging the relation among the sentences.

In order to address this shortcoming, some methods based on Hidden Markov Model (HMM) are exploited [3]. In Conroy et al.'s work [3], there are

two kinds of states, where one kind corresponds to the summary states and the other corresponds to non-summary states. The observations are sentences that are represented by a vector of three features. Given the training data, the state-transition probabilities and the state-specific observation probabilities can be estimated by the Baum-Welch algorithm or an EM algorithm. Given a new document, the probability that a sentence corresponds to a summary state can be calculated. Finally, the trained model can be used to select the most likely summary sentences.

It is clear that such approaches can handle the positional dependence and feature dependence when the feature space is small by taking some special assumptions. However, the HMM based methods have two open problems. Firstly, when the feature space is large and the features are not independent or are even overlapping in appearance, the training process will become intractable. Therefore this approach cannot fully exploit the potential useful features for the summarization task due to the computational inefficiency. Secondly, the HMM based methods set the HMM parameters to maximize the likelihood of the observation sequence. By doing so, the approach fails to predict the sequence labels given the observation sequences in many situations because they inappropriately use a generative joint-model in order to solve a discriminative conditional problem when observations are given. In [14], the authors use Conditional Random Fields (CRF) to replace HMM for text summarization which avoids these problems.

Evaluation

For extractive summarization methods, there are two popular evaluation measurements. The first one is by Precision, Recall and F_1 which are widely used in Information Retrieval. For each document, the manually extracted sentences are considered as the reference summary (denoted by S_{ref}). This approach compares the candidate summary (denoted by S_{cand}) with the reference summary and computes the precision, recall and F_1 values as shown in the following equation:

$$p = \frac{|S_{ref} \bigcap S_{cand}|}{S_{cand}} \quad r = \frac{|S_{ref} \bigcap S_{cand}|}{S_{ref}} \quad F_1 = \frac{2pr}{p + r}$$

A second evaluation method is by the ROUGE toolkit, which is based on N-gram statistics [7]. This tool is adopted by DUC for automatic summarization evaluation that was found to highly correlate with human evaluations.

Key Applications

Text summarization has been applied in many fields. For example, the snippets generated by search engines such as Google, Live Search are successful examples. Another example is to generate summaries for hand-held devices, whose screens are usually small [10]. Besides these, text summarization has also been used as a preprocessing step for some text mining tasks such as Web-page classification, which is expected to catch the main content of the Web pages while removing the noises [13].

Data Sets

Document Understanding Conferences (2001–2007) provide an open data source for different kinds of summarization tasks: http://duc.nist.gov/. More useful URLs are at http://www.summarization.com/.

Cross-references

▶ Text Mining
▶ Text Generation
▶ Text Classification
▶ Summarization
▶ Topic Detection and Tracking

Recommended Reading

1. Berry M.W., Dumais S.T., and O'Brien G.W. Using linear algebra for intelligent information retrieval. SIAM Rev., 37(4): 573–595, 1995.
2. Brandowa R., Mitzeb K., and Rauc L.F. Automatic condensation of electronic publications by sentence selection. Inform. Process. Manage., 41(6):675–685, 1995.
3. Conroy J.M. and O'leary D.P. Text summarization via hidden markov models. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 406–407.
4. Goldstein J., Kantrowitz M., Mittal V., and Carbonell J. Summarizing text documents: sentence selection and evaluation metrics. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 121–128.
5. Gong Y. and Liu X. Generic text summarization using relevance measure and latent semantic analysis. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 19–25.
6. Kupiec J., Pedersen J., and Chen F. A trainable document summarizer. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 68–73.
7. Lin C.-Y. and Hovy E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In Proc. Human Lang. Tech. Conf. of the North American Chapter of Assoc. Comput. Linguistics, 2003, pp. 71–78.

8. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Dev., 2(2), 1958.
9. Mani I. Advances in Automatic Text Summarization. MIT, Cambridge, MA, USA, 1999.
10. Mani I. Recent developments in text summarization. In Proc. 10th Int. Conf. on Information and Knowledge Management, 2001, pp. 529–531.
11. Marcu D. From discourse structures to text summaries. In Proc. ACL Workshop on Intelligent Scalable Text Summarization, 1997, pp. 82–88.
12. Mihalcea R. Language independent extractive summarization. In Proc. 20th National Conf. on AI and 17th Innovative Applications of AI Conf., 2005, pp. 1688–1689.
13. Shen D., Chen Z., Yang Q., Zeng H.-J., Zhang B., Lu Y., and Ma W.-Y. Web-page classification through summarization. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 242–249.
14. Shen D., Sun J.-T., Li H., Yang Q., and Chen Z. Document summarization using conditional random fields. In Proc. 20th Int. Joint Conf. on AI, 2007, pp. 2862–2867.
15. Teufel S. and Moens M. Sentence extraction as a classification task. In Proc. ACL Workshop on Intelligent Text Summarization. 1997, pp. 58–65.

Text Visualization

Haoda Huang, Benyu Zhang
Microsoft Research Asia, Beijing, China

Synonyms
Document visualization

Definition
Text visualization is a subarea of information visualization. The definition of information visualization is as follows: The use of computer-supported, interactive, visual representations of abstract data to amplify cognition [2]. Thus, the definition of text visualization is analogous: The use of computer-supported, interactive, visual representations of abstract text to amplify cognition. A more comprehensive and user-friendly definition is similar: The visual representation of text and its relationships.

Historical Background
In the early years, information such as texts and pictures were organized linearly: they were put in order and searched from beginning to end. But this is obviously not the most efficient way to organize information. It is well known that "A picture is worth a thousand words." People like to see news along with pictures, music, and even videos, rather than see purely raw text. Various technologies are also developed to help people graphically and visually represent their ideas, problems, challenges, solutions, and results. Today, information is not only presented one-dimensionally as in previous centuries, but also presented in two and more dimensions to help people understand the underlying idea clearly and thoroughly. Putting the right data in the right location and right format will greatly facilitate people seeking and understanding the information.

The history of text visualization is not very long. It started in the 1980s, when bandwidth and storage was very expensive. At that time, it was a luxury to play with advanced and real-time interactive graphics and visual effects. But with the rapid developments during the past several years, the standard PC platform provides high performance for processing videos and graphics and can present many advanced visual effects to users now. The 3D graphic interface has gradually dominated the gaming area, and 2D and 3D information visualization will become the mainstream of search engine retrieval and many other services.

Today, people are not satisfied with simply interacting with information in one dimension. Instead, they are more willing to search and manipulate their ideas and contents in multiple dimensions. This is becoming the main trend today and is also presenting a great challenge to traditional information organizations, such as libraries and museums. They are required to develop advanced 2D and 3D text visualization technologies to maintain their market share.

Foundations
Text visualization is about representing the underlying structure of a text or a group of texts. Text visualization offers several benefits for users. With text visualization, for example, users could have a view of texts in different levels of abstraction, could have a better view of the relationships between texts, and could have a high-level view of the topics in the text collection.

Generally, the ideas of many text visualization algorithms are borrowed from data analysis research areas while accounting for the specific properties of texts. These properties include the following [6,8]: "(i) High data dimensionality when using typical bag-of-words representation, where each word and each phrase

represents on dimension in the data space. (ii) High redundancy, meaning that many dimensions can be easily merged into one dimension without losing much information. This is caused by the two properties of words, namely synonymy (different surface word forms having the same meaning – e.g., singer, vocalist) and hyponymy (one word denotes a subclass of an another – e.g., breakfast, is a subclass of a meal). (iii) Ambiguity between words in the cases where the same surface form of the word has different meanings (homonomy – e.g., the word "bank" can mean "river bank" or "financial institution") or in the cases where the same form has related meaning (polysemy – e.g., "bank" can mean "blood bank" or "financial institution"). (iv) Frequency of words (and phrases) follows power distribution. Appropriate weighting schemas (e.g., most popular being TFIDF) are used to normalize importance of the words to be able to work with the standard data analytic techniques."

Meanwhile, there are many types of texts, such as web documents, emails and news group postings, literature, and legal documents. In developing a text visualization algorithm or a system for a special type of text, the properties of the type of text also needs to be considered. For example, to visualize news articles, [8] have considered the following properties in their approach: (i) shorter documents; (ii) written by professionals; (iii) low number of mistakes; (iv) having good rhetorical structure; (v) rich information about people, companies, or phrase; and (vi) single documents as pieces of larger stories spanning over several documents.

Many text visualization algorithms adopt bag-of-words text representation, where text is viewed as a bin of independent words with term dependencies and with any other positional information of terms ignored. This simplification appears to be reasonable, since in many cases the efficiency of solving relevant problems does not degrade much. In the bag-of-words representation, each word is represented as a separate variable with a numeric weight. This numeric weight is often calculated using the famous TFIDF weighting schema: the weight is the multiply of term frequency and the inverse document frequency. The idea of TFIDF is very intuitive: if the word appears more times in a text, it would be more important; and if the word appears in fewer texts in the text corpus, it would be more important. In the bag-of-words representation, a

text is represented as high-dimensional sparse vector, so it cannot be directly visualized. A clustering algorithm needs to be applied to indentify the relationships between texts and then to map them into 2D or 3D space. The cosine similarity measure is widely used to evaluate relationship between texts. There are also some other typical ways of text visualization using graphs or trees to present frequent co-occurrences of words and phrases. Typical visualization algorithms include graph based visualization and tiling based visualization as introduced in [4]:

Graph-based visualization has the following algorithm sketch: (i) First, documents are transformed into the bag-of-words sparse-vectors representation. Words in the vectors are weighted using TFIDF. (ii) Then, K-Means clustering algorithm is used to split the documents into K groups. Each group consists of similar documents and these are compared using cosine similarity. The K groups form a graph with groups corresponding to graph nodes and similar groups linked. Each group is represented by characteristic keywords. (iii) Finally, simulated annealing is used to draw a graph.

Tiling-based visualization has the following algorithm sketch: (i) First, documents are transformed into the bag-of-words sparse-vectors representation. Words in the vectors are weighted using TFIDF. (ii) Then, hierarchical top-down two-wise K-Means clustering algorithm is used to build a hierarchy of clusters. The hierarchy is an artificial equivalent of hierarchical subject index just like Yahoo. (iii) Finally, the leaf nodes of the hierarchy (bottom level) are used to visualize the documents. Each leaf is represented by characteristic keywords and each hierarchical splits the rectangular area into two sub-areas recursively.

Key Applications

WebSom [7] gives self-organizing Maps for Internet Exploration. An ordered map of the information space is provided: similar documents lie near each other on the map. The algorithm automatically organizes the documents onto a two-dimensional grid so that related documents appear close to each other.

ThemeScape [3] graphically displays images based on word similarities and themes in text. Themes within the document spaces appear on the computer screen as a relief map of natural terrain. The mountains indicate where themes are dominant, valleys indicate weak themes. Themes close in content will be close visually

based on the many relationships within the text spaces. The algorithm is based on K-means clustering.

ThemeRiver [5] helps users identify time-related patterns, trends, and relationships across a large collection of documents. The themes in the collection are represented by a "river" that flows left to right through time. The theme currents narrow or widen to indicate changes in individual theme strength at any point in time.

The text representation is the fundamental work for IR and text mining. Besides IR, it can also be used for text categorization, text clustering, topic detection and novelty detection etc.

Future Directions
Although there have been many text visualization systems that have achieved much success in the past, there still exists many ways to improve them. One is to use natural language processing tools to do more detailed analysis or to improve the text summarization. Another may be to design more and better interaction tools for users to manipulate the texts. Advanced 2D and 3D graphics techniques may also be used to make the user interfaces friendlier.

Cross-references
► Data Mining
► Information Retrieval
► Text Segmentation

Recommended Reading
1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. ACM Press, NewYork, NY, 1999.
2. Card S., Mackinlay J., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Academic Press, 1997.
3. Cartia. ThemeScape Product Suite. Available at: http://www.cartia.com/products/index.html
4. Grobelnik M. Text Visualization Tutorial.
5. Havre S., Hetzler E., Whitney P., and Nowell L. ThemeRiver: Visualizing thematic changes in large document collections. IEEE Trans. Vis. Comput. Graph., 8(1):9–20, 2002.
6. Jurafsky D. and Martin J.H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, 2000.
7. Lagus K., Kaski S., and Kohonen T. Mining massive document collections by the WEBSOM method. Inf. Sci., 163 (1–3):135–156, 2004.
8. Marko G. and Dunja M. Visualization of news articles. In SIKDD 2004 at Multiconference IS. Ljubljana, Slovenia, 2004, pp. 12–15.

Text/Document Summarization

► Summarization

Text-based Image Retrieval

► Annotation-based Image Retrieval

TF*IDF

Ibrahim Abu El-Khair
Minia University, Minia, Egypt

Synonyms
Term frequency by inverse document frequency

Definition
A weighting function that depends on the term frequency (TF) in a given document calculated with its relative collection frequency (IDF). This weighting function is calculated as follows [1] Assuming that term j occurs in at least one document d ($dj \neq 0$), the inverse document frequency (idf) would be

$$\mathrm{Log}_2\left(\mathrm{N}/\mathrm{d}_j\right) + 1 = \log_2 \mathrm{N} - \log_2 \mathrm{d}_j$$

The ratio dj/N is the fraction of documents in the collection that contain the term. The term frequency-inverse document frequency weight (TF*IDF) of term j in document i is defined by multiplying the term frequency by the inverse document frequency:

$$\mathrm{W}_{ij} = f_{ij}{}^\star[\log_2 \mathrm{N} - \log_2 \mathrm{d}_j]$$

Where

  N: number of documents in the collection
  d_j: number of documents containing term j
  f_{ij}: frequency of term j in document i
  W_{ij}: is the weight of term j in document i

The use of the logarithm in the formula rather than the actual values of N and D_k moderates the effect of increasing the collection size and the effect of a high term frequency.

Key Points

The significance of a certain term in a given document is determined using the term frequency (TF) of that term in the document. Unfortunately, while term frequency is a good measure for term significance in one document, it is not an adequate measure for its significance in a collection of documents. The use of this factor alone in calculating the term weights in a collection of documents does not guarantee adequate retrieval performance. Some high frequency terms are not concentrated in a few particular documents, they are common in the whole collection which means that all these documents will be retrieved, affecting the performance of the information retrieval system.

The solution for this problem [4] is correlating the term frequency with its relative collection frequency (IDF), making the collection frequency a variable in retrieval. The use of collection frequency places a greater emphasis on the value of a term as a means of distinguishing one document from another than its value as an indication of the content of the document itself. Combining the two factors, TF and IDF enables the information retrieval system to exploit the good features of both. TF emphasizes term significance in a given document, and IDF emphasizes term significance in the collection as a whole; i.e., if the term was common in a document and rare in the collection it would be heavily weighted in both schemes. This way a term can distinguish certain documents from the remainder of the collection.

Even though this weighting function is a good indication of the term importance in a given set of documents it overlooks an important factor which is the document length. This problem may affect the weighting process because in real life documents have different lengths, and longer documents may have higher frequencies for a given term because it is repeated several times in the document. This increases the weight of the term and increases the possibility of retrieving these documents because of the higher weight of terms in them. Long documents also may have more different terms them which may affect the retrieval as well. More terms in a given document increases the possibility of matching between this document and multiple queries [3]. A possible way to overcome this problem is to incorporate a normalization factor for the document length to reduce its effect and make the weighting function more effective. A number of variant formulae for tf*idf weighting are given in [2].

Cross-references

► BM25
► Information Retrieval
► Term Weighting
► Text Indexing Techniques

Recommended Reading

1. Korfhage R.R. Information Storage and Retrieval. Wiley, New York, USA, 1997.
2. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK, 2008.
3. Salton G. and Buckley C. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage., 24(4):513–523, 1988.
4. Sparck J.K. A statistical interpretation of term specify and its application in retrieval. J. Doc., 28:11–20, 1972.

tgd

► Tuple-Generating Dependencies

Thematic Map

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Thematic map; Data map; Chart

Definition

The graphical representation of quantitative data or qualitative data for a given geographic area (e.g., regional unemployment rate or type of crops grown).

The many different methods that cartographers apply to create thematic maps are used to define a thematic map more precisely. The following five are found most often.

1. *Coropleth maps.* In coropleth maps the areas of the map are shaded or patterned in proportion

to the value of the data to be displayed for a particular area.

2. *Dot maps.* These maps use equal sized dots to show the presence of a feature at a particular geographic location and thus display spatial distributions. A dot need not be restricted to a single occurrence; it may indicate any number of entities.

3. *Proportional symbol maps.* When drawing proportional symbol maps, a cartographer selects a symbol (e.g., a circle or a bar), places it at the spot on the map to which the data apply and varies the symbol's size from place to place in proportion to the value of the variable that the symbol represents.

4. *Isarithmic maps.* These maps use contour lines to join points of equal value. Examples are barometric pressure lines in weather maps or elevation above sea level in topographic maps.

5. *Dasymetric maps.* These maps divide a geographic region with contour lines and shade or pattern the resulting regions in proportion to the value that applies to the region bounded by a contour line.

Topographic maps that provide reference to geographic features (political boundaries, roads, lakes, mountains) are generally not categorized as thematic maps.

Key Points

History. Edmund Halley (famous for discovering the comet bearing his name) is recognized as the author of the first thematic map. Drawn in 1686, it shows the direction of trade winds on a world map. Probably the best known example of using thematic maps for data analysis is John Snow's cholera map of 1855, a data display based on principles still applied in today's geographic information systems. These and more historical notes are summarized in [3].

   Usage. Thematic maps display the spatial distribution of data for a specific subject or a specific purpose. They represent information about particular locations in such a way that spatial patterns emerge. Frequently thematic maps are used to compare patterns on two or more maps. Examples are election results, occurrence of particular types of diseases, usage of agricultural land, climatic change over time and so on. Major contributions to thematic mapping come from the French cartographer Jacques Bertin ([1]). A more modern treatment of the topic can be found in [2].

Cross-references
► Data Visualization
► Chart

Recommended Reading
1. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin, New York, 1981.
2. Slocum T.A., McMaster R.B., Kessler F.C., and Howard H.H., Thematic Cartography and Geographic Visualization, 2nd edn. Pearson-Prentice Hall, Upper Saddle River, NJ, 2005.
3. Tufte E.R., The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.

Theme Algebra

► Spatial Operations and Map Operations

Thesauri Business Catalogues

► Lightweight Ontologies

Thiessen Polygons

► Voronoi Diagram

Third Normal Form

MARCELO ARENAS
Pontifical Catholic University of Chile, Santiago, Chile

Definition
Let $R(A_1,...,A_n)$ be a relation schema and Σ a set of functional dependencies over $R(A_1,...,A_n)$. An attribute A_i ($i \in \{1,...,n\}$) is a *prime* attribute if A_i is an element of some key of $R(A_1,...,A_n)$. Then specification (R, Σ) is said to be in Third Normal Form (3NF) if

T

for every nontrivial functional dependency $X \rightarrow A$ implied by Σ, it holds that X is a superkey for R or A is a prime attribute [2].

Key Points

In order to avoid update anomalies in database schemas containing functional dependencies, 3NF was introduced by Codd in [2]. This normal form is defined in terms of the notions of prime attribute and key as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$, it holds that $(R(A, B, C), \Sigma)$ is in 3NF since AB is a superkey and C is a prime attribute (given that AC is a key for R). On the other hand, $(S(A, B, C), \Gamma)$ is not in 3NF if $\Gamma = \{A \rightarrow B\}$, since A is not a superkey for S and B is not a prime attribute.

For every normal form two problems have to be addressed: how to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. On the positive side, for every relation schema S there exists a database schema S' such that, S' is in 3NF and S' is a lossless and dependency preserving decomposition of S. Furthermore, schema S' can be generated efficiently by using the synthesis approach proposed in [1]. On the negative side, it is expensive to check whether a schema is in 3NF. It was shown by Jou and Fischer that this problem is NP-complete [3].

Cross-references
▶ Boyce-Codd Normal Form
▶ Fourth Normal Form
▶ Normal Forms and Normalization
▶ Second Normal Form (2NF)

Recommended Reading

1. Biskup J., Dayal U., and Bernstein P. Synthesizing independent database schemas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 143–151.
2. Codd E.F. Further normalization of the data base relational model. In Proc. Data Base Systems. Prentice-Hall, Englewood Cliffs, NJ, USA, 1972, pp. 33–64.
3. Jou J. and Fischer P. The complexity of recognizing 3NF relation schemes. Inf. Process. Lett., 14(4):187–190, 1982.

Thread Lifecycle

▶ Process Life Cycle

Three-Dimensional GIS and Geological Applications

MARTIN BREUNIG
University of Osnabrueck, Osnabrueck, Germany

Synonyms

Spatial information system; Geoscientific information system

Definition

An information system for the in-/output, modeling, management, processing, analyzing and visualization of geoscientific data including geo-referenced three-dimensional geometric, topological and attribute data. The three-dimensional geometric data may consist of points/vertices (x,y,z-coordinates), curves, surfaces and polyhedra, respectively. The topological data may consist of nodes, edges, faces and solids, respectively. Typical attribute data are descriptions of geological strata, i.e., properties of strata such as "geological age," "soil type," "main components of the stratum" etc.

The implementation of a three-dimensional GIS provides data types, spatial access structures including geometric/topological algorithms and a spatial or visual query language for the modeling, management and analysis of geo-referenced three-dimensional data.

Historical Background

Three-dimensional GIS have two roots in the history of information systems. The first has its origin in the field of 3D modeling and visualization. 3D modeling and visualization systems usually provide a large collection of geometric and topological 3D algorithms. An example is the Discrete Smooth Interpolation (D.S.I.) algorithm of Jean-Laurent Mallet and others [12,13] used for creating and editing triangulated surfaces. Given an arbitrary mesh with an arbitrary set of vertices fixed by the user, D.S.I. assigns coordinates to the other nodes of the mesh, enabling the fixed vertices to be interpolated smoothly [12]. The second root has its origin in the field of *Geographical Information Systems* (GIS). GIS are inherently using both, spatial and attribute data, in their data management. However, standard Geographical Information Systems usually are not prepared to manage and

process real 3D data. They only allow the visualization of 2.5D data representation such as digital elevation models. However, in this representation every (x,y)-coordinate may only have one z-coordinate. That is why Geographical Information Systems cannot be used to solve 3D geological problems. In particular, polyhedra and solids cannot be treated in Geographical Information Systems. In today's Geographical Information Systems, the third dimension is only treated as a thematic attribute such as the height value of an isoline map.

Relevant work in the field of three-dimensional GIS has been published by [2,5–7, 13,16,18,–20,22], and by other authors. For example, theoretical work on three-dimensional GIS and topological data models has been worked out by [15]. Literature about *spatial database systems* can be found in [8]. Spatial database systems can be extended for the management of three-dimensional GIS data. For example, an R-Tree can be used to access 3D data.

The term "three-dimensional GIS" (3D GIS) is not yet used in a standardized way. Unfortunately, the term "3D GIS" or "3D city model" is often used for information systems that only deal with surface and face data, but not with polyhedra and solids. Correctly, such information systems should be called "2.5D GIS," because they only allow to represent surfaces with the following property, valid for all of their points P(x, y, z): z = f (x, y). i.e., every point (x, y) of the surface has a different z-value.

First standardization efforts for the exchange of data between 3D GIS have been undertaken by the Open Geospatial Consortium [14] within the *Geography Markup Language* (GML).

Foundations

Spatial planning processes and the study of geoscientific processes often require three-dimensional information. This digital information must be modeled, managed, visualized and analyzed. A three-dimensional GIS is an information system that accomplishes these requirements. To work adequately, it needs 3D data types, 3D spatial access structures, and 3D geometric and topological algorithms to manage and process three-dimensional data of these applications. Furthermore, a user interface with a query language is required. A spatial database system may be embedded into a 3D GIS to support these requirements.

In a 3D GIS, objects of different dimension d $(0 \leq d \leq 3)$ have to be processed. The 3D geometry of a geological object can be composed of sets of points, lines, surfaces and volumes, respectively. As a reference model for 3D geometric data types in three-dimensional GIS, *Simplicial Complexes* may be used: points, polylines, triangle nets and tetrahedron nets in three-dimensional space. They are often used in geological applications, because they well approximate 3D solids and surfaces formed by nature.

In a 3D topology model of a 3D GIS, the components of the objects are interpreted as a mesh of nodes, edges, faces, and solids that describes both the interior structure of the geological objects and their mutual neighborhood relationships in 3D space. As a reference model for 3D topological data types in three-dimensional GIS, cellular complexes [4,13] and Generalized Maps [10,11,13] respectively, may be used. They provide a general topological model treating 2D, 2.5D and 3D objects in a uniform way. Furthermore, they are based on the clear mathematical theory of algebraic topology.

To access 3D data, existing spatial access structures such as the R-Tree [9] or R*-Tree [3] may be applied. The spatial access structures can also be used internally within the processing of geometric 3D algorithms as a first filter step to compute on approximated geometries – 3D boxes are used most – to achieve better performance, for example to support the intersection of very large geometries such as sets of tetrahedron nets. For the analysis of 3D data, 3D GIS are using geometric and topological algorithms, e.g., for computing the distance between two objects, the intersection between two polyhedra, or the neighbor objects of a given geological object.

Three-dimensional GIS are subject of current research [1,21]. Concepts and prototypical software considering aspects of three-dimensional GIS such as architectural issues, spatial data modeling with constraints, efficient spatial data access and geological applications have been described in detail by many authors of the GIS and database communities (see Recommended Reading). First 3D GIS prototype systems are already used in some application domains of three-dimensional GIS such as geology, geophysics, archaeology, and early diagnosis of natural disasters. Also database vendors are integrating first simple 3D data types and operations in their products.

Key Applications

One of the most relevant applications of three-dimensional GIS is geology. In geology, the starting point of the examinations is the present condition of the earth's structure. Hence the three-dimensional geometric analysis of recent geological strata and solids is the key for further investigations concerning the interaction of former geological structures and processes [19]. Furthermore, consistent geometric 3D models and the use of GIS are the precondition for the production of digital geological maps. For example, the 3D models can be intersected with digital elevation models to improve maps or to define boundary conditions for 3D models. The computational expense of GIS, however, is high, because the third dimension is included.
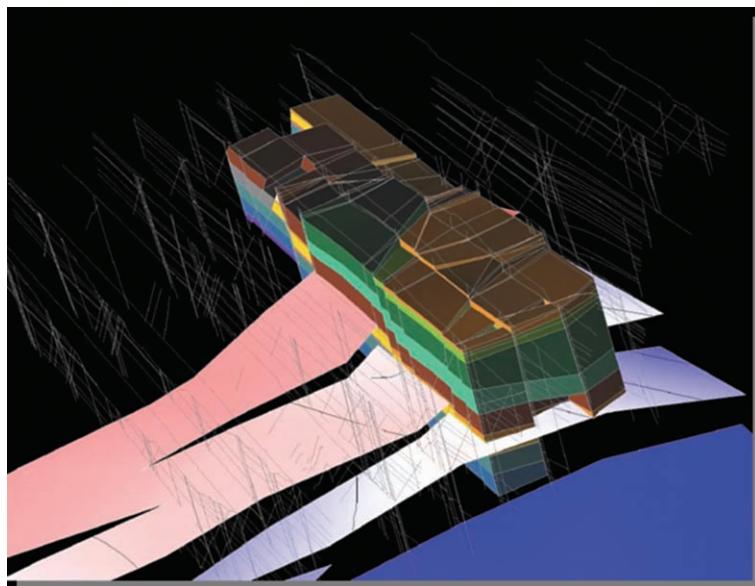
Some of the relevant data in geological applications are sections, stratigraphic boundaries, and faults. Sections describe a mostly vertical intersection through the geological structure of an examination area, i.e., an estimation of the geological surfaces and solids including geological faults. A cross section through the series of sediments consists of a set of stratigraphic lines. Their geometry is mostly given by a point set in three-dimensional space.

The geometry of stratigraphic boundaries is based on stratigraphic lines in the sections where the stratum occurs. The stratigraphic surfaces are spread between the cross sections. Triangle nets in three-dimensional space can result from the triangulation of the surfaces between the point sets of the cross sections. Concerning topology, on each cross section the underlying and hanging stratum of each stratigraphic line can be identified.

Faults are modeled as surfaces, along which the tectonic deformation of the sediments took place. Usually the geometry of a fault consists of a triangle net in three-dimensional space. The modeling of strata and faults from cross section to cross section is an interpolation of new geometries from the sections.

Figure 1 shows an example of geological data managed by a three-dimensional GIS. During the modeling process of geologically defined geometries, a large amount of data are accumulated. Thus the handling of three-dimensional data in a spatial database system [8] is recommended for 3D GIS. for example, the efficient spatial access on a large set of geological objects is necessary to compute intersections between single strata and faults with different thematic attributes, respectively. Therefore, the implementation of efficient three-dimensional geometric algorithms has to be provided by the three-dimensional GIS. Checking the consistency of geometric

Three-Dimensional GIS and Geological Applications. Figure 1. Example of geological data in the Lower Rhine Basin, managed by a three-dimensional GIS (figure constructed in the group of Agemar Siehl, University of Bonn, with the GOCAD Software, Nancy, now distributed by Paradigm, UK).

3D models is essential for the geometric 3D reconstruction of geological structures and geological processes [19].

Cross-references

▶ Digital Elevation Models
▶ Geography Markup Language
▶ Simplicial Complex
▶ Spatial Network Databases

Recommended Reading

1. Abdul-Rahman A., Zlatanova S., and Coors V. (eds.). Innovations in 3D Geoinformation Systems, Lecture Notes in Geoinformation and Cartography, Springer, Heidelberg, 2006.
2. Balovnev O., Bode T., Breunig M., Cremers A.B., Müller W., Pogodaev G., Shumilov S., Siebeck J., Siehl A., and Thomsen A. The story of the GeoToolKit – an object-oriented geodatabase kernel system. Geoinformatica, 8(1):5–47, 2004.
3. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
4. Brisson E. Representing geometric structures in d dimensions: topology and order. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 218–227.
5. Coors V. and Zipf A (eds.). 3D-Geoinformationssysteme, Grundlagen und Anwendungen. Wichmann – Hüthig, Heidelberg, 2004.
6. GOCAD. http://www.gocad.org.
7. Götze H.J. and Lahmeyer B. Application of three-dimensional interactive modelling in gravity and magnetics. Geophysics, 53(8), 1988, pp. 1096–1108.
8. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.
9. Guttman A. R-Trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
10. Lienhardt P. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 228–236.
11. Lienhardt P. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. J. Comp. Geom. App., 4(3): 275–324, 1994.
12. Lévy B. and Mallet J.-L. Discrete Smooth Interpolation: Constrained Discrete Fairing for Arbitrary Meshes, ISA-GOCAD (Inria Lorraine/CNRS), ENSG, Vandoeuvre Nancy, http://www.earthdecision.com/news/white_papers/DSI.pdf.
13. Mallet J.L. Geomodelling. Oxford University Press, New York, NY, 2002.
14. OGC. http://www.opengeospatial.org.
15. Pigot S. A topological model for a 3D spatial information system. In Proc. 5th Int. Symp. on Spatial Data Handling, 1992, pp. 344–360.
16. Raper J. (Ed.). Three dimensional applications in geographical information systems. Taylor & Francis, London, 1989.
17. Samet H. The design and analysis of spatial data structures, Addison-Wesley, Reading, 1990.
18. Schaeben H., Apel M., v.d. Boogart G., Kroner U. GIS 2D, 3D, 4D, nD. Informatik-Spektrum, 26(3), 2003, pp. 173–179.
19. Siehl A. Construction of geological maps based on digital spatial models. Geol. Jb. A., 104:253–261, 1988.
20. Turner A.K. (ed.) Three-dimensional modeling with geoscientific information systems, Kluwer Academic, Dordrecht, 1991.
21. van Oosterom P., Zlatanova S., Penninga F., and Fendel E. (eds.) Advances in 3D geoinformation systems, Lecture Notes in Geoinformation and Cartography. Springer, Heidelberg, 2007.
22. Vinken R. Digital geoscientific maps – a research project of the DFG. In Proc. Int. Colloquium at Dinkelsbühl, Geolog. Jahrbuch A104, 1988, pp. 7–20.

Three-Dimensional Similarity Search

▶ Feature-Based 3D Object Retrieval

Three-Phase Commit

YOUSEF J. AL-HOUMAILY[1], GEORGE SAMARAS[2]
[1]Institute of Public Administration, Riyadh, Saudi Arabia
[2]University of Cyprus, Nicosia, Cyprus

Definition

Three-phase commit (3PC) is a synchronization protocol that ensures *global atomicity* of distributed transactions while alleviating the *blocking* aspect of 2PC (Two-Phase Commit) in the events of *site* failures. That is, 3PC never requires operational sites to wait (i.e., block) until a failed site has recovered.

Historical Background

3PC was one of the first attempts to resolve the blocking aspects of 2PC [6]. The main purpose of the protocol is to allow operational sites to continue transaction processing and reach agreement about the final status of transactions in spite of the presence of site failures. 3PC can tolerate any number of site failures (except for total sites' failures), assuming a highly reliable network (i.e., a network that never causes operational sites to be partitioned into more than one set of communicating sites, implying a network that never fails).

Foundations
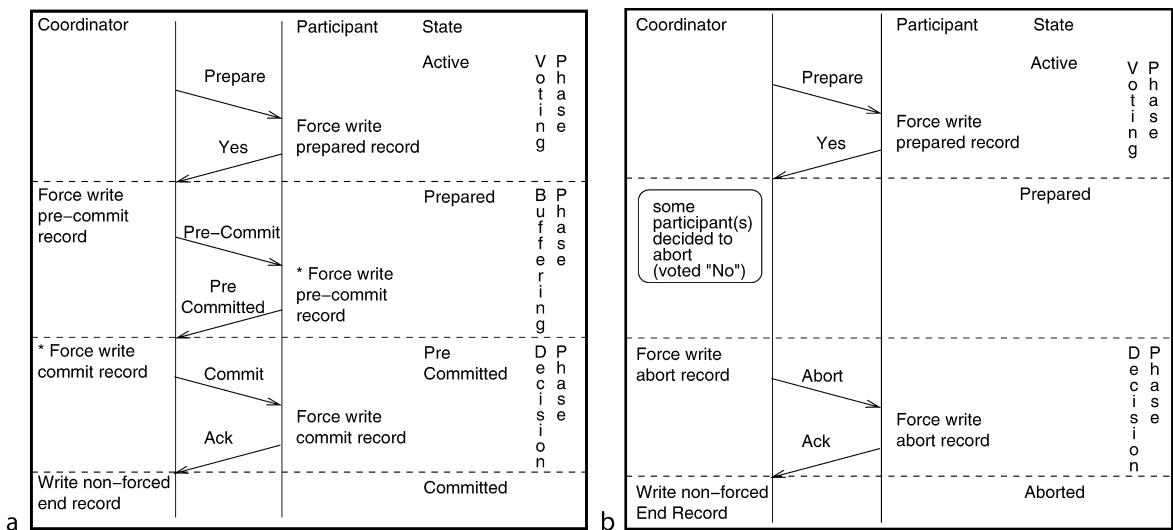
In 2PC, a participant is *blocked* if it fails to communicate with the coordinator of a transaction while in a prepared-to-commit state. Blocking means that the participant cannot determine the final status of the transaction in the presence of a failure, rendering all resources held by the prepared-to-commit transaction at its site unusable by any other transaction until the final status of the transaction is resolved, i.e., the transaction is either committed or aborted. Blocking is inevitable in 2PC and it may occur either because of (i) a communication (link) failure or (ii) a coordinator's system crash. These two types of failures lead to blocking, in 2PC, even under the assumption that all participants remain operational and can communicate collaboratively to resolve the status of the prepared-to-commit transaction. For example, if a coordinator fails after all participating sites in a transaction's execution have entered their prepared-to-commit states; the participants (collectively) can neither commit nor abort the transaction. This is because the operational participants cannot be sure whether the coordinator had received all their votes and made a commit final decision just before it failed or it had received only some votes and did not have the chance to make the final decision before its failure and the transaction will be aborted by the coordinator when it

recovers. Thus, the participants are blocked until the coordinator recovers.

The negative impact of blocking on the (i) overall system performance and (ii) availability of critical data on other transactions motivated the design of *non-blocking* atomic commit protocols (ACPs). In 3PC, an *extra (buffering) phase* is inserted between the two phases of 2PC to capture all the participants' intentions to commit, as shown in Fig. 1.

Dynamics of Three-Phase Commit

The basic idea behind the insertion of the buffering phase is to place the two (possible) reachable final states (i.e., the commit and the abort states) for a transaction apart from each other such that they cannot be reached from the *same* state. That is, if a final state can be reached from the current state of a site, then, the reachable final state can be either the commit or the abort state but not both. In 2PC, when a participant is in a prepared-to-commit state, both final states can be reached. Hence, if the coordinator fails, the participant cannot determine the final state for the transaction without any possible conflict in its decision with the coordinator. For this reason, the protocol is blocking. On the contrary and as shown in Fig. 1, the commit final state for a site (whether it is the coordinator or a participant), in 3PC, cannot be reached from the same

Three-Phase Commit. Figure 1. The three-phase commit protocol.

state as the abort final state. In the former case, the commit state can be reached from the *pre-commit* state whereas, in the latter case, the abort state can be reached from the *prepared* state.

When a site is in a pre-commit state, it means that each of the other sites is at least in a prepared-to-commit state (Notice that the other sites might lag in their states because of system's delays such as queuing and network delays.). Thus, the pre-commit state is called a *committable state* since it implies that all participants have voted "yes" and the coordinator agreed on the commitment of the transaction. In 3PC, a *non-committable* state, i.e., a state that does not imply that all the participants have voted "yes," is not placed adjacent to a commit state. This is not the case in 2PC as the prepared state, which non-committable, is placed adjacent to a commit state.

The insertion of the buffering state makes the structure of 3PC to satisfy the two necessary and sufficient conditions for the construction of synchronous non-blocking ACPs within one state transaction, i.e., a structure where neither the coordinator nor any participant leads each other by more than one state transition during its execution of the protocol. That is, 3PC is synchronous within one state transaction that (i) does not contain a state that is adjacent to both a commit and an abort state, and (ii) it does not contain a non-committable state that is adjacent to a commit state.

Based on the above, if the coordinator of a transaction fails at any point during the execution of the protocol, the operational participants can collectively and deterministically decide the final status of the transaction. The decision is commit if any of the participants is in *at least* a pre-commit state (because it is not possible for the coordinator to have decided to abort). Otherwise, the decision is abort (because it could be possible for the coordinator to have decided to abort but not to commit). To reach an agreement on the final status of a transaction, there is a need for a termination protocol which is invoked when the coordinator fails.

Recovery in Three-Phase Commit

When a participant times out while waiting for a message from the coordinator, it means that the coordinator must have failed (or it is perceived as a coordinator failure). In this case, the participant initiates an election protocol to determine a *new* coordinator. One way to determine the new coordinator is based on sites'

identification numbers such that the participant with the highest (or the lowest) number becomes the new coordinator. Once a new coordinator is elected, the participants exchange status information about the transaction. If the new coordinator finds the transaction in at least a pre-commit state at any participant, it commits (in its local log) the transaction; otherwise, it aborts the transaction. Then, this new coordinator proceeds to complete the 3PC for the transaction in all the other participants. If the new coordinator fails, the election process is repeated again.

When a participant starts recovering from a failure, it needs to determine the status of each prepared or pre-committed transaction as recorded in its log. Notice that a recovering participant cannot commit a transaction even if the participant is in a pre-commit state with respect to the transaction. This is because the operational sites might have decided to abort the transaction after the participant had failed if none of them was in a pre-commit state. In this case, the participant must ask the other sites about the final status of the transaction.

A final note on 3PC is about total sites' failure where there is a need to determine the last participant to have failed. This is because such participant is the only one which can decide the status of the transaction for the other participants. Determining the last participant that has failed could be implemented by maintaining an "UP" list at each participating site. This list contains the identities of operational participants as seen by the participant that is maintaining the list and is stored (in a non-forced or asynchronous manner) onto the stable log of the participant. Thus, the "UP" lists allow a set of participants to determine, upon their recovery from the total failure, whether they contain among themselves the last participant to have failed, reducing the number of participants that needs to recover before the transaction status can be resolved. Alternatively, all participants should recover and become operational again before the status of the transaction can be resolved.

Non-Blocking Commit Protocol Variants

As discussed above, blocking occurs in 2PC when the coordinator of a transaction crashes while the transaction is in its *prepared-to-commit* state at a participating site. In such a case, the participant is blocked until the coordinator of the transaction recovers. In general, all ACPs are susceptible to blocking. They just differ in the

size of the window during which a site might be blocked and the type of failures that cause their blocking. Several ACPs have been designed to eliminate some of the blocking aspects of 2PC, besides 3PC, by adding extra coordination messages and forced log writes. These protocols can be classified into whether they preserve the prepared-to-commit state, such as cooperative 2PC, or allow *unilateral or heuristic* decisions in the presence of unbearable delays, such as IBM's presumed nothing (IBM-PrN).

The *cooperative* 2PC (1981) reduces the *likelihood* of blocking in case of a coordinator's failure. In the cooperative 2PC, the identities of all participants are included in the prepare-to-commit message so that each participant becomes aware of the other participants. In the case of a coordinator's or a communication's failure, a participant does not block waiting until it reestablishes communication with the coordinator. Instead, it inquires the other operational participants in the transaction's execution about the final decision and if any of them has already received the final decision prior to the failure, it informs the inquiring participant accordingly.
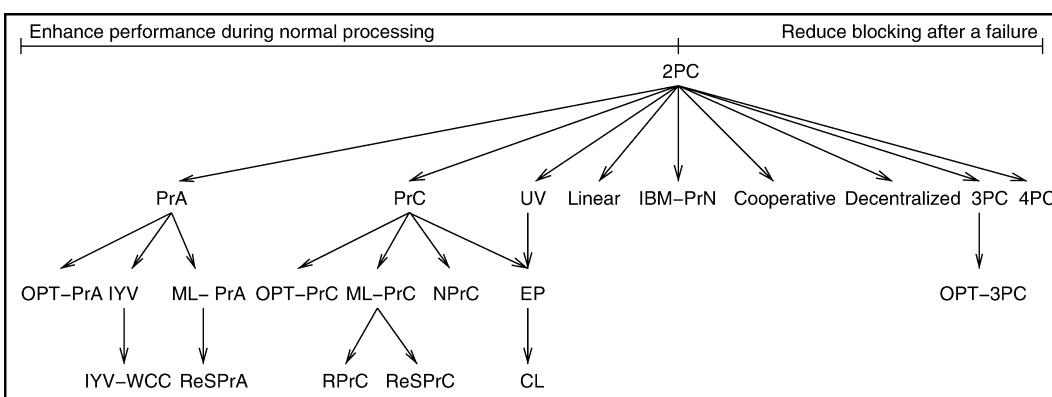
The IBM-PrN (1990) is a 2PC variant that allows blocked participants to unilaterally commit or abort a transaction and detects atomicity violations due to conflicting heuristic decisions. In the event of atomicity violations, it reports any damage on transactions and data, simplifying the task of identifying problems that must be fixed. *Generalized presumed abort* (1994) is another IBM protocol that behaves like IBM-PrN when complete confidence in the final outcome and recognition of heuristic decisions is required and behaves like PrA during normal processing. Recent efforts to enhance commit protocols with heuristic decision processing resulted in the *allow-heuristics presumed nothing* (1996) commit protocol.

Other Atomic Commit Protocol Variants and Optimizations

Figure 2 shows some of the significant steps in the evolution of ACPs including the two most notable 2PC variants which are *presumed abort* (PrA) and *presumed commit* (PrC) [2]. The *new PrC* (NPrC) (1993) and *rooted PrC* (RPrC) (1997) protocols were proposed to reduce the log complexity of PrC further at the cost of slower recovery in the presence of failures. NPrC eliminates the initiation log record at the coordinator's site whereas RPrC eliminates the initiation log record at each cascaded coordinator when the *tree-of-processes* (or *multi-level transaction execution*) model is used.

In contrast to PrA and PrC variants, other 2PC variants have been proposed for specific environments. The common characteristic of these protocols is that they exploit the semantics of the communication networks, the database management systems and/or the transactions to enhance the performance of 2PC. For example, the *linear* 2PC (L2PC) reduces message complexity at the expense of time complexity compared to 2PC by assuming token-ring like networks. In L2PC, the participants are linearly ordered with the coordinator being the first in the linear order. The coordinator initiates the voting and each participant sends its "yes" vote to its successor in the linear order. The last participant in the order makes the decision and sends it to its predecessor and so on. In this way, L2PC maintains the same log complexity as 2PC, reduces

Three-Phase Commit. Figure 2. Some significant steps in the evolution of ACPs.

the message complexity of 2PC from "3" to "2n" while increasing the time complexity of 2PC from "3" to "2n" rounds, where n is the number of participants. In contrast to L2PC, *decentralized* 2PC (D2PC) reduces time complexity at the expense of message complexity which is $n^2 + n$ messages. In D2PC, the interconnecting communication network is assumed to be fully connected and efficiently supports the broadcasting of messages. In D2PC, two rounds of messages are required for each individual participant to make a final decision. During the first round, the coordinator broadcasts its vote (implicitly initiating commit processing) whereas, during the second one, all the participants broadcast their votes. Thus, each participant receives the votes of all the other participants, as well as the coordinator, and thereby, is able to independently conclude the final decision. By reducing the time complexity to two rounds, it becomes less likely for a participant, in D2PL, to be blocked during commit processing in the case of a coordinator's failure.

There are four transaction type specific 2PC protocols, all of which, when applicable, improve both the message and time complexities of 2PC by eliminating the explicit voting phase of 2PC. The *unsolicited-vote* protocol (UV) (1979) shortens the voting phase of 2PC assuming that each participant knows when it has executed the last operation for a transaction. In this way, a participant sends its vote on its own initiative once it recognizes that it has executed the last operation for the transaction. When the coordinator receives the votes of the participants, it proceeds with the decision phase. The *early prepare* protocol (EP) (1990) combines UV with PrC without assuming that a participant can recognize the last operation of a transaction. Every operation is, therefore, treated as if it is the last operation executing at the participant and its acknowledgment is interpreted as a "yes" vote. This means that a participant has to force write its log each time it executes an operation so that it can preserve the global atomicity of the transaction after a system crash. In contrast to EP which reduces time and message complexities at the expense of log complexity, the *coordinator log* (CL) and *implicit yes-vote* (IYV) protocols do not require force writing the log records, at the participants' sites, after the execution of each operation. Instead, they replicate the participants' logs at the coordinators' site. Hence, reducing log complexity compared to EP at the expense, however, of slower recovery. In CL, a participant does not maintain a

local stable log and, therefore, it has to contact all coordinators in the system in order to recover after a failure. Moreover, a participant may need to contact the coordinator of a transaction during normal processing to maintain *write-ahead logging* (WAL) or to undo the effects of an aborting transaction. This is because the log of a participant is scattered across the coordinators' sites. In contrast, the log of a participant in IYV is partially replicated across the coordinators' sites. That is, only the *redo* records are replicated at the coordinators' sites while the *undo* records are stored locally. Thus, a participant, in IYV, never communicates with any coordinator to maintain WAL or to undo the effects of aborting transactions. Thus, in IYV, the replicated records are used only to recover a participant after a system's crash. Furthermore, IYV is based on PrA while CL is derived from PrC.

Existing 2PC variants are incompatible with each other and need to be made to interoperate in order to be integrated in (heterogeneous) multidatabase systems and the Internet. Thus, the continued research for more efficient ACPs has expanded to include the investigation of integrated ACPs. Efforts in this direction include the Harmony prototype system that integrates *centralized* participants that use centralized (asymmetric) ACPs with *centralized* participants that use decentralized (symmetric) ACPs (1991), the integration of *distributed* participants that use symmetric ACPs with *distributed* participants that use asymmetric ACPs (1994), and the *presumed any* protocol (1996) that integrates participants that use 2PC, PrA, or PrC. Besides that, recent efforts are targeted towards understanding the sources of incompatibilities [1] and the integration of ACPs in an adaptive manner to achieve higher system performance [8].

Several optimizations have been proposed that can reduce the costs associated with ACPs [5,2]. These include the *read-only, last agent, group commit, sharing the log, flattening the transaction tree* and *optimistic* optimizations. The read-only optimizations can be considered as the most significant ones, given that read-only transactions are the majority in any general database system. In fact, the performance gains allowed by the *traditional read-only* optimization provided the argument in favor of PrA to become the current choice of ACPs in the ISO OSI-TP (1998) and X/Open DTP (1996) distributed transaction processing standards, and commercial systems. The basic idea behind the read-only optimizations is that a read-only participant,

a participant that has not performed any updates on behalf of a transaction, can be excluded from the decision phase of the transaction. This is because it does not matter whether the transaction is finally committed or aborted at a read-only participant to ensure the transaction's atomicity. In the traditional read-only optimization [4], a read-only participant votes "read-only" instead of a "yes" and immediately releases all the resources held by the transaction without writing any log records. A "read-only" vote allows a coordinator to recognize and discard the read-only participant from the rest of the protocol. The *unsolicited update-vote* (1997) is another read-only optimization that further reduces the costs associated with read-only participants. Not only that, but it incurs the same costs when used with both PrA and PrC, supporting the arguments for PrC to be also included in the standards.

The *last agent* optimization has been implemented by a number of commercial systems to reduce the cost of commit processing in the presence of a *single remote* participant. In this optimization, a coordinator first prepares itself and the nearby participants for commitment (fast first phase), and then delegates the responsibility of making the final decision to the remote participant. This eliminates the voting phase involving the remote participant. This same idea of delegating part of commitment (i.e., transferring the commit responsibilities) from one site to another has been also used to reduce blocking, for example, in open commit protocols (1990) and IYV with a commit coordinator (1996).

The *group commit* optimization has been also implemented by a number of commercial products to reduce log complexity. In the context of centralized database systems, a commit record pertaining to a transaction is not forced on an individual basis. Instead, a single force write to the log is performed when a number of transactions are to be committed or when a timer has expired. In the context of distributed database systems, this technique is used at the participants' sites *only* for the commit records of transactions during commit processing. The *lazy commit* optimization is a generalization of the *group commit* in which not only the commit records at the participants are forced in a group fashion, but *all* log records are lazily forced written onto stable storage during commit processing. Thus, the cost of a single access to the stable log is amortized among several transactions. The s*haring of*

log between the transaction manager and data managers [5] at a site is another optimization that takes advantage of the sequential nature of the log to eliminate the need of force writing the log by the data managers.

The *flattening of the transaction tree* optimization is targeted for the tree-of-processes transaction model and is a big performance winner in distributed transactions that contain deep trees. It can reduce both the message and log complexities of an ACP by transforming the transaction execution tree of *any* depth into a two-level commit tree at commit initiation time. In this way, the root coordinator sends coordination messages directly to, and receives messages directly from, any participant. Thus, avoiding propagation delays and sequential forcing of log records. *Restructuring-the-commit-tree-around-update-participants* (RCT-UP) is an enhancement to the flattening technique that flattens only update participants (participants that have executed update operations on behave of the transaction), thereby, connecting them directly to the coordinator while leaving read-only participants connected in a multi-level manner. This is to reduce the effects of the communication delays on the overall system performance in systems that do not support simultaneous message multicasting to all participants.

Another optimization is *optimistic* (OPT) (1997) which can enhance the overall system performance by reducing blocking arising out of locks held by prepared transactions. OPT shares the same assumption as PrC, that is, transactions tend to commit when they reach their commit points. Under this assumption, OPT allows a transaction to *borrow* data that have been modified by another transaction that has entered a prepared-to-commit state and has not committed. A borrower is aborted if the lending transaction is finally aborted.

Key Applications

3PC has never been implemented in any commercial database system due to its cost, during normal transaction processing, compared to the other ACPs. This is besides its implementation complexity and, especially, the complexity of its termination protocol. Even with the added implementation complexity and cost, 3PC does not *completely* eliminate blocking since it is still susceptible to blocking in case of network partitioning. In fact there is no ACP that is non-blocking in the case of site as well as communication failures. This is an

inherent characteristic of the *Byzantine Generals Problem*, the more general problem of atomic commitment. However, the protocol remains an instrumental theoretical result for understanding the behavior of ACPs and the limitations in solving the atomic commitment problem.

Cross-references
► Atomicity
► Distributed Database Systems
► Distributed Recovery
► Distributed Transaction Management

Recommended Reading
1. Al-Houmaily Y. Incompatibility dimensions and integration of atomic commit protocols. Int. Arab J. Inf. Technol., 5(4):2008.
2. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, Chapter 13. In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Upper Saddle River, NJ, 1998, pp. 370–416.
3. Lamport L., Shostak R., and Pease M. The byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
4. Mohan C., Lindsay B., and Obermarck R. Transaction Management in the R* Distributed Data Base Management System. ACM Trans. Database Syst., 11(4):378–396, 1986.
5. Samaras G., Britton K., Citron A., and Mohan C. Two-phase commit optimizations in a commercial distributed environment. Distrib. Parall. Databases, 3(4):325–361, 1995.
6. Skeen D. Non-blocking Commit Protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 133–142.
7. Skeen D. and Stonebraker M. A Formal model of crash recovery in a distributed system. IEEE Trans. Softw. Eng., 9(3):219–228, 1983.
8. Yu W. and Pu C. A Dynamic two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1), 2007.

Thresholding

► Image Segmentation

Tight Coupling

SERGUEI MANKOVSKII
CA Labs, CA Inc., Thornhill, ON, Canada

Synonyms
Strong coupling

Definition
Tight coupling indicates strong dependency between software components. The dependency is strong in the sense that two or more components have multiple and complex dependencies between internal states, data and functions of the components.

Key Points
Tight coupling often achieves high performance characteristics at the expense of flexibility and ease of maintenance. It is often justified for systems that are going to be used as a black box, with no expectation of ongoing maintenance or upgrade. The smaller a systems is, the more congruent its function is, the more likely it would justify tight coupling as a design principle. This is why one can find tight coupling in the components of larger systems. In this situation a number of tightly coupled components might be interacting using loosely coupled approach. This type of systems leads to robust designs where maintainability and flexibility is achieved by simply replacing one or more components. At the same time, each component performs in the best possible way because of tight coupling inside of it. This is the idea behind Service Oriented Architectures (SOA). This way SOA achieves balance between performance and flexibility. However it is not of the case for other architectures and because of that it is often beneficial to evaluate trade-offs of both approaches during system design.

Cross-references
► Loose Coupling
► SOA
► Tight Coupling

Time Aggregated Graphs

BETSY GEORGE, SHASHI SHEKHAR
University of Minnesota, Minnesota, Minneapolis, MN, USA

Synonyms
Spatio-temporal graphs; Time-dependent graphs; Time-dependent networks

Definition
A time aggregated graph [1,2] is a model that can be used to represent a spatio-temporal network. The topology and the attributes in a spatio-temporal

network are typically time-dependent. Time aggregated graphs aggregate the time-dependent attributes over edges and nodes. This model facilitates the computation of path queries on a network accounting for the time-dependence of network parameters.

Key Points

Graphs have been extensively used to model spatial networks; weights assigned to nodes and edges are used to encode additional information. For example, the travel time between two intersections in a road network can be represented by the weight of the edge connecting the nodes that represent the intersections. In a real world scenario, it is not uncommon for these network parameters to be time-dependent. A time aggregated graph is a model that can capture the time-dependence of network parameters. In addition, the model captures the possibility of edges and nodes being absent during certain instants of time.

The model represents the time-variance of attributes by modeling them as time series.

Spatio-temporal networks have critical applications in domains such as transportation science. Developing storage efficient models for such networks that support the design of efficient query processing algorithms is important. Time aggregated graphs provide a means to model spatio-temporal networks and formulate algorithms that honor the time dependence of spatial networks.

Cross-references

▶ Spatio-Temporal Graphs
▶ Spatio-Temporal Networks

Recommended Reading

1. George B. and Shekhar S. Time-aggregated graphs for modeling spatio-temporal networks – an extended abstract. In Proc. Workshops at Int. Conf. on Conceptual Modeling, 2006, pp. 85–93.
2. George B. Kim S. and Shekhar S. Spatio-temporal network databases and routing algorithms: a summary of results. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 460–477.

Time and Information Retrieval

OMAR ALONSO, MICHAEL GERTZ
University of California at Davis, Davis, CA, USA

Synonyms

Temporal information retrieval

Definition

Traditional information retrieval (IR) is concerned with models, algorithms, and architectures for the retrieval and ranking of documents from a document collection based on their relevance to search queries. In temporal information retrieval, expressions (words or phrases) that relate to instants in time, events, time periods, or other temporal descriptions are extracted from documents and handled in a special way to rank (and optionally group) the documents returned for a search query. Thus, in temporal information retrieval, temporal expressions extracted from documents play a special role in the overall relevance and in the organization and exploration of search results along timelines.

Historical Background

Research on using time information for retrieval and browsing activities is fairly recent. From a search perspective, there is previous work on placing search results in a timeline to facilitate the exploration of information [2,3,10]. A general overview of the basic idea of search result clustering using temporal document annotations obtained through a named-entity extraction approach has been outlined by Alonso and Gertz [4].

Research on temporal annotations has gained a lot of attention lately, and it is covered in great depth in the book edited by Mani et al. [8]. The work also includes discussions about tense and structural analysis and temporal reasoning techniques. The special issue on temporal information processing shows a wide range of current research directions and applications like question-answering and summarization [9].

News in particular have been the preferred information source for most of the related work in temporal information retrieval. Swan and Allan combine news topic detection and tracking with timelines as a browsing interface [14]. Automatic assignment of document event-time periods and automatic tagging of news messages using entity extraction is presented by Schilder and Habel in [11]. Their work also presents a temporal tagger along with its evaluation. There is very interesting work on adding time to applications like news for presenting temporal summaries as introduced by Allan et al. [1]. The work by Shaparenko et al. [12] concentrates on analyzing the development of a document collection over time and identifying temporal pattern. The work by Koen and Bender in Time Frames is one approach to augment news

articles by extracting time information [7]. Recently, new research has emerged for future retrieval proposed by Baeza-Yates [5] where the idea of exploiting temporal information is developed for searching the future.

Foundations

Overview and Motivation

Time is an important dimension of any information space and can be very useful in information retrieval. Time and time measurements can help in outlining a particular historical period or establishing the context of a document. As an alternative to document ranking techniques like those based on popularity, time can be valuable for placing search results in a timeline for document exploration purposes. Current information retrieval systems and applications, however, do not take advantage of all the time information available within documents to provide better search results and thus to improve the user experience.

A quick look at any of the current search engines and information retrieval systems shows that the temporal viewpoint is restricted to sorting the search result represented in a hit list by date only. The date attribute is mainly the creation or last modified date of a Web page or document. In some cases it can be misleading, because the timestamp provided by a Web server or any other document management system may not be accurate. Other search applications provide a range date search as part of the advanced search options. Still, the search results are filtered based on the date attribute. For search purposes, the time axis is mainly constructed using that type of document metadata.

Even simple queries against Web search engines show that oftentimes organizing the documents in a hit list along some timeline can be helpful. For example, a query for "soccer world cup" against search engines now returns mostly pointers to documents that cover the recent event in Germany. But every soccer fan knows that this event happens every four years. Another example is "Iraq war;" here, results are primarily related to the latest events with little from the 1990's war. Clearly, it would be useful if a tool on top of a traditional retrieval system is more aware of the temporal information embedded in the documents and allows the user to have search results presented in different ways based on the temporal information. For this, it is essential to extract temporal information

from documents and associate documents with points in time along well-defined timelines.

Time and Timelines

As the basis for associating points in time with documents, it is customary to assume a discrete representation of time based on the Gregorian Calendar, with a single day being an atomic time interval called a *chronon*. A base timeline, denoted T_d, is an interval of consecutive day chronons. For example, the sequence "March 12, 2002; March 13, 2002; March 14, 2002" is a contiguous subsequence of chronons in T_d. Contiguous sequences of chronons can be grouped into larger units called *granules*, such as weeks, months, years, or decades. A grouping based on a granule results in a more coarse-grained timeline, such as T_w based on weeks, T_m based on months, or T_y based on years. Examples of week chronons in T_w are "3rd week of 2005" or "last week of 2006." Depending on the type of underlying calendar, base timeline, and grouping of chronons, timelines of different time granularity can be constructed. Chronons from two timelines then can also be compared. For example, "March 18, 2002" (chronon in T_d) lies before "December 2006" (chronon in T_m). Timelines constructed in this way then serve as the basis to have temporal expressions in documents refer to chronons in one or more timelines.

Temporal Expressions

There is quite a lot of temporal information in any corpus of documents. For example, financial news tend to be rich in describing near future events; resume documents contain several references to the past in a very precise way; and project documentation involves phase milestones that are captured in time. However, what types of temporal information (besides a simple document timestamp) are there and how do they relate to timelines?

In general, with the textual content of a document, a set of *temporal entities* can be associated. A temporal entity describes a point in time, event, or time period at a conceptual level. The identification of such entities involves a linguistic analysis of the document, where approaches based on named-entity extraction determine so-called *temporal expressions*. A temporal expression is basically a sequence of tokens that represent an instance of a temporal entity. Contrary to other entities such as names and places, temporal entities can be represented as temporal expressions that are sequences of not necessarily contiguous tokens or

words. Expressions can be mapped to temporal entities that are defined in some time ontology. Similar to the approach by Schilder and Habel [11], this discussion distinguishes between explicit, implicit, and relative temporal expressions. *Explicit temporal expressions* directly describe entries in some timeline, such as an exact date or year. For example, the token sequences "December 2004" or "September 12, 2005" in a document are explicit temporal expressions and can be mapped directly to chronons in a timeline (here T_m and T_d, respectively).

Depending on the underlying time ontology and capabilities of the named-entity extraction approach, even apparently imprecise temporal information, such as names of holidays or events can be anchored in a timeline. For example, the token sequence "Columbus Day 2006" in the text of a document can be mapped to the expression "October 12, 2006," or the sequence "Labor Day 2008" can be mapped to "September 1, 2008." Such types of temporal expressions whose mapping to entities relies on the capability of the underlying time ontology are called *implicit temporal expressions*.

Relative temporal expressions represent temporal entities that can only be anchored in a timeline in reference to another explicit or implicit, already anchored temporal expression (which, in the worst case, is the document timestamp). For example, the expression "today" alone cannot be anchored in any timeline. However, it can be anchored if the document is known to have a creation date. This date therefore can be used as a reference for that expression, which then can be mapped to a chronon. There are many instances of relative temporal expressions, such as the names of weekdays (e.g., "on Thursday") or months (e.g., "in July") or references to such points in time like "next week" or "last Friday."

Given a document collection $\mathcal{D}$, the temporal expressions that have been determined for each document $d \in \mathcal{D}$ can be represented in the form of a *temporal document profile*. For example, a profile then records for each type of expression the token sequence, position of the sequence in the document d, and the chronon to which the expression has been mapped. With a document, several expressions of different types and corresponding chronons (in different timelines) can be associated. The same expression and chronon can even appear several times in the same document, but then at different positions in the document. Each

document at least has one explicit temporal expression, which is assumed to be the document timestamp. Next, an extraction approach for temporal expressions using existing tools is outlined.

Temporal Processing Pipeline

Given a document collection $\mathcal{D}$, the identification of the temporal expressions in each document $d \in \mathcal{D}$ is realized through a *document processing pipeline*, which includes a sequence of operations as follows. The first step is to extract the timestamp from the document. This can be the creation or last modified date for a file. In case of a Web page, one can rely on the information provided by the Web server. The second step is to run a part of speech tagger (POS tagger) on every document. A POS tagger returns the document with parts of speech assigned to each word/token like noun, verb etc. The tagger also tags sentence delimiters that later are needed for extracting the temporal expressions. The third step is to run a temporal expression tagger on the POS-tagged version of the document, which recognizes the extents and normalized values of temporal expressions [6]. This step extracts temporal expressions based on the TimeML standard and produces an XML document. The TimeML specification for temporal annotations seems to be a suitable approach here, because it has emerged as the standard markup language for events and temporal expressions in natural language [15]. The resulting XML document is the original document annotated by various information about the temporal expressions that have been determined for the original document. This information then can be used to construct the temporal document profile for the document, which, in turn, can be used for different time-centric document information retrieval and exploration approaches.

Document Retrieval

A fundamental property of any information retrieval system is the ability to help users find documents that satisfy their information needs. At a first glance, this looks pretty obvious but it is not, because users are not very expressive in describing what information they want. Furthermore, the information needs they specify can be ambiguous, making the retrieval task even harder. A user will judge whether or not a query result satisfies her information needs based on whether she

considers the result to the search query relevant. The central idea underlying a temporal information retrieval approach is to utilize the temporal expressions that have been determined for each document in a given document collection $\mathcal{D}$ in order to group and/or rank search results based on the temporal information embedded in the documents. By using this approach, time plays a central role in the overall quality of the search results. Assume a standard information retrieval or search application that returns a hit list of n documents $L_q = \langle d_1,...,d_n \rangle$ for a search query q. The search application retrieves the result based on the relevance of the documents with respect to q using traditional metrics based on tf/idf and the distance of the query terms to the first token of temporal expressions in the documents. After all, *tense* happens at the sentence level so it is important to detect these "boundaries" with respect to the query q. There are several ways in which the temporal expressions in the documents in L_q can be used to group the documents using temporal aspects. In the following, only the general idea of these approaches is illustrated. For example, the following algorithm outlines how the usage of temporal expressions can help to group search results based on the temporal expressions determined from the documents in L_q.

1. Determine the document hit list $L_q = \langle d_1,...,d_n \rangle$ that satisfies the search query q, sorted by relevance of the documents.
2. Determine the temporal expressions $T = \{te_1,...,te_m\}$ for all the n documents in the hit list. Note that a document $d_i \in L_q$ can have several temporal expressions, represented in d_i's temporal document profile.
3. Choose a type of time granule g (e.g., year or month). Sort the temporal expressions in T using that granule as key. For example, the expression "September 1, 2007" then comes before the expression "August 2007," assuming a descending order.
4. For each granule of type g (e.g., a particular year or month), take all those documents from L_q that contain a temporal expression covered by that granule. Rank these documents using the distance between the query terms in q to the temporal expressions.
5. Display document groups using the granule type g as label in a timeline fashion in descending order.

For example, if the granule type year has been chosen, for each year, there is a group of documents that contain a temporal expression related to that year (perhaps at a finer level of granularity). Note that instances of the granule type are based on a timeline.

If a document $d \in L_q$ contains several temporal expressions, this document can end up in different groups and at a different rank in each group. In general, the above approach organizes documents from a hit list along a timeline based on a time granule type. A group of documents related to a particular instant of a granule then can be organized further, i.e., based on a more fine-grained time granule.

Key Applications

Recently, exploratory search system have emerged as a specialization of information exploration to support serendipity, learning, and investigation, and, more generally, to allow users to browse available information. For such systems, taking advantage of temporal expressions embedded in the documents leads to a much richer framework for exploration. This is an important ingredient for the information forager who is trying to see the profit in terms of the interaction cost required to gain useful information from an information source. Users tend to prefer sources, in this case search engines, that are richer in good results. These good results involve adding important nuggets such as temporal information and relationships. The news domain is another area where search, presentation, and filtering can be greatly improved by using more temporal information. As an example, financial news about a company's Q4 earnings has a very precise meaning in a time-related context. Exploring and analyzing news by these types of temporal expressions can be very useful for particular application domains, for example, in the area of financial analysis.

Timeline-Based Exploration

Current interfaces to search engines typically present search results sorted by the relevance of documents from a document collection to a search query. For this, the freshness of the information is considered an important part of the quality of the result. Temporal attributes in Web pages or documents such as date,
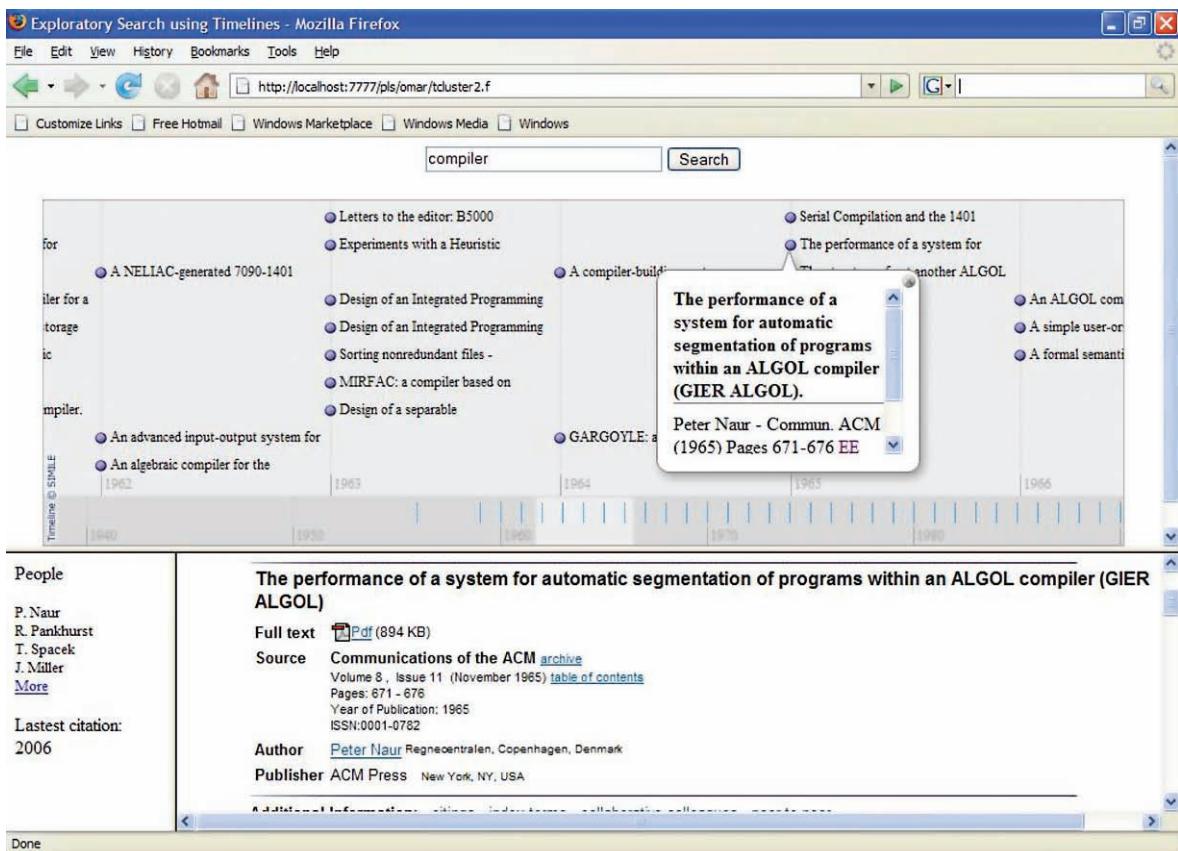
however, are just viewed as some structured criteria to sort the result in descending order of relevance. At the time of this writing there is a new experimental feature for timelines as part of Google (view:timeline).

Another exploratory search prototype outlined in the following is a Web-based application that uses the SIMILE timeline toolkit for visualization and exploration purposes [13], here with respect to a document collection of journal articles from DBLP (http://www.informatik.uni-trier.de/~ley/db/). The interface is organized as follows. The main section takes half of the screen and contains the search box and the timeline. The timeline consists of two bands that represent different time scales (types of granules): decade and year. Both bands are synchronized such that panning one band also scrolls the other. The lower band (decade) is much smaller since the goal is to show the activity in a decade. The upper band shows all articles in a given year. Figure 1 shows the exploratory search interface in action for the query "compiler" against the collection. The system retrieves all journal articles that contain the term "compiler" in the title and returns a hit list clustered by year (in which the article appeared). Search results are anchored in the timeline, that is, documents (or rather the embedded temporal expressions) are linked to instants in time. If more than one article falls within a year, the order of the documents in such a group is based on each document's relevance to the query. Obviously, such an information exploration and visualization approach and tool on top of a more traditional search application can be extremely valuable in the context of temporal information retrieval.

Future Directions

Evaluations of the impact of temporal attributes for search and retrieval are needed to asses the importance of these techniques. This, of course, can only be

Time and Information Retrieval. Figure 1. Exploring research articles using the SIMILE timeline toolkit.

applied to those applications where the usage of time information has some (expected) benefit.

Cross-references

► Document Clustering
► Information Extraction
► Information Retreival
► Structured Document Retrieval
► Web Search Relevance Ranking

Recommended Reading

1. Allan J., Gupta R., and Khandelwal V. Temporal summaries of news topics. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 10–18.
2. Allen R.B. A focus-context browser for multiple timelines. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2005, pp. 260–261.
3. Alonso O., Baeza-Yates R., and Gertz M. Exploratory search using timelines. In Proc. SIGCHI 2007 Workshop on Exploratory Search and HCI Workshop, 2007.
4. Alonso O. and Gertz M. Clustering of search results using temporal attributes. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 597–598.
5. Baeza-Yates R. Searching the future. In ACM SIGIR 2005 Workshop on Mathematical/Formal Methods in Information Retrieval, 2005.
6. GUTime. Available at: http://complingone.georgetown.edu/~linguist
7. Koen D.B. and Bender W. Time frames: temporal augmentation of the news. IBM Syst. J., 39(3–4):597–616, 2000.
8. Mani I., Pustejovsky J., and Gaizauskas R. (eds.). The Language of Time. Oxford University Press, New York, NY, USA, 2005.
9. Mani I., Pustejovsky J., and Sundheim B. Introduction to the Special Issue on Temporal Information Processing. ACM Trans. Asian Lang. Inform. Process., 3(1):1–10, March 2004.
10. Ringel M., Cutrell E., Dumais S.T., and Horvitz E. Milestones in time: the value of landmarks in retrieving information from personal stores. In Proc. IFIP TC13 Int. Conf. on Human-Computer Interaction, 2003, pp. 184–191.
11. Schilder F. and Habel C. From temporal expressions to temporal information: semantic tagging of news messages. In Proc. ACL 2001 Workshop on Temporal and Spatial Information Processing, 2001.
12. Shaparenko B., Caruana R., Gehrke J., and Joachims T. Identifying temporal patterns and key players in document collections. In Proc. IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications, 2005, pp. 165-174.
13. SIMILE Timeline toolkit. Available at: http://simile.mit.edu/timeline/
14. Swan R. and Allan J. Automatic generation of overview timelines. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 49–56.
15. TimeML, markup language for temporal and event expressions. Available at: http://www.timeml.org/

Time Dependent Geometry

► Moving Object

Time Distance

► Time Interval
► Time Span

Time Domain

ANGELO MONTANARI[1], JAN CHOMICKI[2]
[1]University of Udine, Udine, Italy
[2]State University of New York at Buffalo, Buffalo, NY, USA

Synonyms

Temporal domain; Temporal structure

Definition

In its full generality, a time domain can be defined as a set of *temporal individuals* connected by a set of *temporal relations*. Different choices for the temporal individuals and/or the temporal relations give rise to different temporal ontologies.

In the database context, the most common temporal ontology takes *time instants* (equivalently, points or moments) as the temporal individuals and a *linear order* over them as the (unique) temporal relation [5]. In addition, one may distinguish between discrete and dense, possibly continuous, time domains and between bounded and unbounded time domains. In the discrete case, one may further consider whether the time domain is finite or infinite and, in the case of

unbounded domains, one can differentiate between left-bounded, right-bounded, and totally unbounded domains. Moreover, besides linear time, one may consider *branching time*, where the linear order is replaced with a partial one (a tree or even a directed acyclic graph), or circular time, which can be used to represent temporal periodicity.

As for temporal individuals, time instants can be replaced with *time intervals* (equivalently, periods or anchored stretches of time) connected by (a subset of) Allen's relations `before`, `meets`, `overlaps`, `starts`, `during`, `equal`, and `finishes`, and their inverses or suitable combinations [7]. As in the case of instant-based domains, one may distinguish between discrete and dense domains, bounded and unbounded domains, linear, branching, and circular domains, and so on.

Finally, as most temporal database applications deal with both qualitative and quantitative temporal aspects, instant-based time domains are usually assumed to be isomorphic to specific numerical structures, such as those of natural, integer, rational, and real numbers, or to fragments of them, while interval-based ones are obtained as suitable intervallic constructions over them. In such a way, time domains are endowed with *metrical features*.

Historical Background

The nature of time and the choice between time instants and time intervals as the primary objects of a temporal ontology have been a subject of active philosophical debate since the times of Zeno and Aristotle. In the twentieth century, major contributions to the investigation of time came from a number of disciplines. A prominent role was played by Prior who extensively studied various aspects of time, including axiomatic systems of tense logic based on different time domains.

Nowadays, besides physics, philosophy, and linguistics, there is a considerable interest in temporal structures in mathematics (theories of linear and branching orders), artificial intelligence (theories of action and change, representation of and reasoning with temporal constraints, planning), and theoretical computer science (specification and verification of concurrent and distributed systems, formal analysis of hybrid temporal systems that feature both discrete and continuous components). A comprehensive study and logical analysis of instant-based and interval-based temporal ontologies, languages, and logical systems can be found in [2].

As for *temporal databases*, the choice of the time domain over which temporal components take their value is at the core of any application. In most cases, a discrete, finite, and linearly ordered (instant-based) time domain is assumed. This is the case, for instance, with SQL standards [9]. However, there is no single way to represent time in a database, as witnessed by the literature in the field. To model when something happened, time instants are commonly used; validity of a fact over time is naturally represented by the (convex) set of time instants at which the fact holds, the time *period of validity* in the temporal database terminology; finally, to capture processes as well as some kinds of temporal aggregation, time intervals are needed.

Foundations

Basics

The choice between time instants and time intervals as the basic time constituents is a fundamental decision step that all temporal systems have in common. In mathematics, the choice of *time instants*, that is, points in time without duration, is prevalent. Although quite abstract, such an solution turned out extremely fruitful and relatively easy to deal with in practice. In computer science, additional motivations for this choice come from the natural description of computations as possibly infinite sequences of instantaneous steps.

The alternative option of taking *time intervals*, that is, anchored stretches of time with duration, as temporal individuals seems to better adhere to the concrete experience of people. Physical phenomena as well as natural language expressions involving time can be more easily described in terms of time intervals instead of time instants. Nevertheless, the complexity of any systematic treatment of time intervals prevents many systems from the adoption of an interval-based ontology.

The instant and the interval ontologies are systematically investigated and compared in [2]. The author identifies the conditions an instant-based (resp., interval-based) structure must satisfy to be considered as an adequate model of time. Then, through an axiomatic encoding of such conditions in an appropriate language, he provides a number of (first-order and higher order) logical theories of both instant-based and interval-based discrete, dense, and continuous structures. Finally, he illustrates the strong connections that link the

two time ontologies. In particular, he shows how interval-based temporal structures can be obtained from instant-based ones through the standard process of interval formation and how instant-based temporal structures can be derived from interval-based ones by a (non-trivial) limiting construction.

A metric of time is often introduced to allow one to deal with time distance and/or duration. In particular, a time metric is needed to define calendar times, such as those based on the commonly used Gregorian calendar.

Temporal Models and Query Languages

The choice of the time domain has an impact on various components of temporal databases. In particular, it influences temporal data models and temporal query languages.

As for *temporal data models*, almost all of them adopt an instant-based time ontology. Moreover, most of them assume the domain to be linear, discrete and finite. However, many variants of this basic structure have been taken into consideration [8]. Right-unbounded domains have been used to record information about the future. Dense and continuous domains have been considered in the context of temporal constraint databases, that allow one to represent large, or even infinite, sets of values, including time values, in a compact way. Branching time has been exploited in applications where several alternatives have to be considered in the future and/or past evolution of temporal data.

Many data models distinguish between absolute (anchored) and relative (unanchored) time values. Absolute time values denote specific temporal individuals. In general, they are associated with a time metric, such as that of calendar times. As an example, the 14th of September 2007 is an absolute time value that denotes a specific element of the domain of days in the Gregorian calendar. Relative time values specify the distances between pairs of time instants or the durations of time intervals. Absolute and relative time values can also be used in combination. As an example, the expression 7 days after the 14th of September 2007 denotes the 21st of September 2007.

As for *temporal query languages*, they typically assume that time is isomorphic to natural numbers. This is in agreement with the most common, *linear-time* dialect of temporal logic. In temporal constraint databases, however, the use of classical query languages like relational calculus or algebra accommodates a variety of time domains, including dense and continuous ones.

Time Domain and Granularity

Despite its apparent simplicity, the addition of the notion of time domain to temporal databases presents various subtleties. The main ones concern the nature of the elements of the domain. As soon as calendar times come into play, indeed, the abstract notion of instant-based time domain must be contextualized with respect to a specific *granularity* [3,4]. Any given granularity can be viewed as a suitable abstraction of the real time line that partitions it into a denumerable sequence of homogeneous stretches of time. The elements of the partition, granules in the temporal database terminology, become the individuals (non-decomposable time units) of a discrete time domain. With respect to the considered granularity, these temporal individuals can be assimilated to time instants. Obviously, if a shift to a finer granularity takes place, e.g., if one moves from the domain of months to the domain of days, a single granule must be replaced with a set of granules. In such a way, being instantaneous is not more an intrinsic property of a temporal individual, but it depends on the time granularity one refers to. A detailed analysis of the limitations of the temporal database management of instant-based time domains can be found in [9].

The Association of Time with Data

The association of the elements of the time domain with data is done by *timestamping*. A timestamp is a time value associated with a data object. In the relational setting, one distinguishes between attribute-time-stamped data models, where timestamps are associated with attribute values, and tuple-timestamped data models, where timestamps are associated with tuples of values. As a third possibility, a timestamp can be associated with an entire relation/database.

Timestamps can be single elements as well as sets of elements of the time domain. Time instants are usually associated with relevant events, e.g., they can be used to record the day of the hiring or of the dismissal of an employee. (Convex) sets of time instants are associated with facts that hold over time. As an example, if a person E works for a company C from the 1st of February 2007 to the 31st of May 2007,

one keeps track of the fact that every day in between the 1st of February 2007 and the 31st of May 2007, endpoints included, E is an employee of C.

Time intervals are needed to deal with situations where validity over an interval cannot be reduced to validity over its subintervals (including point sub-intervals) [10]. This is the case with processes that relate to an interval as a whole, meaning that if a process consumes a certain interval it cannot possibly transpire during any proper subinterval thereof. Exam-ples are the processes of baking a cake or of flying from Venice to Montreal. This is also the case when validity of a fact at/over consecutive instants/intervals does not imply its validity over the whole interval. As an exam-ple, two consecutive phone calls with the same values are different from a single phone call over the whole period. The same happens for some kinds of temporal aggregation [4]. Finally, the use of time intervals is common in several areas of AI, including knowledge representation and qualitative reasoning, e.g., [1].

It is important to avoid any confusion between this latter use of intervals as timestamps and their use as compact representations of sets of time points (time periods in the temporal database literature). Time inter-vals are indeed often used to obtain succinct represen-tations of (convex) sets of time instants. In such a case, validity over a time period is interpreted as validity at every time instant belonging to it. As an example, the fact that a person E worked for a company C from the 1st of February 2007 to the 31st of May 2007 can be repre-sented by the tuple (E, C, [2007/02/01, 2007/05/31]) meaning that E worked for C every day in the closed interval [2007/02/01, 2007/05/31].

Key Applications

As already pointed out, the time domain is an essential component of any temporal data model, and thus its addition to SQL standards does not come as a surprise.

In SQL, time domains are encoded via *temporal data types* (they have been introduced in SQL-92 and preserved in SQL:1999). In SQL-92, five (anchored) time instant data types, three basic forms and two variations, are supported (DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE). In addition, SQL-92 features two (unanchored) data types that allow one to model positive (a shift from an instant to a future one) and negative (a shift from an instant to a past one) distances between instants. One can be used to specify distances in terms of years and months

(the YEAR-MONTH INTERVAL type), the other to specify distances in terms of days, hours, minutes, sec-onds, and fractions of a second (the DAY-TIME INTER-VAL type). As a matter of fact, the choice of using the word interval to designate a time distance instead of a temporal individual – in contrast with the standard use of this word in computer science – is unfortunate, because it confuses a derived element of the time domain (the interval) with a property of it (its duration). An additional (unanchored) temporal data type, called PERIOD, was included in the SQL/Temporal proposal for the SQL3 standard, which was eventually withdrawn. A period is a convex sets of time instants that can be succinctly represented as a pair of time instants, namely, the first and the last instants with respect to the given order.

SQL also provides *predicates*, *constructors*, and *functions* for the management of time values. General predicates, such as the equal-to and less-than predicates, can be used to compare pairs of comparable values of any given temporal type; moreover, the specific overlap predicate can be used to check whether two time periods overlap. Temporal constructors are expressions that re-turn a temporal value of a suitable type. It is possible to distinguish datetime constructors, that return a time instant of one of the given data types, and interval con-structors, that return a value of YEAR-MONTH INTER-VAL or DAY-TIME INTERVAL types. As for functions, they include the datetime value functions, such as the CURRENT_DATE function, that return an instant of the appropriate type, the CAST functions, that convert a value belonging to a given (temporal or non temporal) source data type into a value of the target temporal data type, and the extraction functions, that can be used to access specific fields of instant or interval time values.

Future Directions

Despite the strong prevalence of instant-based data mod-els in current temporal databases, a number of interesting problems, such as, for instance, that of temporal aggrega-tion, motivate a systematic study and development of *interval-based data models*. Moreover, in both instant-based and interval-based data models intervals are de-fined as suitable sets of elements of an instant-based time domain. The possibility of assuming time intervals as the primitive temporal constituents of the temporal domain is still largely unexplored. Such an alternative deserves a serious investigation.

Cross-references

Recommended Reading

1. Allen J. and Ferguson G. Actions and events in interval temporal logic. J. Logic Comput., 4(5):531–579, 1994.
2. van Benthem J. The Logic of Time. A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, 2nd edn. Kluwer, Dordrecht, Holland, 1991.
3. Bettini C., Jajodia S., and Wang X.S. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, NJ, USA, 2000.
4. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data? In Proc. 13th Int. Symp. Temporal Representation and Reasoning, 2006, pp. 121–136.
5. Chomicki J. and Toman D. Temporal databases. In Chapter 14 of the Handbook of Temporal Reasoning in Artificial Intelligence, M. Fisher, D. Gabbay, L. Vila (eds.). Elsevier B.V., Amsterdam, The Netherlands, 2005, pp. 429–467.
6. Euzenat J. and Montanari A. Time granularity. In Chapter 3 of the Handbook of Temporal Reasoning in Artificial Intelligence, M. Fisher, D. Gabbay, L. Vila (eds.). Elsevier B.V., Amsterdam, The Netherlands, 2005, pp. 59–118.
7. Goranko V., Montanari A., and Sciavicco G. A road map of interval temporal logics and duration calculi. J. Appl. Non-Class. Logics, 14(1-2):9–54, 2004.
8. Montanari A. and Pernici B. Temporal reasoning. In Chapter 21 of Temporal Databases: Theory, Design and Implementation, A. Tansell, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R. Snodgrass (eds.). Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 534–562.
9. Snodgrass R.T. Developing time-oriented database applications in SQL. In Chapter 3 of Instants and Intervals. Morgan Kauffman, San Francisco, CA, USA, 2000, pp. 24–87.
10. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal databases: a treatment of the telic/atelic distinction. IEEE Trans. Knowl. Data Eng., 16(5):540–551, 2004.

Time Granularity

Time in Philosophical Logic

Peter Øhrstrøm, Per F. V. Hasle
Aalborg University, Aalborg, Denmark

Synonyms

Temporal logic; Logic of time

Definition

The aim of the study of time in philosophical logic is to provide a conceptual framework for an interdisciplinary study of the nature of time and to formalize and study various conceptions and systems of time. In addition, the introduction of time into logic has led to the development of formal systems, which are particularly well suited to represent and study temporal phenomena such as program execution, temporal databases, and argumentation in natural language.

Historical Background

The philosophy of time is based on a long tradition, going back to ancient thought. It is an accepted wisdom within the field that no attempt to clarify the concept of time can be more than an accentuation of some aspects of time at the expense of others. Plato's statement that time is the "moving image of eternity" and Aristotle's suggestion that "time is the number of motion with respect to earlier and later" are no exceptions (see [17]). According to St. Augustine (354–430) time cannot be satisfactorily described using just one single definition or explanation: "What, then, is time? If no one asks me, I know: if I wish to explain it to one that asketh, I know not." [5, p. 40] Time is not definable in terms of other concepts. On the other hand, according to the Augustinian insight, all human beings have a tacit knowledge of what time is. In a sense, the endeavor of the logic of time is to study important manifestations and structures of this tacit knowledge.

There were many interesting contributions to the study of time in Scholastic philosophy, e.g., the analysis of the notions of beginning and ending, the duration of the present, temporal ampliation, the logic of "while," future contingency, and the logic of tenses. Anselm of Canterbury (ca. 1033–1109), William of Sherwood (ca. 1200–1270), William of Ockham (ca. 1285–1349), John Buridan (ca. 1295–1358), and Paul of Venice (ca. 1369–1429) all contributed significantly to the development of the philosophical and logical analysis of

time. With the Renaissance, however, the logical approach to the study of time fell into disrepute, although it never disappeared completely from philosophy.

However, the twentieth century has seen a very important revival of the philosophical study of time. The most important contribution to the modern philosophy of time was made in the 1950s and 1960s by A. N. Prior (1914–1969). In his endeavors, A. N. Prior took great inspiration from ancient and medieval thinkers and especially their work on time and logic.

The Aristotelian idea of time as the number of motion with respect to earlier and later actually unites two different pictures of time, the dynamic and the static view. On the one hand, time is linked to motion, i.e., changes in the world (the flow of time), and on the other hand time can be conceived as a stationary order of events represented by numbers. In his works, A. N. Prior logically analyzed the tension between the dynamic and the static approach to time, and developed four possible positions in regard to this tension. In particular, A. N. Prior used the idea of branching time to demonstrate that there is a model of time which is logically consistent with his ideas of free choice and indeterminism. (See [8, 189 ff.].)

After A. N. Prior's development of formalised temporal logic, a number of important concepts have been studied within this framework. In relation to temporal databases the studies of the topology of time and discussions regarding time in narratives are particularly interesting.

Foundations

In the present context, the following four questions regarding time in philosophical logic seem to be especially important:

1. What is the relation between dynamic and static time?
2. What does it mean to treat time as "branching"?
3. What is the relation between punctual and durational time (i.e., instants and durations)?
4. What is the role of time in storytelling (narratives)?

In the following, a brief introduction to each of these issues will be given.

Dynamical and Static Time: A-Theory vs. B-Theory

The basic set of concepts for the *dynamic* understanding of time are past, present, and future. In his very influential analysis of time the philosopher John Ellis

McTaggart (1866–1925) suggested to call these concepts (i.e., the tenses) the A-concepts. The tenses are well suited for describing the flow of time, since the future will become present, and the present will become past, i.e., flow into past. The basic set of concepts for the *static* understanding of time are before/after and "simultaneous with." Following McTaggart, these are called the B-concepts, and they seem especially apt for describing the permanent and temporal order of events. The two kinds of temporal notions can give rise to two different approaches to time. First, there is the dynamic approach (the A-theory) according to which the essential notions are past, present and future. In this view, time is seen "from the inside." Secondly, there is the static view of time (the B-theory) according to which time is understood as a set of instants (or durations) ordered by the before-after relation. Here time is seen "from the outside." It may be said to be a God's eye-perspective on time.

There is also an ontological difference between the two theories. According to the A-theory the tenses are real whereas the B-theorists consider them to be secondary and unreal. According to the A-theory the Now is real and objective, whereas the B-theories consider the Now to be purely subjective.

The debate between proponents of the two theories received a fresh impetus with A. N. Prior's formal analysis of the problem. (See [9, 216 ff.].). According to the B-theory, time is considered to be a partially ordered set of instants, and propositions are said to be true or false at the instants belonging to the set. According to the A-theory, time is conceived in terms of the operators P (Past) and F (Future), which are understood as being relative to a "Now." A. N. Prior suggested a distinction between four possible grades of tenselogical involvement corresponding to four different views of how to relate the A-notions (past, present and future) to the B-notions ("earlier than"/"later than," "simultaneous with"):

1. The B-notions are more fundamental than the A-notions. Therefore, in principle, the A-notions have to be defined in terms of the B-notions.
2. The B-notions are just as fundamental as the A-notions. The A-notions cannot be defined in terms of the B-notions or vice versa. The two sets of notions have to be treated on a par.
3. The A-notions are more fundamental than the B-notions. All B-notions can be defined in terms

of the A-notions and a primitive notion of temporal possibility.

4. The A-notions are more fundamental than the B-notions. Therefore, in principle the B-notions have to be defined in terms of the A-notions. Even the notion of temporal possibility can be defined on terms of the A-notions.

A. N. Prior's four grades of tense-logical involvement represent four different views of time and also four different foundations of temporal logic. In fact, theory 1 is the proper B-theory and theory 3 and 4 are versions of the proper A-theory. Theory 2 is a kind of intermediate theory.

In theory 1, the tense operators, P (past) and F (future), can be introduced in the following way:

$$T(t, Fq) \equiv _{def} \exists t_1 : t < t_1 \wedge T(t_1, q)$$

$$T(t, Pq) \equiv _{def} \exists t_1 : t_1 < t \wedge T(t_1, q)$$

where $T(t, q)$ is read "q is true at t," and $t < t_1$ is read "t is before t_1."

In theory 3 and 4, A. N. Prior has shown how instants can be introduced as maximally consistent sets of tense-logical propositions and how the before-after relation can be consistently defined in terms of tense-logical concepts (i.e., A-notions).

From a B-theoretical viewpoint, at any instant, an infinite number of propositions, including tensed ones, will be true about that instant. But from the A-theoretical point of view, precisely the infinite conjunction of the propositions in this set is a construction which, when called an "instant," makes the B-theoretical notion of "instant" secondary and derivable.
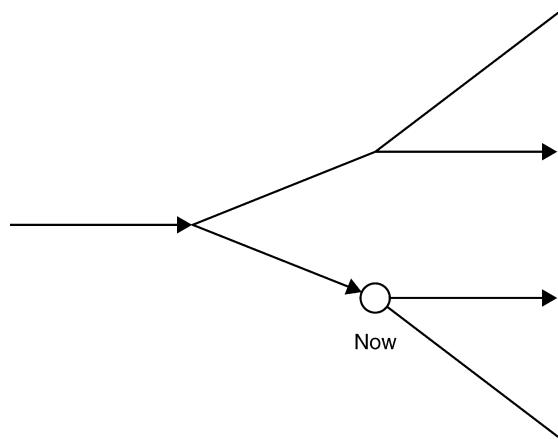
It should be noted, that whereas the A-theorist (theory 3 or 4) can translate any B-statement into his language, many A-statements cannot be translated into the B-language. For instance, there is no way to translate the A-statement "it is raining now in Aalborg" into the B-language. The "now" cannot be explained in terms of the B-language consisting of an ordered set of instants and the notion of a proposition being true at an instant. This asymmetry seems to be a rather strong argument in favor of the A-theory (i.e., A. N. Prior's theory 3 or 4).

Linear vs. Branching Time
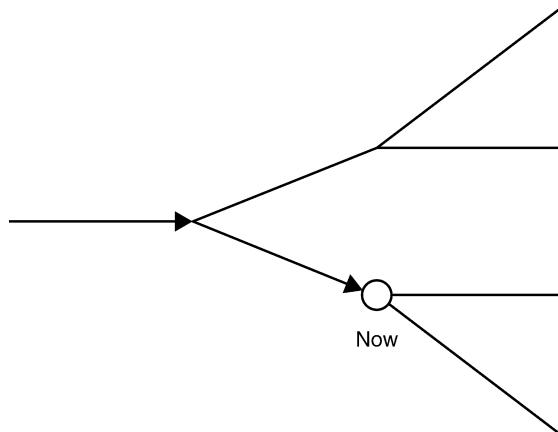
The idea of formalised branching time was first brought forward by Saul Kripke in a letter to A. N. Prior in

1958 [8, pp. 189–90]. Kripke's later development of the semantics for modal logics is well-known within computer science. But it has in fact been shown by Jack Copeland [3] that the kernel of the ideas published by Kripke were in fact present already in the work of Meredith and A. N. Prior in 1956.

The difference between A. N. Prior's theory 3 and 4 is important if time is considered to be branching. In theory 3, the notion of possibility is primitive. In theory 4, this notion can be derived from the tenses. But then it turns out to be very difficult to distinguish between the possible future, the necessary future and the "plain" future — e.g., between "possibly tomorrow," and "necessarily tomorrow" and just "tomorrow." In all obvious models constructed in accordance with A. N. Prior's theory 4, "tomorrow" is conflated either with "possibly tomorrow" or with "necessarily tomorrow." On the basis of theory 3, there is no difficulty in maintaining a difference between the three kinds of notions discussed. In a theory 3 model, one can refer not only to what happens in some possible future, $\Diamond Fq$, and to what happens in all possible futures, $\Box Fq$, but one can also refer to what is going to happen in the future, Fq, as something different from the possible as well as the necessary future. A branching time model with this property is said to be Ockhamistic, whereas a branching time model in which Fq is identified with $\Box Fq$ is said to be Peircean. Graphically, the two kinds of branching time models can be presented as in Figs. 1 and 2 respectively.

Time in Philosophical Logic. Figure 1. An Ockhamistic model of branching time. At every branching point there will be one possible future which is the true future.

Time in Philosophical Logic. Figure 2. A Peircean model of branching time. There is no difference between the status of the possible futures at any branching point.

Punctual vs. Durational Time

The notion of a "duration" is important within the study of time. Several logicians have tried to formulate a logic of durations. The medieval logician John Buridan (ca. 1295–1358) regarded the present as a duration and not as a point in time. One example which he considered was the sentence: "If a thing is moving, then it was moving." In his analysis Buridan suggested that the logic of tenses can be established in two different ways based on the durational structure of time. Either the tenses can be taken absolutely, in the sense that no part of the present time is said to be past or future. Or the tenses can be taken in the relative sense, according to which "the earlier part of the present time is called past with respect to the later, and the later part is called future with respect to the earlier." Buridan pointed out that if a thing is moving now, then there is a part of the present during which it is moving, and hence, it is moving in some part of the present, which is earlier than some other part of the present. Therefore, if the thing is moving, then it was moving (if the past is taken in the relative sense), i.e., $moving(x) \Rightarrow P(moving(x))$. For this reason, the above sentence must be accepted if the past is understood relatively, whereas it has to be rejected if the past tense is understood absolutely. The reason is that one could in principle imagine a beginning of a process of motion. (Details can be found in [8, 43 ff.].)

The first modern logician to formulate a kind of durational calculus was Walker [15]. Walker suggested

a model according to which time is considered as a structure $(S, <)$, where S is a non-empty set of periods (also called "durations" or "intervals"). The "$a < b$"-relation is to be considered as "strict" in the sense that no overlap between a and b is permitted, and the ordering is assumed to be irreflexive, asymmetrical, and transitive. In addition, he considered the notion of overlap, which can be defined as:

$$a|b \equiv_{def} \neg(a < b \vee b < a)$$

Walker formulated an axiomatic system using the following two axioms:

Definition

(W1) $a|a$

(W2) $(a < b \wedge b|c \wedge c < d) \Rightarrow a < d$

Using a set-theoretic method, Walker demonstrated that it is possible to define instants in terms of durations, thus making it possible to view a temporal instant as a "secondary" construct from the logic of durations.

In 1972 Charles Hamblin [6] independently also put forth a theory of the logic of durations. He achieved his results using a different technique involving the relation:

$$a \text{ meets } b \equiv_{def} a < b \wedge \neg(\exists c : a < c \wedge c < b)$$

A decade later, James Allen [1], in part together with Patrick Hayes [2], showed that two arbitrary durations (in linear time) can be related in exactly 13 ways. It has been shown that all these durational theories are equivalent when seen from an ontological point of view. They all show that just as durations (temporal intervals) can be set-theoretically constructed from an instant-logic, it is also possible to construct instants mathematically from durations. In fact, all the durational theories put forth so far appear to give rise to the same ontological model.

The theories formulated by Walker, Hamblin, and Allen can all be said to be B-theoretical. But Buridan's studies already suggested that it is possible to take an A-theoretical approach to durational logic. In modern durational logic an idea similar to Buridan's absolute/relative distinction was introduced in 1980 by Peter Röper [13] and others (see [8, 312 ff]).

Time and Narratives

A narrative is a text which presupposes a kind of event structure, i.e., a story. The temporal order of the story

is often called "told time." In many cases the story can be represented as a linear sequence of events. However, even if the event structure of the system is linear, the discourse structure can be rather complicated, since the reader (user) can in principle be given access to the events in any order. The order in which the events are presented is often referred to as "telling time." Keisuke Ohtsura and William F. Brewer [10] have studied some interesting aspects regarding the relation between the event structure (told time) and the discourse structure (telling time) of a narrative text.

Key Applications

The philosophy of time has typically been carried out for its own sake. In many cases philosophers and logicians have seen the study of time as intimately related to essential aspects of human existence as such. For this reason, the study of time within philosophical logic has been motivated by a fundamental interest in the concepts dealing with time themselves and not by the search for a possible application. Nevertheless, such fundamental studies of time have turned out to give rise to theories and models which are useful in many ways. For instance, A. N. Prior's analysis of the systematic relation between the dynamic and the static approach to time led him to the invention of what is now called hybrid logic (http://hylo.loria.fr). In general, temporal logic has turned out to be very useful in artificial intelligence and in other parts of computer science.

A. N. Prior's tense logic seems especially relevant for a proper description of the use of interactive systems. A description of such systems from a B-logical point of view alone cannot be satisfactory, since that would ignore the user's "nowness" which is essential in relation to the user's choices and thus to the very concept of interactivity. On the other hand, if a conceptual start is made from A. N. Prior's tense logic (i.e., the A-logical point of view), all B-logical notions can be defined in terms of the A-language.

The need for an A-logical description becomes even clearer when turning to a temporal analysis of systems which are non-linear even from a B-logical perspective, for instance a game-like multimedia system. In her studies of narratives and possible-world semantics, Marie-Laure Ryan [14] has made it clear that such a system is not to be viewed as a static representation of a specific state of affairs. Rather, it contains many different narrative lines which thread together many different states of affairs. Thus it is the choices of the user which will send the history in case on its specific trajectory.

Cross-references

► Allen's Relations
► Now in Temporal Databases
► Qualitative Temporal Reasoning
► Temporal Database
► Temporal Granularity
► Temporal Logic in Database Query Languages
► Temporal Logical Models
► Temporal Object-Oriented Databases
► Time Domain

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26:832–843, 1983.
2. Allen J.F. and Hayes J.P. A common-sense theory of time. In Proc. 9th Int. Joint Conf. on AI, 1985, pp. 528–531.
3. Copeland J. Meredith, Prior, and the history of possible world semantics. Synthese, 150(3):373–397, 2006.
4. Fraser J.T., Haber F.C., and Müller G.H. (eds.). The Study of Time, Vol. I. Springer, Berlin 1972.
5. Gale R., (ed.). The Philosophy of Time. Prometheus Books, New Jersey, 1968.
6. Hamblin C.L. Instants and intervals. In J.T. Fraser, F.C. Haber, G.H. Müller (eds.). The Study of Time, Vol. I. Springer, Berlin 1972, pp. 324–331.
7. Hasle P. and Øhstrøm P. Foundations of Temporal Logic – the WWW-site for Prior-studies. http://www.prior.aau.dk.
8. Øhrstrøm P. and Hasle P. Temporal Logic. From Ancient Ideas to Artificial Intelligence. Kluwer Academic, Dordrecht, 1995.
9. Øhrstrøm P. and Hasle P. The flow of time into logic and computer science. Bull. Eur. Assn. Theor. Comput. Sci., (82):191–226, 2004.
10. Ohtsuka K. and Brewer W.F. Discourse organization in the comprehension of temporal order in narrative texts. Discourse Processes, 15:317–336, 1992.
11. Prior A.N. Past, Present and Future. Oxford University Press, Oxford, 1967.
12. Prior A.N. Papers on Time and Tense, 2nd edn. Oxford University Press, Oxford, 2002.
13. Röper P. Intervals and tenses. J. Phil. Logic, 9:451–469, 1980.
14. Ryan M.-L. Possible Worlds, Artificial Intelligence, and Narrative Theory. Indiana University Press, 1991.
15. Walker A.G. Durées et instants. La Revue Scientifique, (3266):131 ff., 1947.
16. Whitrow G.J. Reflections on the concept of time. In The Study of Time, Vol. I. J.T. Fraser, F.C. Haber, G.H. Müller (eds.). Springer, Berlin, 1972, pp. 1–11.
17. Whitrow G.J. The Natural Philosophy of Time, 2nd edn. Oxford University Press, Oxford, 1980.

T

Time Instant

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Event; Moment; Time point

Definition

A time *instant* is a single, atomic time point in the time domain.

Key Points

Various models of time have been proposed in the philosophical and logical literature of time. These view time, among other things, as discrete, dense, or continuous.

Instants in the dense model of time are isomorphic to the rational numbers: between any two instants there is always another. Continuous models of time are isomorphic to the real numbers, i.e., they are dense and also, unlike the rational numbers, without "gaps."

A discrete time domain is isomorphic to (a possibly bounded subset of) the natural numbers, and a specific instant of such a domain then corresponds to some natural number.

The elements of a discrete time domain are often associated with some fixed duration. For example, a time domain can be used where the time elements are specific seconds. Such time elements are often called *chronons*. In this way, a discrete time domain can approximate a dense or continuous time domain.

A time domain may be constructed from another time domain by mapping its elements to granules. In this case, multiple instants belong to the same granule, and the same granule may therefore represent different instants. For example, given a time domain of seconds, a time domain of day-long granules can be constructed.

Concerning the synonyms, the term "event" is already used widely within temporal databases, but is often given a different meaning, while the term "moment" may be confused with the distinct terms "chronon" or "granule."

Cross-references

► Chronon
► Event
► Temporal Database
► Temporal Domain
► Time Domain
► Time Granularity
► Time in Philosophical Logic

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

Time Interval

CHRISTIAN S. JENSEN[1], RICHARD T. SNODGRASS[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Duration; Span; Time distance; Time period

Definition

Definition 1:

A *time interval* is a convex subset of the time domain. A time interval may be open or closed (at either end) and can be defined unambiguously by its two delimiting time instants. In a system that models the time domain using granules, an interval may be represented by a set of contiguous granules.

Definition 2:

An *interval* is a directed duration of time. A duration is an amount of time with known length, but no specific starting or ending instants. For example, the duration "1 week" is known to have a length of 7 days, but can refer to any block of seven consecutive days. An interval is either positive, denoting forward motion of time, or negative, denoting backwards motion in time.

Key Points

Unfortunately, the term "time interval" is being used in the literature with two distinct meanings: as the time between two instants, in the general database research

literature and beyond, and as a directed duration of time, in the SQL database language. The term "time period" is associated with the first definition above.

Definition 1 is recommended for non-SQL-related scientific work. Definition 2 is recommended for SQL-related work.

Concerning the synonyms, the unambiguous term "span" has been used previously in the research literature, but its use seems to be less widespread than "interval." While precise, the term "time distance" is also less commonly used. A "duration" is generally considered to be non-directional, i.e., always positive.

Cross-references
▶ Temporal Database
▶ Temporal Granularity
▶ Time Domain
▶ Time Instant
▶ Time Period
▶ Time Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
3. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.

Time Period

Nikos A. Lorentzos
Agricultural University of Athens, Athens, Greece

Synonyms
Time interval

Definition
If the *time domain* is a totally ordered set $T = \{t_1, t_2, t_3...\}$ then *a time period* over T is defined as a convex subset of elements from T.

Example: If $T = \{d_1, d_2, d_3,...\}$, where d_i are consecutive dates, then $[d_{10}, d_{20}]$ and $[d_{30}, d_{80}]$ represent two time periods over T.

Key Points
In the area of *temporal databases*, a time period over T is usually defined as a distinct data type. Some researchers define a time period data type of the form $[t_p, t_q]$. Some others define such a data type of the form $[t_p, t_q)$, i.e., its right end is closed.

Note that, initially, the term *time interval* was used instead of *time period*. This was later abandoned in order to avoid confusion, given that *interval* is a reserved word in SQL. Instead, time interval in today used with a different meaning (see *time interval*).

Cross-references
▶ Absolute Time
▶ Chronon
▶ Period-Stamped Temporal Models
▶ Time Domain
▶ Time Interval
▶ Temporal Granularity

Time Period Set

▶ Temporal Element

Time Point

▶ Time Instant

Time Quantum

▶ Chronon

Time Sequence

▶ History in Temporal Databases

Time Sequence Query

▶ Time Series Query

Time Sequence Search

Time Series

Time Series Data Mining

Time Series Database Querying

Time Series Query

Like Gao[1], X. Sean Wang[2]
[1]Teradata Corporation, San Diego, CA, USA
[2]University of Vermont, Burlington, VT, USA

Synonyms

Time sequence query; Time series search; Time sequence search

Definition

A time series query refers to one that finds, from a set of time series, the time series or subseries that satisfy a given search criteria. *Time series* are sequences of data points spaced at strictly increasing times. The search criteria are domain specific rules defined with time series statistics or models, temporal dependencies, similarity between time series or patterns, etc. In particular, similarity queries are of great importance for many real world applications like stock analysis, weather forecasting, network traffic monitoring, etc., which often involve high volumes of time series data and may use different similarity measures or pattern descriptions. In many cases, query processing consists of evaluating these queries in real-time or quasi-real time by using time series approximation techniques, indexing methods, incremental computation, and specialized searching strategies.

Historical Background

Time series queries play a key role in temporal data mining applications such as time series analysis and forecasting. It was in the recent years that these applications with massive time series data became possible due to the rapidly emerging query processing techniques, especially those for similarity queries. In 1993, Rakesh Agrawal et al. [1] proposed an indexing method for processing similarity queries in sequence databases. The key was to map time series to a lower dimensionality space by only using the first few Fourier coefficients of the time series, and building R*-trees to index the time series. This work laid out a general approach of using indexes to answer similarity queries with time series. In 1994, Christos Faloutsos et al. [6] extended this work to subsequence matching and proposed the GEMINI framework for indexing time series. In 1997, Davood Rafiei and Alberto Mendelzon [12] proposed a set of linear transformations on the Fourier series representation of a time series that can be used as the basis for similarity queries. Subsequent works have focused on new dimensionality reduction techniques [2,9,10], new similarity measures [4,5,14], and queries over streaming time series [3,7,15].

Foundations

Basic Concepts

A *time series* x is a sequence of data points spaced at strictly increasing times. The number of elements in the sequence is called the *length* of the time series. The data points are often real numbers, and therefore x can often be represented as a vector of real numbers, $x(n) = \langle x_{t_1},...,x_{t_n} \rangle$, where n is the length of x, and each t_i is the timestamp associated with the data point with $t_i < t_{i+1}$ for $i = 1,...,n-1$. The time intervals between successive data points are usually, but not always, assumed uniform, and hence $t_{i+1} - t_i$ is often assumed a constant for $i = 1,...,n-1$. When the specific t_i values are irrelevant but only signify the temporal order, x is also represented as $x(n) = \langle x_1,..., x_n \rangle$. A subsequence of x that consists of consecutive elements is called a *subseries* or *segment*

of *x*. Time series normally refer to those finite sequences of *static* elements. If the sequence has new elements continuously appended over time, they are specially called *streaming* time series [7].

Raw time series often need pre-processing to fit the application needs. It is possible to perform the following pre-processing to remove irregularities of time series. *Time regulation* is to convert a non-uniform time series to a uniform one with interpolation functions to obtain the elements at uniform time intervals. *Normalization* is to make the distance between two time series invariant to offset shifting and/or amplitude scaling. For example, given time series *x*, its mean $\bar{x}$ and standard deviation σ_x, the normalization function can be either $\tilde{x} = (x - \bar{x})$ or $\tilde{x} = (x - \bar{x})/\sigma_x$. *Linear trend reduction* is to remove the seasonal trend impact on time series, e.g., $\tilde{x}_{t_i} = x_{t_i} - (a * t_i + b)$ for all *i*. To reduce data noise, *smoothing techniques* such as moving average can also be applied.

Similarity is the degree of resemblance between two time series, and the choice of similarity measure is highly domain dependent. For applications without value scaling and time shifting concerns, a simple L_p-norm Distance, of which L_1 and L_2 are the well known Manhattan and Euclidean Distances, respectively, is sufficient. For other applications, more robust similarity measures may be needed, such as scale invariant distances (such as correlation), warping distances that allow an elastic time shifting (such as DTW or Dynamic Time Warping, Longest Common Subsequence and Spatial Assembling Distances [3]), Edit Distance With Real Penalty (ERP) [4], and model based and histogram based distances. Since most similarity measures are defined non-negative, and the smaller the values, the closer the time series, the notions of similarity and distance are often used interchangeably.

Example 1 (L_p-norm Distances, a.k.a. Minkowski Distance): Given time series $x(n)$ and $y(n)$, and positive integer *p*, let

$$L_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \, .$$

A special case is given as $L_\infty(x, y) = \max\{|x_i - y_i|, i = 1, ..., n\}$. Note when $p \rightarrow \infty$, $L_p(x, y) = L_\infty(x, y)$.

Example 2 (DTW Distance): Given time series $x(m)$ and $y(n)$, then recursively, for all $1 \le i \le m$ and $1 \le j \le n$

$$DTW(x(i), y(j)) = d(x_i, y_j) + \min\{DTW(x(i - 1), y(j - 1)), DTW(x(i - 1), y(j)), DTW(x(i), y(j - 1))\},$$

where $d()$ is a distance function defined on two elements x_i and y_j, and $x(i)$ and $y(j)$ denote the prefixes of the time series $x(m)$ and $y(n)$ of lengths *i* and *j*, respectively. The base case of the above is when $i = j = 1$ in which case $DTW(x(1), y(1)) = d(x_1, y_1)$. When both *i* and *j* are 0, $DTW(x(0), y(0))$ is defined as 0. Otherwise, when either *i* or *j* is out of range, $DTW(x(i), y(j))$ is defined as $+ \infty$. DTW is usually accompanied with one of the following global constraints, in regard to the two prefix lengths *i* and *j*.

Sakoe-Chiba Band: The allowed range of *i* and *j* in the definition above satisfies $|j - i| \le r$ for some $r \ge 0$.

Itakura Parallelogram: Use the constraint $g(i) \le j - i \le f(i)$ for *i* and *j*, where *f* and *g* are functions of *i* such that the allowed region for $j - i$ given by the constraint shows a parallelogram shape with two opposing corners at (0,0) and (*m*,*n*), respectively.

Example 3 (Edit Distance With Real Penalty [4]): Given time series $x(m)$ and $y(n)$, recursively for all $i \le m$ and $j \le n$, let

$$ERP(x(i), y(j)) = \min \begin{cases} ERP(x(i - 1), y(j - 1)) + d(x_i, y_j) \\ ERP(x(i - 1), y(j)) + d(x_i, g) \\ ERP(x(i), y(j - 1)) + d(g, y_j) \end{cases}$$

In the above, *g* is a constant value (and can be 0), $x(i) = y(j) = \langle g \rangle$ (i.e., a time series of length 1) is assumed for all $i \le 0$ and $j \le 0$, and $d(a, b) = |a - b|$. The base case of the above is when both argument time series are of length 1 in which case $ERP(x, y) = d(x_1, y_1)$. Intuitively, the constant *g* is used to fill a *gap* referring to an added element.

Time Series Query

Many forms of time series queries have been proposed over the years. Among them, one of the mostly used is *similarity search*, defined as follows: Given a set of candidate time series *X*, a similarity measure *D*, and a query series *y*, (i) find all the series in *X* whose distances to *y* are within a given threshold (near neighbor query), and (ii) find *k* series in *X* that are closest to

y (*k*-nearest neighbor query). Other queries are also possible, e.g., all pairs query that finds, in *X*, all pairs of series with distance below a given threshold. Besides similarity search, other types of queries include detecting elastic burst over streaming time series, retrieving values at any arbitrary time [13], etc. In the following, time series query refers to similarity search.

The time series query can be either *whole series matching* or *subseries matching*. The former refers to the query that concerns the whole time series, both for the query series *y* and each candidate series *x* in *X*. The latter concerns the subseries of all *x* in *X*. For example, given time series *y*(*l*), for each *x*(*n*) ∈ *X*, the latter query may need to consider $x(i + 1, i + l) = \langle x_{i+1}, ..., x_{i+l}\rangle$ for all $0 \le i \le n - l$.

In the above definition, if the query object is a set of *patterns*, the query is called *pattern matching*. A pattern is an abstract data model of time series, often seen as a short sequence, representing a class of time series that have the same properties. For pattern matching, all the involved time series may be mapped to the space in which the patterns and the similarity measure are defined.

Like the notion of time series, time series queries by default refer to those with static time series. In case of streaming time series, the queries are often monitoring the subseries within a sliding widows, and need to be evaluated periodically or continually to identify the similar series or those with the given patterns [7,15].

Query Processing: Index-based Methods for Similarity Search

Due to large volumes of data and the complexity of similarity measures, directly evaluating time series queries is both I/O and CPU intensive. There are many approaches to process these queries efficiently. Among them, one is to convert time series to other data types (e.g., strings and DNA sequences), so that the corresponding search techniques (e.g., string matching and DNA sequence matching) can be applied. Another approach is to index time series based on their approximations (or *features*), which is detailed in the following.

Time series *x* of length *n* can be viewed as a point in an *n*-dimensional space. However, spatial access methods such as *kd*-tree and *R*-tree cannot be used to index the time series directly. The problem is due to the *dimensionality curse* – the performance of spatial access methods degrades rapidly once the dimensionality is

above 16, while *n* is normally much larger than this number.

The general solution is to map time series to points in a lower *N*-dimensional space ($N << n$) and then construct the indexes in this space. The mapped points are called the time series *approximation*. Each dimension of the *N*-dimensional space represents one characteristic of the time series, such as mean, variance, slope, peak values, or a Fourier coefficient, at some coarse levels of time granularity and possibly within a shifted time window. Further, the domain of the *N*-dimensional space can be nominal so the time series approximation can be represented as a symbolic sequence [11].

Example 4 (`Piecewise Aggregate/Constant Approximation` [14,8]): Time series *x* of length *mN* can be mapped to a point in the *N*-dimensional space: $\bar{x} = (\bar{x}_1, ..., \bar{x}_N)$ where the value in the *ith* dimension is the mean over the *ith* segment of *x*, $\bar{x}_i = \frac{1}{m}\sum_{j=m(i-1)+1}^{mi} x_j$.

Example 5 (`Line Fitting Approximation` [11]): Given an alphabet $A\{$"*up*", "*down*", "*flat*"$\}$, define a mapping function $s(z) \in \mathcal{A}$ where *z* is a time-series of length *m*. Time series *x* of length *mN* can be mapped to a length-*N* symbolic sequence, $\langle s(x_1, ..., x_m), s(x_{m+1}, ..., x_{2m}), ..., s(x_{(N-1)m+1}, ..., x_{mN})\rangle$, e.g., $\langle$"*up*", "*up*", ..., "*down*"$\rangle$. Function *s* can be line fitting and, based on the slope of the fitting line, decides if the value is "*up*" or "*down*" etc.

A multi-step algorithm can be used to process a query. Take the *k*-nearest neighbor search as an example. First step: find *k*-nearest neighbors in the lower-dimensional index structure. Find the actual distance between the query series and the *kth* nearest neighbor. Second step: use this actual distance as a range query to find (in the lower-dimensional index) all the database points. Third step: calculate the actual distances found in step 2 and obtain the actual *k*-nearest neighbors. An improvement to this algorithm is to incrementally obtain nearest neighbors in the lower-dimensional approximation, and each time an actual distance is obtained, it is used to remove some database points that are returned by the range query (second step above).

The index-based methods need to guarantee soundness (no false alarms) and completeness (no false dismissals) in the query result. Soundness can be guaranteed by checking the original data as in step 3. The completeness can be guaranteed only if the chosen

approximation method has the *lower-bounding property* [6]. That is, given a similarity measure D, for any candidate time series x and query time series y, let $\bar{x}$ and $\bar{y}$ be their lower-dimensional approximations and D' be the distance defined on $\bar{x}$ and $\bar{y}$, then $D'(\bar{y}, \bar{x}) \leq D(y, x)$ must hold for any x and y.

Example 6 (Lower Bounding Approximation for Euclidian Distance): Method (1): apply an orthonormal transform (Fourier transform, Wavelet transform, and SVD) to both query and candidate time series and ignore many "insignificant" axes after the transform. The distance defined on the remaining axes gives the lower bounding approximation for Euclidian Distance [1]. Method (2): apply segmented mean approximation to both query and candidate time series. It is easy to see $\sqrt[p]{m} L_p(\bar{x}, \bar{y}) \leq L_p(x, y)$ for all p, while m is the factor of dimensionality reduction, i.e., x's length divided by $\bar{x}$'s. Since this lower-bounding approximation works for all p, one index tree can be used for all L_p-norm distances [8,14].

Example 7 (Lower Bounding Approximation for DTW Distance [9]): To derive a lower bounding approximation for DTW, approximate the candidate time series x using segmented mean approximation, and approximate the query time series y as follows. Let $y = \langle y_1,...,y_{mN} \rangle$. Define $U = \langle U_1,...,U_{mN} \rangle$ and $L = \langle L_1,...,L_{mN} \rangle$ where $U_i = \max(y_{i-r},...,y_{i+r})$ and $L_i = \min(y_{i-r},...,y_{i+r})$ (r is the allowed range for $m - n$ in Sakoe-Chiba Band or Itakura Parallelogram). Sequences U and L form a bounding envelope that encloses y from above and below. Then reduce U and L to a lower dimension N, define $\hat{U} = \langle \hat{U}_1,...,\hat{U}_N \rangle$ and $\hat{L} = \langle \hat{L}_1,...,\hat{L}_N \rangle$, where $\hat{U}_i = \max(U_{(i-1)m+1},...,U_{im})$ and $\hat{L}_i = \min(L_{(i-1)m},...,L_{im})$, that is, $\hat{U}$ and $\hat{L}$ are piecewise constant functions that bound U and L, respectively. Let

$$LB\_PAA(y, \bar{x}) = \sqrt{m \sum_{i=1}^{N} \begin{cases} (\bar{x}_i - \hat{U}_i)^2 & \text{if } \bar{x}_i > \hat{U}_i; \\ (\bar{x}_i - \hat{L}_i)^2 & \text{if } \bar{x}_i < \hat{L}_i; \\ 0 & \text{otherwise.} \end{cases}},$$

then $LB\_PAA(y, \bar{x}) \leq DTW(y, x)$.

Query Processing: Similarity Search over Streaming Time Series

These queries are different from those with static time series, in that (i) having a sliding window or windows of multiple lengths at the same time; (ii) continuous monitoring; and (iii) incremental evaluation. In the following, consider the two problems: Euclidean distance or correlation monitoring among pairs of streams, and Euclidean distance or correlation monitoring between a stream time series and a time series database.

The first query problem is, given many streaming time series, how to find pairs of time series that have strong (positive or negative) correlations in the last sliding window [15].

The idea is to use the notion of "basic windows," similar to segmented mean application. Instead of mean, the coefficients of the Fourier transform of each segment is used to approximate the time series. Given $x = \langle x_1,...,x_b \rangle$ and $y = \langle y_1,...,y_b \rangle$, where b is the size of the basic window. If $x_i = \sum_{m=0}^{N-1} C_m^x f_m(i)$ and $y_i = \sum_{m=0}^{N-1} C_m^y f_m(i)$ is a family of orthogonal functions, then the inner product of x and y,

$$\begin{aligned} x * y &= \sum_{i=1}^{b} x_i y_i \\ &= \sum_{i=1}^{b} \left(\sum_{m=0}^{N-1} C_m^x f_m(i) \sum_{p=0}^{N-1} C_p^y f_p(i) \right) \\ &= \sum_{m=0}^{N-1} \sum_{p=0}^{N-1} C_m^x C_p^y \left(\sum_{i=1}^{b} f_m(i) f_p(i) \right). \end{aligned}$$

Note $\sum_{i=1}^{b} f_m(i) f_p(i)$ does not depend on x and y and can be pre-computed. From this, the inner product of two time series can be computed for each sliding window aligned with the basic windows (i.e., a sliding window must be the union of some basic windows). Fourier bases can be used as the f functions, and discrete Fourier transform (DFT) can compute the coefficients efficiently in an incremental way.

By only taking a few Fourier coefficients (small N), the approximate inner products and hence the Euclidean distance can be evaluated efficiently. To compute correlations, the normalized series of $\hat{x}_i = (x_i - \bar{x})/\sigma_x$ need only be considered, where $\bar{x}$ and σ_x are the mean and standard deviation of x over the sliding window.

A step further: since two series are highly correlated if their DFT coefficients are similar, an indexing structure can help to store the coefficients and look for series with high correlation only in series with similar coefficients.

The second query problem is, given a database of pattern time series, a streaming time series, and window size N, how to find the nearest neighbor of the

T

streaming time series (using the last N values) in the database, at each time position [7].

The idea is a batch processing that uses fast Fourier transform (FFT) and its inverse to calculate the cross correlation of streaming time series and patterns at many time positions. Given $x = \langle x_1,...,x_N \rangle$ and $y = \langle y_1,...,y_N \rangle$, the circular cross correlation sequence is defined as

$$CirCCorr_d^{x,y} = \sum_{i=1}^{N} x_{(d+i-1) \bmod N} y_i, \quad d = 1, 2,..., N,$$

where d is the time lag. Let $\dot{x}$ and $\dot{y}$ be the DFT transforms of x and y respectively, then sequence $\langle \dot{x}_1 \dot{y}^*_1,..., \dot{x}_N \dot{y}^*_N \rangle$ is the result of DFT transform of $CirCCorr^{x,y}$. Here $\dot{y}^*_i$ is the conjugate of $\dot{y}_i$.

With the CirCCorr, calculation of the Euclidean distances of a number of time positions can be done in a batch mode. This is faster than calculating the individual distances, as the batch process has time complexity $O(N \lg N)$, as compared to the direct computing of $O(Nl)$, where l ($l < N$) is the number of time positions covered by one batch processing. So it is profitable to wait for a few time steps and then find the nearest neighbors for these time steps all together. The longer the wait is, the more computation time saved. However, this causes a lengthening of the response time, i.e., a loss of the chance of finding the answer as early as possible. To overcome this, one may use a certain model to roughly predict the future values of the time series and apply the batch processing to compute all the Euclidean distance (or correlations) of many future time positions. When the actual values come, triangular inequality can filter out a lot of time series in the database that are not the nearest neighbor [7].

Key Applications

Market data analysis and trend predication, network traffic control, intrusion detection, temporal data mining.

Data Sets

1. UCR Time Series Classification/Clustering Page: http://www.cs.ucr.edu/~eamonn/time_series_data/
2. PhysioBank: Physiologic Signal Archives for Bio-medical Research (including ECG and synthetic time series with known characteristics): http://www.physionet.org/physiobank/

URL to Code

1. Above URL 1.
2. ANN: A Library for Approximate Nearest Neighbor Searching: http://www.cs.umd.edu/~mount/ANN/

Cross-references

▶ Curse of Dimensionality
▶ Dimensionality Reduction
▶ Discrete Wavelet Transform and Wavelet Synopses
▶ High Dimensional Indexing
▶ Indexing and Similarity Search
▶ Nearest Neighbor Query
▶ Range Query
▶ R-tree (and Family)
▶ Sequential patterns
▶ Singular Value Decomposition
▶ Stream Similarity Mining
▶ Temporal data mining
▶ Top-K Selection Queries on Multimedia Datasets

Recommended Reading

1. Agrawal R., Faloutsos C., and Swami A.N. Efficient similarity search in sequence databases. In Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms, 1993, pp. 69–84.
2. Chan K.P. and Fu A.W.-C. Efficient time series matching by wavelets. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 126–133.
3. Chen Y., Nascimento M.A., Ooi B.C., and Tung A.K.H. Spade: on shape-based pattern detection in streaming time series. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 786–795.
4. Chen L. and Ng R. On the marriage of lp-norms and edit distance. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 792–803.
5. Das G., Gunopulos D., and Mannila H. Finding similar time series. In Principles of Data Mining and Knowledge Discovery, 1st European Symp., 1997, pp. 88–100.
6. Faloutsos C., Ranganathan M., and Manolopoulos Y. Fast subsequence matching in time-series databases, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 419–429.
7. Gao L. and Wang X.S. Continuous similarity-based queries on streaming time series. IEEE Trans. Knowl. Data Eng., 17(10):1320–1332, 2005.
8. Keogh E.J. and Pazzani M.J. Scaling up dynamic time warping for datamining applications. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 285–289.
9. Keogh E.J. and (Ann) Ratanamahatana C. Exact indexing of dynamic time warping. Knowl. Inform. Syst., 7(3):358–386, 2005.
10. Korn F., Jagadish H.V., and Faloutsos C. Efficiently supporting ad hoc queries in large datasets of time sequences. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 289–300.

11. Qu Y., Wang C., Gao L., and Wang X.S. Supporting movement pattern queries in user-specified scales. IEEE Trans. Knowl. Data Eng., 15(1):26–42, 2003.

12. Rafiei D. and Mendelzon A. Similarity-based queries for time series data. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1997, pp. 13–25.

13. Revesz P., Chen R., and Ouyang M. Approximate query evaluation using linear constraint databases. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 170–175.

14. Yi B.-K. and Faloutsos C. Fast time sequence indexing for arbitrary LP norms. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 385–394.

15. Zhu Y. and Shasha D. Statstream: statistical monitoring of thousands of data streams in real time. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 358–369.

Time Series Search

► Time Series Query

Time Span

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tuscon, AZ, USA

Synonyms
Time interval; Time distance

Definition
A span is a directed duration of time. A duration is an amount of time with known length, but no specific starting or ending instants. For example, the duration "1 week" is known to have a length of 7 days, but can refer to any block of seven consecutive days. A span is either positive, denoting forward motion of time, or negative, denoting backwards motion in time.

Key Points
Concerning the synonyms, the terms "time interval" is generally understood to denote an anchored span in the general community of computer science. Only in the SQL language does "time interval" denote a span. The term "span," which has only one definition, is thus recommended over "time interval" for works not related to the SQL language. This use is unambiguous.

A "duration" is generally considered to be non-directional, i.e., always positive. The term "time distance" is precise, but is longer.

Cross-references
► Fixed Span
► Temporal Database
► Time Instant
► Time Interval
► Variable Span

Recommended Reading
1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.

2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer Verlag, Berlin, 1998, pp. 367–405.

Time Unit

► Chronon

Time-based Access Control

► Temporal Access Control

Time-based Window

► Windows

Time-Constrained Transaction Management

► Real-Time Transaction Processing

Time-Dependent Graphs

► Time Aggregated Graphs

Time-Dependent Networks

► Time Aggregated Graphs

Time-Line Clock

Curtis Dyreson
Utah State University, Logan, UT, USA

Synonyms

Clock; Base-line clock; Time-segment clock

Definition

In the discrete model of time, a *time-line clock* is defined as a set of *physical clocks* coupled with some specification of when each physical clock is authoritative. Each *chronon* in a time-line clock is a chronon (or a regular division of a chronon) in an identified, underlying physical clock. The time-line clock switches from one physical clock to the next at a *synchronization point*. A synchronization point correlates two, distinct physical clock measurements.

Key Points

A time-line clock is the clock for (concrete) times stored in a temporal database. A time-line clock glues together a sequence of physical clocks to provide a consistent, clear semantics for a time-line. Since the range of most physical clocks is limited, a time-line clock is usually composed of many physical clocks. For instance, a tree-ring clock can only be used to date past events, and the atomic clock can only be used to date events since the 1950s. Though several physical clocks might be needed to build a time-line, in some cases a single physical clock suffices. For instance SQL2 uses the mean solar day clock – the basis of the *Gregorian calendar* – as its time-line clock.

Cross-references

▶ Chronon
▶ Physical Clock
▶ Time Instant

Recommended Reading

1. Dyreson C.E. and Snodgrass R.T. Timestamp semantics and representation. Inf. Syst., 18(3):143–166, 1993.
2. Dyreson C.E. and Snodgrass R.T. The baseline clock. The TSQL2 Temporal Query Language. Kluwer, Norwell, MA, 1995, pp. 73–92.
3. Fraser J.T. Time: The Familiar Stranger. University of Massachusetts Press, Amherst, MA, 1987, p. 408.

Time-Oriented Database

▶ Temporal Database

Time-Segment Clock

▶ Time-Line Clock

Timeslice Operator

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Rollback operator; State query

Definition

The *valid-timeslice operator* may be applied to any temporal relation that captures valid time. Given also a valid-time element as a parameter, it returns the argument relation reduced in the valid-time dimension to just those time(s) specified by the valid-time element. The *transaction timeslice operator* is defined similarly, with the exception that the argument relation must capture transaction time.

Key Points

Several types of timeslice operators are possible. Some may restrict the time parameter to intervals or instants. Some operators may, given an instant parameter, return a conventional relation or a transaction-time relation when applied to a valid-time or a bitemporal relation, respectively; other operators may always return a result relation of the same type as the argument relation.

Oracle supports timeslicing through its flashback queries. Such queries can retrieve all the versions of a row between two transaction times (a key-transaction-time-range query) and allows tables and databases to be rolled back to a previous transaction time, discarding all changes after that time.

Concerning the synonyms, "rollback operator" is an early term that has since been abandoned. This term indicates that the result of a timeslice is a

relation obtained by moving backwards in time, presumably from the current transaction time. This kind of result is less general than those that may be obtained using a timeslice operator. Specifically, this kind of result assumes a time parameter that extends from the beginning of the time domain to some past time (with respect to the current time). Similarly, "state query" suggests a less general functionality than what is actually offered by timeslice operators.

Cross-references
► Bitemporal Relation
► Temporal Database
► Temporal Element
► Temporal Query Languages
► Time Instant
► Time Interval
► Transaction Time
► TSQL2
► Valid Time

Recommended Reading
1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

TIN
► Triangulated Irregular Networks

Tiny Aggregation (TAG)
► In-Network Query Processing

TinyDB
► In-Network Query Processing

TinySQL
► Database Languages for Sensor Networks

t-Norm
► Triangular Norms

Topic Detection and Tracking

NING LIU
Microsoft Research Asia, Haidian, China

Definition
According to the definition at http://projects.ldc.upenn.edu/TDT/, Topic Detection and Tracking (TDT) is a multi-site research project to develop core technologies for a news understanding systems. Specifically, TDT systems discover the topical structure in unsegmented streams of news reporting as it appears across multiple media and in different languages. Some terms are defined below before the TDT problem is fully understood (The definitions are borrowed from Omid Dadgar's work).

1. Event – An event is something that happens at some specific time and place, and the unavoidable consequences. Specific elections, accidents, crimes and natural disasters are examples of events.
2. Activity – An activity is a connected set of actions that have a common focus or purpose. Specific campaigns, investigations, and disaster relief efforts are examples of activities.
3. Story – A story is a newswire article or a segment of a news broadcast with a coherent news focus. They must contain at least two independent, declarative clauses.
4. Topic – The topic is defined as a seminal event or activity, along with all directly related events and activities.

With the definition of topic, the Topic Detection and Tracking can be known as to investigate the state of the art in finding and following new events in a stream of broadcast news stories. According to the Pilot-study, the original TDT problem consists of three major tasks: (i) segmenting a stream of data, especially recognized speech, into distinct stories; (ii) identifying those news stories that are the first to discuss a new event occurring in the news; and (iii) given a small

number of sample news stories about an event, finding all following stories in the data stream.

Historical Background

The TDT study starts from 1996 through a Pilot-study [2] which aims to explore the approaches and performance baselines. After that, it quickly attracted much attention. Followed by the TDT2, TDT3 etc, increasing number of research works are focusing on the TDT problem. TDT2 in 1998 was the first major step in TDT after the pilot study since it established the foundation for the following works. It addresses the same three problems, which are segmentation, detection, and tracking with the original Pilot study. The evaluation procedures were modified and the volume and variety of data and the number of target topics were expanded. TDT2 attacked the problems introduced by imperfect, machine-generated transcripts of audio data. The TDT3 was used for the year 1999, 2000 and 2001 test. The TDT4 and TDT5 are used for the year 2002, 2003 and 2004 test respectively.

From the algorithms' perspective, start from the Pilot-study, many algorithms and applications about TDT have been proposed [1]. As some examples, in the Pilot-study, the Dragon approach, UMass approach and CMU approach were introduced for text segmentation. The same three approaches are used for new event detection. They are finally utilized for event tracking. As some recent progresses, Masaki et al. proposed the Topic Detection and Tracking for News Web Pages by cluster and SuffixTree [4]. He et al. proposed to conduct the topic detection and Tracking by topic sensitive language model [5]. Makkonen et al. proposed to utilizing the temporal information for topic detection and tracking. In the next section, the TDT problems is considered as several sub-problems. Some algorithms are summarized to address the sub-problems of TDT.

Foundations

According to the Pilot study [1], the TDT problem has the following several sub-problems (the major contents of this section is borrowed from the Pilot study).

The Segmentation Task

The segmentation task is defined to be the task of segmenting a continuous stream of text (including transcribed speech) into its constituent stories. To support this task the story texts from the study corpus will be concatenated and used as input to a segmenter. This concatenated text stream will include only the actual story texts and will exclude external and internal tag information. The segmentation task is to correctly locate the boundaries between adjacent stories, for all stories in the corpus.

The Detection Task

The detection task is characterized by the lack of knowledge of the event to be detected. In such a case, one may wish to retrospectively process a corpus of stories to identify the events discussed therein, or one may wish to identify new events as they occur, based on an on-line stream of stories. Both of these alternatives are supported under the detection task.

Retrospective Event Detection

The retrospective detection task is defined to be the task of identifying all of the events in a corpus of stories. Events are defined by their association with stories, and therefore the task is to group the stories in the study corpus into clusters, where each cluster represents an event and where the stories in the cluster discuss the event. It will be assumed that each story discusses at most one event. Therefore each story may be included in at most one cluster.

Online New Event Detection

The on-line new event detection task is defined to be the task of identifying new events in a stream of stories. Each story is processed in sequence, and a decision is made whether or not a new event is discussed in the story, after processing the story but before processing any subsequent stories). A decision is made after each story is processed. The first story to discuss an event should be flagged YES. If the story doesn't discuss any new events, then it should be flagged NO.

The Tracking Task

The tracking task is defined to be the task of associating incoming stories with events known to the system. An event is defined ("known") by its association with stories that discuss the event. Thus each target event is defined by a list of stories that discuss it.

To solve these sub-problems, various algorithms have been proposed. For the segmentation, as

addressed by the Pilot study, there is a relatively small but varied body of previous work that has addressed the problem of text segmentation. This work includes methods based on semantic word networks, vector space techniques from information retrieval and decision tree induction algorithms. As for some classical algorithms, the Dragon's approach to segmentation is to treat a story as an instance of some underlying topic, and to model an unbroken text stream as an unlabeled sequence of these topics. In this model, finding story boundaries is equivalent to finding topic transitions. Given a text stream, a probability can be attached to any particular hypothesis about the sequence and segmentation of topics in the following way:

1. Transition from the start state to the first topic, accumulating a transition probability.
2. Stay in topic for a certain number of words or sentences, and, given the current topic, accumulate a selfloop probability and a language model probability for each.
3. Transition to a new topic, accumulating the transition probability. Go back to step 2.

A search for the best hypothesis and corresponding segmentation can be done using standard HMM techniques and standard speech recognition tricks.

After the segmentation work, Event *detection* is the problem of identifying stories in several continuous news streams that pertain to new or previously unidentified events. Using the same Dragon approach as example, Dragon's online and retrospective detection systems are applications of the clustering technology used to train background models for the segmenter. This technology is an implementation of a k-means clustering algorithm. The next step after event detection is the event tracking.

The TDT event tracking task is fundamentally similar to the standard routing and filtering tasks of Information Retrieval (IR). Given a few sample instances of stories describing an event (i.e., stories that provide a description of the event), the task is to identify any and all subsequent stories describing the same event. Event tracking is different from those IR tasks in that events rather than queries are tracked, and in that events have a temporal locality that more general queries lack. These differences shift the nature of the problem slightly but at the same time shift the possible solutions significantly. The narrowing of the scope of information filtering encourages modifications to existing approaches and invites entirely new approaches that were not feasible in a more general query centric setting.

Dragon's event tracker is an adaptation of its segmenter. As discussed there, the segmentation algorithm does segmentation and topic assignment simultaneously. In general, the topic labels assigned by the segmenter are not useful for classification, as they are few in number and do not necessarily correspond to categories a person would find interesting. However, by supplementing the background topic models with a language model for a specific event of interest, and allowing the segmenter to score segments against this model, it becomes possible for the segmenter to output a notification of an occurrence of that event in the news stream whenever it assigns that event model's label to a story. In this implementation, the topic models have the role of determining the background against which the event model must score sufficiently well to be identified.

Key Applications

TDT techniques have wide range of applications especially on the Web documents. As for the key applications, the major goal of TDT is to finding and following new events in a stream of broadcast news stories.

Future Directions

The future directions of TDT are in several fold. The first is to propose more effective algorithms for some classical problems such as monitoring streams of news in multiple languages (e.g., Mandarin) and media – newswire, radio, television, web sites or some future combination. On the other hand, due to the rapid growth of World Wide Web, the scale of the data for topic detection and tracking is getting larger and larger, thus more scalable algorithms are highly desired. Another direction for exploring is to find new online applications of the TDT problem.

Data Sets

The most commonly used corpus for TDT study start from the Pilot-study. And then the TDT2, TDT3, TDT2000, TDT2001, TDT4 and TDT5 were released. The details about the datasets, tasks and evaluation metrics can be found at http://projects.ldc.upenn.edu/TDT/. LDC is the provider of the corpus for the second phase of TDT and is currently developing the

phase three corpus. As an example, some details about TDT2 are introduced.

Recommended Reading

1. Allan J. Topic Detection and Tracking. Kluwer, Norvell, MA, 2002.
2. Allan J., Carbonell J., Doddington G., Yamron J., and Yang Y. Topic detection and tracking pilot study final report. In Proc. DARPA Broadcast News Transcription and Understanding Workshop, 1998, pp. 194–218.
3. Makkonen J. and Ahonen-Myka H. Utilizing Temporal Expressions in Topic Detection and Tracking. In Proc. 7th European Conf. Research and Advanced Technology for Digital Libraries, 2003, pp. 393–404.
4. Mori M., Miura T., and Shioya I. Topic detection and tracking for news web pages. In Proc. 2006 IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2006, pp. 338–342.
5. Ruifang H., Bing Q., Ting L., and Sheng L. The topic detection and tracking with topic sensitive language model. In Proc. Int. Conf. on Mutilingual Information Processing, 2005, pp. 324–327.

Topic Hierarchies

▶ Lightweight Ontologies

Topic Maps

JAMES CAVERLEE
Texas A&M University, College Station, TX, USA

Definition

Topic Maps provide a standardized way to represent and interchange knowledge through the modeling of abstract concepts (called *topics*), the relationships among topics (called *associations*), and the connection between abstract concepts and real-world resources (called *occurrences*). By distinguishing the high-level topic space from real-world resources, Topic Maps may be used both as a semantic map among related concepts (in the topic space) and as a way to describe real-world resources (through occurrence mapping from the topic space into the resource space). Topic Maps have been formally standardized by the international standards body ISO.

Historical Background

The pre-cursors of what are now known as Topic Maps began in the early 1990s with an effort to merge independently created and maintained indexes for information sharing. This early work motivated the need for a more general and more useful knowledge description meta framework. By 1999, the original ISO standard for Topic Maps was published as ISO/IEC 13250 based primarily on SGML and the hypermedia linking language HyTime.

Foundations

Topic Maps support the modeling and exchange of knowledge based on a standardized framework centered around topics, occurrences, and associations [1,3,4]. These constructions serve as n-ary connections between items that express overarching semantics.

Using Topic Maps

According to the ISO standard, Topic Maps [2] are designed to facilitate knowledge representation and interchange by:

- Providing an abstract layer of topic-centered metadata over information resources for supporting navigational tools like indexes, glossaries, and citation systems.
- Linking topics in a clearly defined fashion so that users can navigate between them. Such linking can support thesaurus-like interfaces to disparate information stores.
- Supporting different "views" over a set of information resources by filtering information resources based on the metadata described in Topic Maps.
- Adding a structured layer over unstructured information resources in the form of a markup that is completely external to the original resources.

The Basics of Topic Maps

To represent and interchange knowledge in a standardized way, Topic Maps rely on three key concepts: (i) topics, which represent abstract concepts; (ii) associations, which model relationships among topics; and (iii) occurrences, which connect topics to real-world resources.

The most basic element in a topic map is a *topic*. In general, a *topic* is an abstract concept or subject of concern within the framework of Topic Maps. A topic

can be used to represent people, places, events, organizations, Web pages, documents, or any other reasonable unit of interest. In an example universe of discourse, it may be appropriate to represent the author Jane Austen as a topic, as well as the concepts of author, person, novel, and so on. A topic may be associated with one or more names; in the running example, Jane Austen the topic may be referred to by multiple names, including Austen, Jane Austen, and jane-austen.

Using XTM, the topic for novel and the topic for Jane Austen (as an instance of a Writer) can be expressed as:

```
<topic id="novel">
   <baseName>
      <baseNameString>Novel
      </baseNameString>
   </baseName>
</topic>

<topic id="jane-austen">
   <instanceOf><topicRef xlink:href=
   "#writer"/></instanceOf>
   <subjectIdentity>
      <subjectIndicatorRef xlink:href=
      "{http://en.wikipedia.org/wiki/
      Jane_Austen}"/>
   <subjectIdentity>
   <baseName>
      <baseNameString>Austen, Jane 1775-
      1817</baseNameString>
   </baseName>
</topic>
```

Note that the Jane Austen topic includes a special *subject indicator* syntax that refers to the Wikipedia entry for Jane Austen. A subject indicator is a guideline for a human consumer of the topic map that, although Jane Austen the writer cannot be directly addressed by a URL, the Wikipedia article uniquely identifies her. In this way, the *subject indicator* serves as a *subject identifier* for the topic Jane Austen. By construction, two topics that have the same subject identifier must refer to the same abstract concept.

The second key component of Topic Maps is an *association*. An *association* is a relationship between topics in a topic map. For example, a topic for the author Jane Austen and a topic for the novel *Pride and Prejudice* (which was written by Jane Austen) could be

linked by the association "written-by" which links an instance of an Author with an instance of a Novel. These instances serve as roles in the association. Since associations imply no directionality, the "written-by" association implicitly has a dual association "wrote." There are no limits on the number and nature of associations in a Topic Map, so the Topic Maps paradigm may be used to model complex and sophisticated domains, as well as simpler domains as in the Jane Austen example.

```
<association>
   <instanceOf><topicRef xlink:href="#
   written-by"/></instanceOf>
   <member>
      <roleSpec><topicRef xlink:
      href="#author"/> </roleSpec>
      <topicRef xlink:href="#jane_
      austen"/>
   </member>
   <member>
      <roleSpec><topicRef xlink:
      href="#novel"/> </roleSpec>
      <topicRef xlink:href="#pride_and_
   prejudice"/>
   </member>
</association>
```

Finally, an *occurrence* is a real representations of a topic. For example, a Web-accessible file of the book *Pride and Prejudice* is an occurrence of the topic of the same name. Similarly, the *Pride and Prejudice* topic may also occur in a scholarly article discussing the role of women in British literature that happens to mention *Pride and Prejudice*. In an XTM topic map, an occurrence must be a resource that is addressable using a Uniform Resource Identifier (URI) or may be placed inline as character data. Hence, an occurrence of Jane Austen could be an external resource like a Web page or an image, or a brief in-line description of the author.

In the following example, two of Jane Austen's works are referenced as occurrences:

```
<topic id="pride_and_prejudice">
   <instanceOf><topicRef xlink:href="#
   novel"/> </instanceOf>
      <baseName><baseNameString>Pride
      and Prejudice</baseNameString>
      </baseName>
```

```
<occurrence>
    <instanceOf><topicRef  xlink:href=
    "#pdf-format"/></instanceOf>
    <resourceRef xlink:href="http://
    www.gutenberg.org/dirs/etext98/
    pandp12p2.pdf"/>
</occurrence>
</topic>

<topic id="sense_and_sensibility">
    <instanceOf><topicRef xlink:href="
    #novel"/> </instanceOf>
    <baseName><baseNameString>Sense and
    Sensibility</baseNameString>
    </baseName>
    <occurrence>
        <instanceOf><topicRef xlink:href=
        "#pdf-format"/></instanceOf>
        <resourceRef xlink:href="http://
        www.gutenberg.org/dirs/etext94/
        sense11p.pdf"/>
    </occurrence>
</topic>
```

Extending the Basic Model

Topic Maps can be further refined through the use of *types* and *scope*.

A *type* is fundamentally a special kind of association between topics, used to indicate that one topic is an "instance-of" another topic. For example, since *Pride and Prejudice* is a book, there may also exist in the topic map an "instance-of" association between the topic *Pride and Prejudice* and a topic representing the concept of a book. A topic may have multiple types.

A *scope* provides additional contextual information about the elements of a Topic Map. Scope allows for the same topic map to exist at different levels of specification. Users of the topic maps can then decide if they are interested in things from any scope or only from one particular scope or subset of scopes. For example, scope can be used to provide localized names for topics – one name for English, one for Spanish, and one for French.

Merging Topic Maps

Since Topic Maps may be developed in a distributed or independent environment, one of the key features of Topic Maps is the notion of *merging*. Merging means that two topic maps with identical topics can have their associations and occurrences combined together to build a richer semantic model. In particular, two topics can be combined (or "merged") into a single topic containing the union of the types, the names, and the occurrences of the original two topics. This merged topic replaces the original two topics wherever they participate as a role in an association or serve as a topic type.

Using Topic Maps

The Topic Maps standard provides a reference point for the appropriate syntax and functionality of topic maps, leaving the implementation details of a topic maps processing engine to commercial and non-commercial applications and tools. Currently, there are topic map processing libraries in most popular languages like Java, C, Perl, and Python. In an effort to provide a uniform programmatic interface to topic maps, regardless of the particular language and platform, the Common Topic Map Application Programming Interface (TMAPI) has been recently developed. The TMAPI specification provides a base set of core interfaces for accessing and manipulating topic maps.

Key Applications

Web, Semantic Web, digital libraries, business-to-business exchange.

Cross-references

► Conceptual Schema Design
► RDF
► Semantic Web

Recommended Reading

1. Garshol L. Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all. J. Inf. Sci., 30(4):378–391, 2004.
2. International Organization for Standardization. ISO 13250-2003 Information technology – SBML applications – Topic maps. Available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc /catalogue_detail.htm?csnumber=38068
3. Park J. and Hunting S. (eds.). XML Topic Maps. Addison-Wesley, Boston, MA, USA, 2002.
4. Pepper S. The TAO of topic maps: finding the way in an age of infoglut. In Proc. XML Europe Conf., 2000.

Topical-Hierarchical Relevance

► Relevance

Topic-based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Synonyms

Subject-based publish/subscribe

Definition

Topic-based publish/subscribe is a communication abstraction that supports selective message dissemination among many sources and many sinks. Messages are associated with topics and are selectively routed to destinations with matching topic interests. Data sinks specify interest in receiving messages of a given topic and data sources publish messages on different topics. Topic-based publish/subscribe is an instance of the more general publish/subscribe concept.

Key Points

Topic-based publish/subscribe is an instance of the more general publish/subscribe concept. In the topic-based publish/subscribe model, a data source submits publication messages associated with a topic to the publish/subscribe system, while a data sink subscribes its interest in receiving messages of certain topics by submitting subscription expressions on available topics to the system. The kind of topics to publish or subscribe that exist is either out of band information and must be know to clients, or is dynamically discoverable by clients based on additional support provided by the system. For example, by subscribing to control channel topics, where the creation of new topics is announced. Topics are an integral part of the messages disseminated through the publish/subscribe system. The publish/subscribe system only knows how to interpret the topics, but not the rest of the publication message, which remains opaque to the system.

A publication message published to the topic-based publish/subscribe system is delivered to all subscribers with matching subscriptions. A subscription matches a publication if the topic associated with the publication message matches the subscription expression. In very simple realizations of this model, a topic is simply a string that represents a name, a subject, or a topic according to which messages are classified. In more sophisticated realizations, topics draw from a hierarchical topic space. The topic space is used to categorize messages. For example, in a market data dissemination scenario, messages may be classified according to the stock exchange, the traded commodity, and the kind of information disseminated. A message could for instance be associated with the following topic: `/NAS-DAQ/ABC-Inc/AskingPrice`. A subscriber can express interest in receiving messages of a specific topic, such as by subscribing to `/NASDAQ/ABC-Inc/AskingPrice`, or by defining a set of messages it is interested in, such as `/NASDAQ/ABC-Inc/*`, which indicates that the subscriber would like to receive any message published with topic `NASDAQ` and `ABC-Inc`.

As in the other publish/subscribe models, the topic-based publish/subscribe model decouples the interaction among publishing data sources and subscribing data sinks. The same decoupling characteristics as discussed under the general publish/subscribe concept apply here as well. Specific realizations of this model found in practice vary in the exact decoupling offered. To properly qualify as publish/subscribe, at least the anonymous communication style must exist. That is publishing clients must not be aware of who the subscribing clients are and how many subscribing clients exist, and vice versa. Thus, topic-based publish/subscribe enables the decoupled interaction of n sources with m sinks for $n, m \geq 1$.

In topic-based publish/subscribe, the publication data model is defined by the topics that can be associated with messages. Simplistic models allow the application developer to categorize messages by defining a flat topic space, simply a collection of topics. More sophisticated approaches allow the application developer to select topics from a hierarchical topic space. Whether flat or hierarchical, topics are often strings, possibly structured with separators for the hierarchical case. Some approaches additionally type the various levels of a hierarchical topic space allowing the application developer to use various operators supported by the type for expressing subscriptions. For example, in he above example, `AskingPrice` could be defined as `Integer`, to allow the subscriber to express a relational condition on the messages returned. This approach is close in expressiveness to the capabilities of content-based or type-based publish/subscribe, as the matching mechanism now also inspects the message content, i.e., the value associated with `AskingPrice`.

The subscription language model depends on the publication data model. A subscription expression defines the subscriber's interest in receiving messages.

Given a flat publication data model, subscribers can express interest in receiving messages of a given topic by specifying the exact topic or by specifying a regular expression that defines interest in a set of possible topics. For a hierarchical publication data model, any part of the hierarchy can be specified as interest by a subscriber in using a wildcard notation to select all messages published by the specified topics.

The publish/subscribe matching problem has the standard interpretation and is defined as determining the set of subscribers based on their subscription expression for a given publication message. This problem is solved over the topic space, which is much simpler than its content-based counter part.

Topic-based publish/subscribe systems are distinguished by the qualities of service the system offers to its clients, such as various degrees of reliability, topic persistence, message ordering constraints, message delivery guarantees, and message delivery latencies constraints. Topic-based publish/subscribe relates to channel-based publish/subscribe in that publishing a message to a channel is similar to associating a message with a topic, which could be the name or identity of the channel. However, in topic-based publish/subscribe this association is reflected in the message itself, while in channel-based publish/subscribe the association is indirect, reflected by selecting a channel, which must not be represented in the message. Topic-based publish/subscribe has more limited filtering capabilities than content-based publish/subscribe, as the message is opaque to the system, while in content-based publish/subscribe the message structure and content is used for determining the set of recipients of a message.

Examples that follow the topic-based publish/subscribe model are the information bus [3], TIBCO's RendezVous product [5], and the series of Web services standards: WS Topics, WS Base Notifications, WS Brokered Notifications [1]. Elements of channel-based publish/subscribe can also be found in the Java Messaging Service [2], the OMG Data Dissemination Service [4], and other messaging middleware. However, these systems are not directly following the topic-based model as described above; rather these approaches are enriched with elements of message queuing, channel-based publish/subscribe, and content-based publish/subscribe.

Topic-based publish/subscribe is intended to support applications that need to selectively disseminate messages from one or more data source to several data sinks, where the mapping of sources to sinks changes dynamically. That is not all sources always communicate with the same sinks. The mapping of which source communicates with which sink is represented through associating topics with messages and subscribing to topics. Most existing systems allow the application to dynamically change subscriptions to topics. Also, applications of topic-based publish/subscribe exist that use the topic as a message log. For these applications the filtering capabilities of the topic-based model is not so important, but message order guarantees, reliability of the queues underlying each topic, and low message delivery latencies are crucial. There are many applications that follow these characteristics. Examples include system integration, selective information dissemination, system management, and database replication.

The term topic-based publish/subscribe is not used uniformly. Abstractions that exhibit the above described functionality are also often referred to as subject-based publish/subscribe systems that offer subject-based addressing to the applications using the system. Subject-based addressing means that interacting applications address each other by publishing messages associated with subjects and by subscribing to subjects of interest. The term subject and topic are used synonymously. Based on the subscriptions registered with the system, the system determines the set of recipients for a given message, without needing explicit address information that identifies that a given message is to be sent to a given destinations. Also, many messaging systems exhibit part of the above described functionality and are simply referred to as messaging systems, message-oriented middleware, or message queuing systems.

Cross-references
► Channel-Based Publish/Subscribe
► Publish/Subscribe
► Type-Based Publish/Subscribe

Recommended Reading

1. Chappell D. and Liu L. (ed). Web Services Brokered Notification 1.2 (WS-BrokeredNotification), working draft 01 edition, July 2004.
2. Hapner M., Burridge R., and Sharma R. Java Message Service. Sun Microsystems, version 1.0.2 edition, November 9th 1999.
3. Oki B., Pfluegl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed systems. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.

4. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07-01-01 edition, January 2007.
5. TIBCO. TIBCO Rendezvous, software release 8.1 edition, April 2008.

Topic-Directed Web Crawling

► Focused Web Crawling

Top-k Queries in P2P Systems

► Approximate Queries in Peer-to-Peer Systems

Top-K Selection Queries on Multimedia Datasets

AMÉLIE MARIAN
Rutgers University, Piscataway, NJ, USA

Synonyms

Ranked multimedia retrieval; Aggregation algorithms for middleware systems; Evaluation of fuzzy queries over multimedia systems

Definition

Traditionally, queries over structured (e.g., relational) data identify the exact matches for the queries. This exact-match query model is not appropriate for a multimedia dataset scenario where queries are inherently fuzzy – often expressing user preferences and not hard Boolean constraints – and are best answered with a ranked, or "top-k," list of the best matching objects. Efficient top-k query algorithms for such applications must take into account the specific challenges in accessing multimedia data. In particular, the query model should consider the access interfaces available to retrieve object attribute information, as well as the cost of retrieving this attribute information.

Historical Background

Content management in multimedia repositories is an important problem as more and more multimedia applications are developed. For example, digitization of photo and art collections is increasingly popular, multimedia mail and groupware applications are becoming widely available, and satellite images are being used for weather predictions. To access such large repositories efficiently, multimedia objects need to be queried via their attribute values, such as the date the multimedia object was authored, a free-text description of the object, and features like color histograms.

There are at least three major ways in which accesses to a multimedia repository differ from that of a structured database (e.g., a relational database). First, the data are inherently fuzzy: rarely does a user expect an exact match with the features of a multimedia object (e.g., color histogram). Rather, an object does not either satisfy or fail a condition, but has instead an associated grade of match [3,5]. Thus, an atomic query condition will not be a filter testing for an equality between two values (e.g., between a given color c and the color $O.c$ of an object O) as is usually the case in an exact query model scenario, but instead will assign a score representing the grade of match between the two values (e.g., *GradeColor*($c,O.c$)). Next, every condition on an attribute of a multimedia object may only be separately evaluated through calls to a system or index that handles that particular attribute. This is in contrast to a traditional database where, after accessing a tuple, all selection predicates can be evaluated on the tuple. Finally, the process of querying and browsing over a multimedia repository is likely to be interactive, and users will tend to ask for only a few best matches according to a ranking criterion.

Foundations

Existing query processing techniques for relational data cannot efficiently be applied to multimedia scenarios as object attribute information is often kept separate, possibly in different subsystems, and is typically expensive to retrieve. In addition, the fuzzy nature of the queries means that users are only interested in the best matches, making it unnecessary to evaluate every object.

Query Model

Consider a collection C of objects with attributes $A_1,...,A_n$. A top-k query over collection C simply specifies target values for each attribute A_i. Therefore, a top-k query q is an assignment of values $\{A_1 = q_1,...,A_n = q_n\}$ to the attributes of interest. The answer to the top-k query $q = \{A_1 = q_1,...,A_n = q_n\}$ over a collection of objects C and for a scoring function is a list of the k objects in the collection with the highest score for

the query. The final score that each object t in C receives for q is generally a function of a score for each individual attribute A_i of t. Typically, the scoring function that is associated with each attribute A_i is application-dependent. Top-k algorithms presented in the literature can be applied to a variety of aggregate scoring functions as long as they satisfy some monotonicity requirements [1,4,5,8].

Typically, multimedia attribute values (or scores) can only be accessed through specific interfaces. Two types of access to data, along with their associated costs, can be distinguished. The first type of access is sorted (or sequential) access, which allows retrieving objects through a list sorted by the objects' attribute scores (for instance, all images stored by degree of redness). The second type of access is random access, which allows to directly access the attribute score of a given object. A sorted access is usually cheaper than a random access as it can make use of sequential access to precomputed index structures. However, sorted access does require to access every object in the attribute's score order. The multimedia system may allow either sorted- or random-access, or both, for each attribute score, depending on the underlying subsystems.

Top-k Query Evaluation Algorithms

A naive brute-force top-k query processing strategy would consist of computing the score for the query for every object to identify and return k objects with the best scores. For large collections of objects, it is easy to see that this brute-force evaluation could be prohibitively expensive. Fortunately, the top-k query model provides the opportunity for efficient query processing, as only the best k objects need to be returned. Objects that are not part of the top-k answer, therefore, might not need to be processed. The challenge faced by top-k query processing techniques is then to identify the top-k objects efficiently, to limit the amount of processing done on non-top-k objects. To this end, various top-k query processing strategies have been presented.

The Threshold Algorithm

To process queries involving multiple multimedia attributes, Fagin et al. proposed a family of algorithms [3,4,5], developed as part of IBM Almaden's Garlic project. These algorithms can evaluate top-k queries that involve several independent multimedia "subsystems," each producing scores that are combined using arbitrary monotonic aggregation functions. The initial *FA* algorithm [3] was followed by "instance optimal"

query processing algorithms over sources that allow for sorted accesses and possibly random accesses (*TA* algorithm) or only for sorted accesses (*NRA* algorithm) [4]. In later work, Fagin et al. [5] introduced the TA_z algorithm, a variation of *TA* that also handles sources that only provide random-access interfaces. These algorithms rely on making dynamic choices for scheduling index lookups during query execution in order to prune low-scoring candidate items as early as possible. *TA* does not need an unbounded buffer, and dynamically considers object grades to decide when to stop retrieving new objects. Specifically, *TA* stops retrieving new objects when it finds a threshold grade G such that (1) at least k objects with grade G or higher have been identified and (2) no unretrieved object can have a grade greater than G.

Nepal and Ramakrishna [10] and Güntzer et al. [6] presented variations of Fagin et al.'s *TA* algorithm [4] for processing queries over multimedia databases. In particular, Güntzer et al. [6] reduce the number of random accesses through the introduction of more stop-condition tests and by exploiting the data distribution.

While these algorithms are proved "instance optimal," i.e, the consider the minimum number of objects needed to correctly identify the top-k answer, they completely evaluate each object they consider.

Algorithms based on Expensive Predicates Evaluation

Some works have built upon the *TA* family of algorithms to further improve query processing efficiency by reducing the number of expensive random accesses.

Marian et al.'s *Upper* algorithm [8] picks the most *promising* object-attribute pair to process at any given time based on the result of previous accesses. The *Upper* algorithm requires keeping track of partially evaluated object score bounds. By interleaving the processing of objects, and discarding objects that are not fully evaluated, *Upper* results in significant savings in random access costs. Marian et al. also proposed *TA-EP*, an optimization of *TA* that exploits existing techniques for processing selections with expensive predicates. In addition, Marian et al. [8] proposes extending the *Upper* algorithm to efficient parallel evaluation.

Chang and Hwang [1] presented *MPro*, an algorithm that relies on identifying *necessary* probes to optimize the execution of *expensive predicates* for top-k queries. Unlike *Upper*, *MPro* always evaluate attributes in the same order for every object. Chang and Hwang also briefly discussed parallelization

techniques for *MPro* and proposed the *Probe-Parallel-MPro* algorithm.

Filter/Restart Method

Algorithms based on Fagin et al.'s *TA* dynamically refine a threshold value G based on the status of evaluated objects. As a result, these algorithms can be processed continuously, until a solution is reached. In contrast, some techniques have focused on translating top-k queries into standard selection queries. While this approach allows using existing query evaluation and optimization implementations, it may require to "restart" a query, if the translation does not return at least k results.

In particular, Chaudhuri et al. built on Fagin's original *FA* algorithm and proposed a cost-based approach for optimizing the execution of top-k queries over multimedia repositories [2]. Their strategy translates a given top-k query into a selection (filter) query that returns a (hopefully tight) superset of the actual top-k tuples using data distribution information to estimate the value G that is expected to be the score of the kth object. Ultimately, the evaluation strategy consists of retrieving the top-k' tuples from as few sources as possible, for some $k' \geq k$, and then probing the remaining sources by invoking existing strategies for processing selections with expensive predicates.

Using Pre-computed Views

Another approach to top-k query evaluation is to use precomputed top-k query indexes. Various top-k queries, with different scoring functions, are evaluated to create indexes, which are used whenever a new top-k query is entered. A challenge of such an approach is to correctly identify the most efficient index for the new query. The *PREFER* system [7] uses pre-materialized views to efficiently answer ranked preference queries over commercial DBMSs. *PREFER* precomputes a set of materialized views that provide guaranteed query performance and, for any new top-k query, selects a near optimal set of views under space constraints.

Handling Joins

Top-k query evaluation algorithms over arbitrary joins have been presented for multimedia applications [1,9]. They use a ranking function that combines individual tuple scores. These algorithm handle the possible explosion in the number of results resulting from the join operation.

Key Applications

Multimedia Search

Typical search queries in multimedia systems require for fuzzy matches on predicates that are expensive to evaluate as the corresponding information is not stored in indexes (e.g., similarity to a user-specified image). Top-k algorithms are designed to minimize the number of accesses to the data, only focusing on those that are needed to identify the best query answers.

Information Integration

In scenarios where many sources may be accessed to answer a query (e.g., web databases, legacy systems), using algorithms that are designed to minimize the number of these expensive remote accesses is an important aspect of query processing efficiency.

Future Directions

Research on top-k query processing in multimedia scenarios has so far focused mostly on efficiency. Relatively little attention has been devoted to evaluating the quality and usefulness of the resulting top-k answers. In contrast, the design of good scoring functions for (relatively unstructured) text documents has been the main focus of the IR community for the last few decades. Many lessons and techniques from IR can be applied to a more structured multimedia scenario.

Cross-references

Recommended Reading

1. Chang K.C.-C. and Hwang S. Minimal probing: supporting expensive predicates for top-k queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 346–357.
2. Chaudhuri S., Gravano L., and Marian A. Optimizing top-k selection queries over multimedia repositories. IEEE Trans. Knowledge and Data Eng., 16(8):992–1009, August 2004.
3. Fagin R. Combining fuzzy information from multiple systems. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 216–226.
4. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 102–113.

5. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4), 2003.

6. Güntzer U., Balke W.-T., and Kießling W. Optimizing multi-feature queries for image databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 419–428.

7. Hristidis V., Koudas N., and Papakonstantinou Y. PREFER: a system for the efficient execution of multi-parametric ranked queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 259–270.

8. Marian A., Bruno N., and Gravano L. Evaluating top-k queries over web-accessible databases. ACM Trans. Database Syst., 29(2):319–362, 2004.

9. Natsev A., Chang Y.-C., Smith J.R., Li C.-S., and Vitter J.S. Supporting incremental join queries on ranked inputs. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 281–290.

10. Nepal S. and Ramakrishna M.V. Query processing issues in image (multimedia) databases. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 22–29.

Top-k XML Query Processing

▶ Ranked XML Processing

Topological Data Models

Erik Hoel
Environmental Systems Research Institute, Redlands, CA, USA

Synonyms

Topology; Topological fabric; Topological data structure

Definition

Topology is defined as a mathematical model used to define the location of and relationships between geographical phenomena. These topological relationships are independent of distance or direction. Topology may depict connectivity of one entity to another; for example, an edge will have topological relationships to it's from and to nodes.

Topology is useful with spatial data because many spatial modeling or geoprocessing operations do not require geometric coordinate values. For example, to find the shortest path between two nodes requires a list of which edges connect to each other and the cost of traversing along each edge. Geometric coordinates are only necessary to draw the shortest path after it is calculated.

More generally, topology, in the context of spatial data, can have several other meanings:

- A mathematical model of features in space (e.g., nodes, edges, and faces).
- A physical data model for efficient representation of feature data.
- A mechanism that can be used to ensure data quality (e.g., no gaps or overlaps between polygons).
- A mechanism that allows the management of shared geometry.
- A mechanism that facilitates navigation between features using topological relationships (e.g., equal, disjoint, intersects, touches, crosses, within, contains, overlaps, and relate – the nine topological relationships in the dimensionally extended nine-intersection model [5]).

A topological data model is used to represent collections of features that are assembled into a topology (or topological fabric). Topological data models come in many different variants (as described in the following), but the central theme for each of the models is the storage and representation of spatial data that forms a topological fabric.

Historical Background

Topological data structures have been used to represent geographic information for over 40 years [3,14]. The topological model has been the basis of a number of operational systems (see, for example DIME [3], GIRAS [11], ODYSSEY [13], ARC/INFO [1], TIGRIS [7], and TIGER [9]). Many of these systems have been based on binary file and in-memory data structures and supported a single-writer editing model on geographic libraries organized as a set of individual map sheets or tiles.

Topology has historically been viewed as a spatial data structure used primarily to ensure that the associated data forms a consistent and clean topological fabric. Topology is used most fundamentally to ensure data quality (e.g., no gaps or overlaps between polygons representing land parcels) and allow a GIS

to more realistically represent geographic features. Topology allows one to control the geometric relationships between features and maintain their geometric integrity.

Foundations

Topological data structures for representing geographic information are a standard topic in geographic information science (see [6], for example, for an excellent definition of the mathematical theory underlying this information model). In general, the topological data model represents spatial objects (point, line, and area features) using an underlying set of topological primitives. These primitives, together with their relationships to one another and to the features, are defined by embedding the feature geometries in a single planar graph. Such datasets are said to be "topologically integrated."

The model associates one or more topological primitives (i.e., nodes, edges, and faces; or 0-cells, 1-cells, and 2-cells in the TIGER parlance) with spatial objects of varying geometry type (i.e., points, lines, and polygons respectively). More specifically, a feature with point geometry is associated with a single node element, a feature with line geometry is associated with one or more edge elements, and a feature with polygon geometry is associated with one or more face elements. This is depicted in Fig. 1 as the generic topology model.
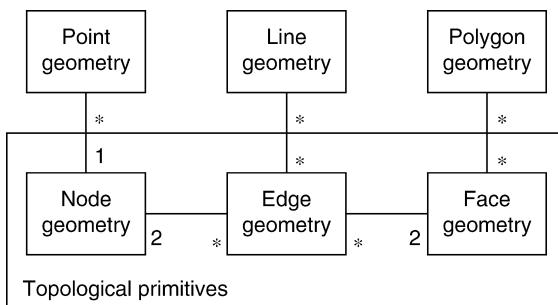
There are additional relationships between the topological elements themselves as is also shown in Fig. 1. A node element may or may not be associated with a collection of edge elements. A face element may be associated with one or more edge elements.

Finally, an edge element is associated with two node elements and two face elements. The relationships between nodes and faces may either be implicit or explicit.

The common representation of a topology is as a collection of topological primitives – i.e., nodes, arcs, and faces, with explicit relationships between the primitives themselves. For example, an arc would have a relationship to the face on the left, and the face on the right. With advances in GIS development, an alternative view of topology has evolved. Topology can be modeled as a collection of rules and relationships that, coupled with a set of editing tools and techniques, enables a GIS to more accurately model geometric relationships found in the world.

Topology, implemented as feature behavior and user specified rules, allows a more flexible set of geometric relationships to be modeled than topology implemented as a data structure. For example, older data structure based topology models enforce a fixed collection of rules that define topological integrity within a collection of data. The alternative approach (feature behavior and rules) allows topological relationships to exist between more discrete types of features within a feature dataset. In this alternative view, topology may still be employed to ensure that the data forms a clean and consistent topological fabric, but also more broadly, it is used to ensure that the features obey the key geometric rules defined for their role in the database.

Topological data structures, beginning with DIME in 1967 [3] have been used to represent features assembled into a topological fabric in a number of different ways over the past 40 years. In the following, the seven or so significant variants that have emerged during this period are described.

DIME Files

The US Census Bureau, as part of the New Haven Census Use Study of 1967, undertook the development of an explicit topological data model for their geographic data [3]. This system was called DIME (for Dual Independent Map Encoding) and was intended to facilitate the automation of detecting topological errors in the base geographic data. DIME files were based upon planar line segments defined by two endpoints. The line segments correspond to the Street Segment records as shown in Fig. 1. Each street

Topological Data Models. Figure 1. Generic topology model.

segment was associated with a start and end node identifier (endpoint), left and right side block and tract identifiers, as well as an address range. Drawbacks of the DIME model include the need to perform searches in order to assemble polygons from the street segment records, or to determine all the segments sharing a given node.

In Fig. 2, Main Street is represented as a collection of four records in the Street Segment Records. Each (line) segment is defined by two end points (there are no midspan shape points for the segment). The segment is associated with the start and end node, identifiers for the blocks and tracts on the left and right sides of the segment, as well as the low and high address ranges for each side of the segment. In addition, it is assumed that the segments are planar – segments may not cross each other.
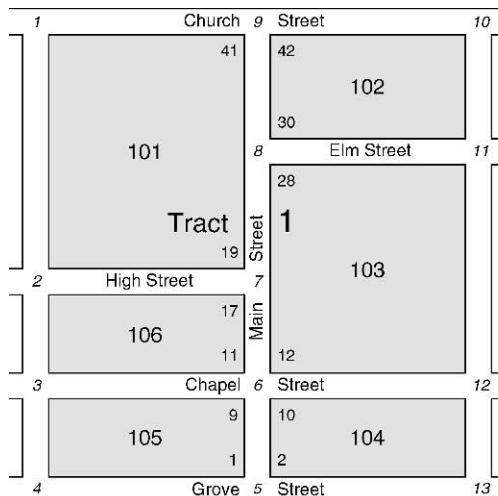
The "dual independent" portion of the DIME model reflects the redundant nature of how the topology is represented. Topological fabrics can be represented using the collection of relationships between edges and nodes, or nodes and faces, or faces and edges. With DIME, the topological correctness may be verified using either block chaining or node chaining. Block chaining involves finding all the segment records that have a given block on the left or right side. By rearranging the from/to orientations of each segment (and the associated left/right oriented attributes) such that the block is on the right side of each segment, it is possible to chain the nodes. This involves
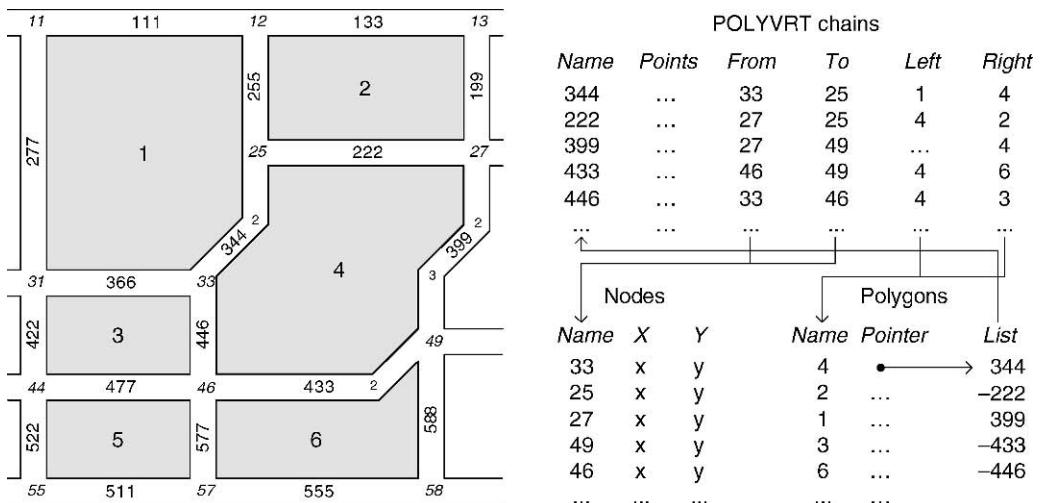
walking from the "to node" of one segment to the "from node" of another segment, continuing until all segments are visited. If there is a topological problem, it will not be possible to chain the blocks in this manner [2].

Due to the dual nature of DIME, it is also possible to chain the nodes in order to find topological -problems. Specifically, for a given node, select all segments that have it as a "to" or "from" node. Then, after rearranging the from/to orientations of each segment such that the node is always in the "to node" position, one may chain all the blocks surrounding the node (moving from the "block right" record in one segment to the "block left" record in another segment). Thus, because of the dual independent encoding of DIME, one may use two independent mechanisms to detect topological problems.

POLYVRT

The Harvard Laboratory for Computer Graphics and Analysis developed a topological data structure, termed POLYVRT, that was intended to serve as a data structure to facilitate interchange between various other data models [14]. POLYVRT extended the representational capabilities of DIME to allow planar segment chains to exist between nodes. This allowed line detail to be efficiently handled. In addition, POLYVRT enables the user to readily flip between segment chains and polygons.


Census address coding guide records

Street	Tract	Block	Low Address	High Address
Main	1	102	30	42
Main	1	103	12	28
Main	1	104	2	10
Main	1	105	11	9
Main	1	106	11	17
Main	1	101	19	41

DIME Street segment records

Street	Start Node	End Node	Tract Left	Tract Right	Block Left	Block Right	LLow Addr.	LHigh Addr.	RLow Addr.	RHigh Addr.
Main	5	6	1	1	105	104	1	9	2	10
Main	6	7	1	1	106	103	11	17	12	18
Main	7	8	1	1	101	103	19	27	20	28
Main	8	9	1	1	101	102	29	41	30	42

Topological Data Models. Figure 2. Example of the DIME approach to storing topology (adapted from [3]).

Topological Data Models. Figure 3. Example of the POLYVRT approach to storing topology.

Figure 3 contains an analogous example as shown in Fig. 2 but using the POLYVRT representation as well as adding shape points to various segments (e.g., segments 344, 399, and 433). A POLYVRT chain record contains a unique identifier, a pointer to the shape points (termed Points), identifiers of the from and to nodes, and identifiers of the polygons on the left and right sides. Records in the nodes table contain a unique identifier and an (x, y) coordinate value. Entries in the Polygons table have a unique identifier and a pointer to a list of all associated chains that compose the boundary of the polygon (e.g., for polygon 4, the list contains chains 344, 222, 399, 433, and 446).

GIRAS
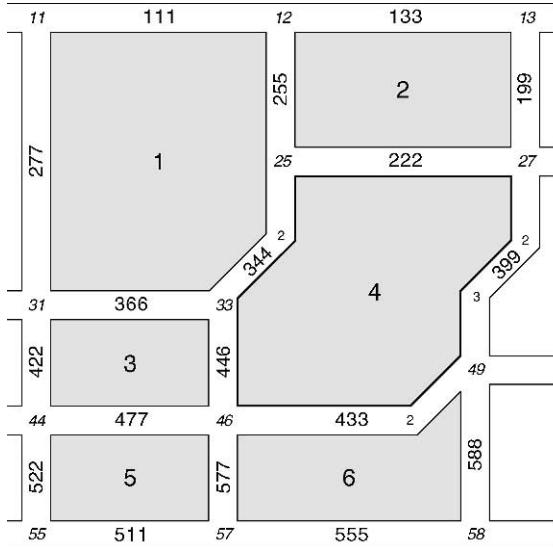
GIRAS (Geographical Information Retrieval and Analysis System) was developed by the US Geological Survey the mid 1970s [11]. The GIRAS topological data structure was motivated by the need to represent polygonal data. In addition, it was determined that it was more efficient to store certain types of data rather than recompute it. As a consequence, a large amount of ancillary data such as polygon perimeter and area was explicitly stored in the data structure. The data structure was based upon arcs and was considered a direct descendent of DIME and POLYVRT due to its topological similarities.

Figure 4 contains the same example dataset as shown in Figs. 2 and 3 except using the GIRAS topology

model. The GIRAS model differs from DIME and POLYVRT as various attributes unrelated to storing the topology are maintained. This includes the length as well as bounding rectangle information for the Arc records (Length, MinXY, and MaxXY). In addition, the Arc records store the attribute codes of the left and right polygons (this is not represented in the figure). The Polygon records similarly store the area and perimeter of the polygon, the bounding rectangle information, as well as the attribute code and the island count (number of islands found within the polygon). Note depicted in the figure is the identifier of the enclosing polygon if the polygon serves as an island to another polygon. Finally, there are structures for representing the vertex coordinates of the arcs (e.g., LastCoord in the Arc Records table), as well as another structure that maps arcs to polygons (the FAP file in GIRAS).

TIGER

TIGER (Topologically Integrated Geographic Encoding and Referencing) was developed by the US Census Bureau during the 1980s as an evolution of the earlier DIME-file model [9]. New features and capabilities were added (e.g., curve points for linear features) to the model in order to create a comprehensive system that could support all of the various censuses. TIGER was first used for the 1990 Census of Population and Housing.

Topological Data Models. Figure 4. Example of the GIRAS approach to storing topology.

The topology model within TIGER was based upon the two dimensional network model of Corbett [6]. This model used three topological primitives termed 0, 1, and 2-cells (analogous to nodes, edges, and faces in other topological data models).
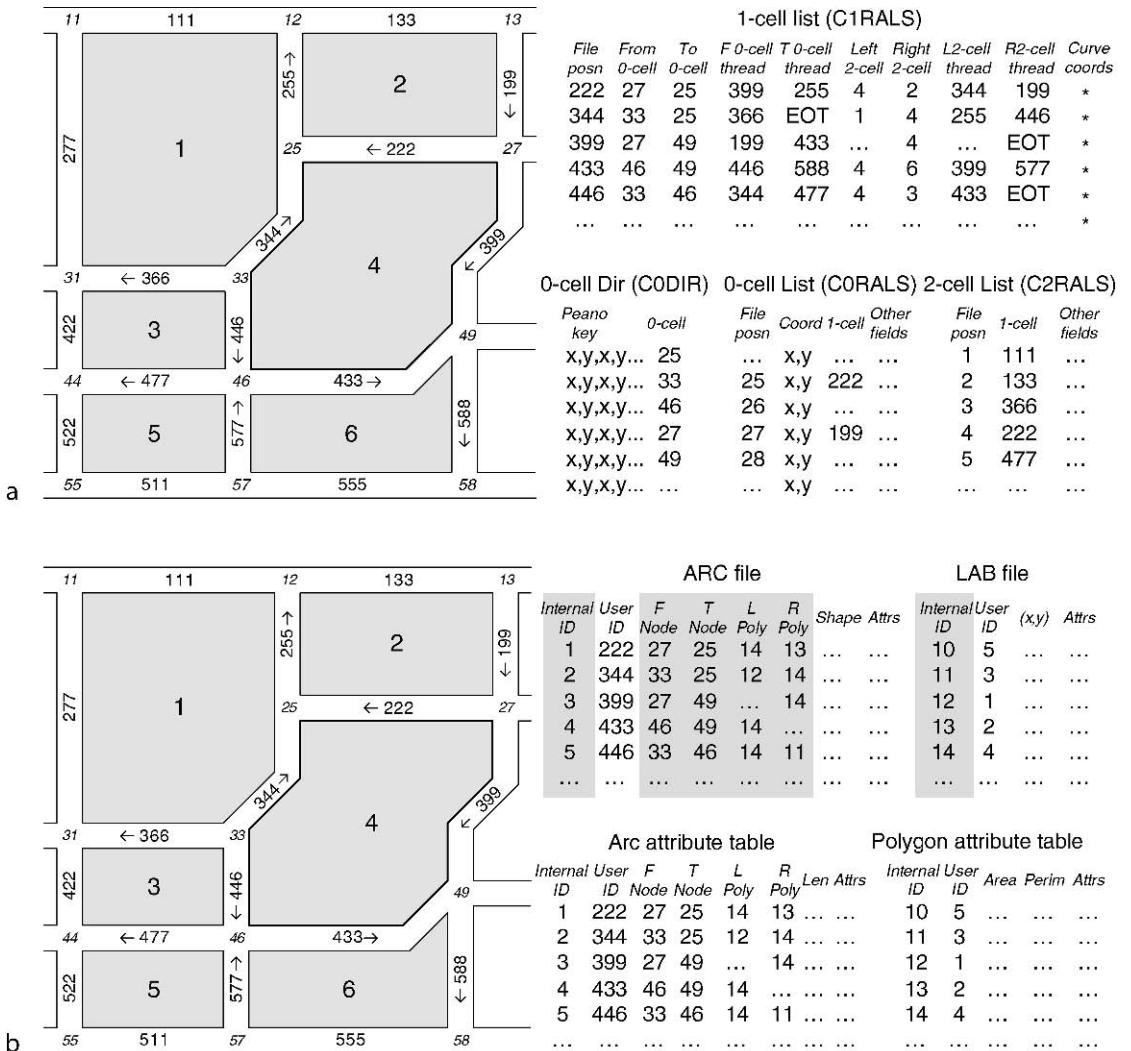
Figure 5 contains the same example dataset as shown in Figs. 2–4 except using the TIGER topology model (note – some liberty has been taken as in reality, the entries in the list tables are all consecutively numbered – this is not necessarily shown here; e.g., the file position in the C1RALS table). Within TIGER, 0 and 2-cells may be accessed via a directory mechanism. In the figure, the directory for 0-cells is shown (termed C0DIR). This directory contains a record for each 0-cell in the 0-cell List table (C0RALS). The directory entries are sorted by a simple Peano key (an alternating bit merging of longitude and latitude values for the associated point); this enables nearest point queries, etc. Each record in the directory table references a 0-cell in the 0-cell List table. The 0-cell List table is not geographically sorted. The 0-cell entries contain the x, y coordinate value of the point, and a pointer to the 1-cell List table (C1RALS) for the lowest value 1-cell (according to the file position) that is associated with the 0-cell at an endpoint. The entries in the 1-cell List table contain back-pointers to the from and to 0-cells, as well as threading pointer to other 1-cell records that enable a counter-clockwise traversal of all 1-cells associated with a given 0-cell (note that the last record in the thread contains a terminator – "EOT" in the figure). The 1-cell records also contain references to the 2-cells (faces) on the left and right sides as well as threading pointers to other 1-cells that enable the traversal of all 1-cells associated with a 2-cell. Finally, the 2-cell List (or C2RALS) table contains an entry for each 2-cell. Each entry contains a reference to the first 1-cell found in the C1RALS table that is associated with the 2-cell.
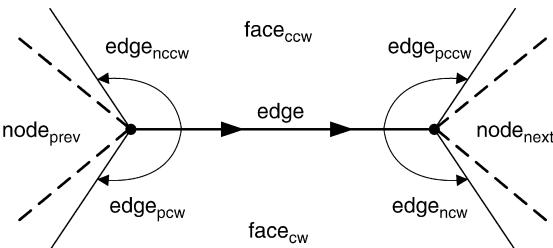
ARC/INFO Coverages

The motivating requirements behind the development of the ARC/INFO Coverage model were a model that had a strong theoretical basis (topology), as well as simplicity and efficiency (e.g., ability to support efficient geoprocessing functions such as polygon overlays and dissolves) [12].

The Coverage topological data model uses a collection of tables. End users are responsible for populating the ARC and LAB files. The entries in the ARC file correspond to line segments with optional attributes. In Fig. 6, the shaded portions of the ARC file correspond to system generated fields. Thus, the user specifies the user id field along with the shape (geometry) as well as any attributes. End users are also responsible for populating entries in the LAB (for label) file. Each label is used to specify the attributes

1-cell list (C1RALS)

File posn	From 0-cell	To 0-cell	F 0-cell thread	T 0-cell thread	Left 2-cell	Right 2-cell	L2-cell thread	R2-cell thread	Curve coords
222	27	25	399	255	4	2	344	199	*
344	33	25	366	EOT	1	4	255	446	*
399	27	49	199	433	...	4	...	EOT	*
433	46	49	446	588	4	6	399	577	*
446	33	46	344	477	4	3	433	EOT	*
...	...	...	...	...	...	...	...	...	*

0-cell Dir (C0DIR)

Peano key	0-cell
x,y,x,y...	25
x,y,x,y...	33
x,y,x,y...	46
x,y,x,y...	27
x,y,x,y...	49
x,y,x,y...	...

0-cell List (C0RALS)

File posn	Coord	1-cell	Other fields
...	x,y	...	...
25	x,y	222	...
26	x,y	...	...
27	x,y	199	...
28	x,y	...	...
...	x,y	...	...

2-cell List (C2RALS)

File posn	1-cell	Other fields
1	111	...
2	133	...
3	366	...
4	222	...
5	477	...
...	...	...

ARC file

Internal ID	User ID	F Node	T Node	L Poly	R Poly	Shape	Attrs
1	222	27	25	14	13	...	...
2	344	33	25	12	14	...	...
3	399	27	49	...	14	...	...
4	433	46	49	14	...	...	...
5	446	33	46	14	11	...	...
...	...	...	...	...	...	...	...

LAB file

Internal ID	User ID	(x,y)	Attrs
10	5	...	...
11	3	...	...
12	1	...	...
13	2	...	...
14	4	...	...
...	...	...	...

Arc attribute table

Internal ID	User ID	F Node	T Node	L Poly	R Poly	Len	Attrs
1	222	27	25	14	13	...	...
2	344	33	25	12	14	...	...
3	399	27	49	...	14	...	...
4	433	46	49	14	...	...	...
5	446	33	46	14	11	...	...
...	...	...	...	...	...	...	...

Polygon attribute table

Internal ID	User ID	Area	Perim	Attrs
10	5	...	...	...
11	3	...	...	...
12	1	...	...	...
13	2	...	...	...
14	4	...	...	...
...	...	...	...	...

Topological Data Models. Figure 5. Example of the TIGER approach to storing topology.
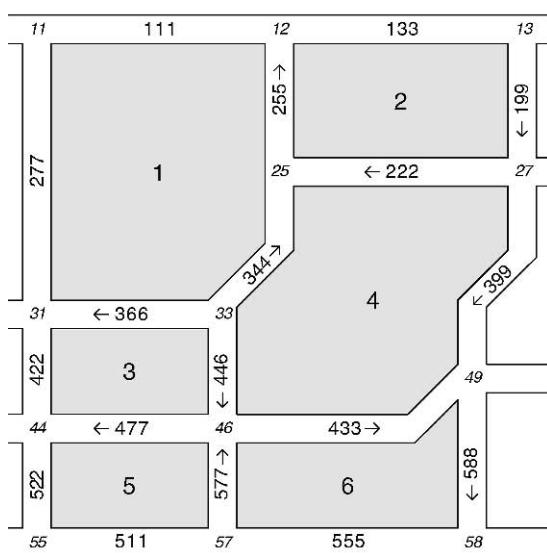
Topological Data Models. Figure 6. Example of the Coverage approach to storing topology (simplified).

that will be associated with a polygon in the topology. Following the population of the ARC and LAB files, the user will perform a topological integration and structuring of the data. The result of this is the population of the Arc Attribute Table (AAT) and the Polygon Attribute Table (or PAT). In addition, other fields in the ARC file are populated (i.e., the FNODE, TNODE, LPOLY, and RPOLY). There are other tables that form the Coverage data model that are not depicted in Fig. 6 (e.g., the Polygon Arc List, or PAL file) that provide additional explicit topological relationships.

Relational

Many topological data models stored in relational databases utilize a storage representation that relies upon a fixed length record. The winged-edge data structure [1], a fixed-length storage representation for representing the relationships between nodes, edges, and faces, simplified the task of representing

Edges

Edge ID	Start node	End node	Next LEdge	Prev LEdge	Next REdge	Prev REdge	LFace	RFace	Geom
344	33	25	255	−366	−222	−446	1	4	...
222	27	25	−344	−399	255	199	4	2	...
399	27	49	...	...	−433	−222	...	4	...
433	46	49	−399	446	588	577	4	6	...
446	33	46	433	−344	477	−366	4	3	...
...	...	...	...	...	...	...	...	...	...

Nodes

Node ID	Edge ID	Face ID	Geom
33	344	−	...
25	−222	−	...
27	399	−	...
49	−433	−	...
46	477	−	...
...	...	...	...

Faces

Face ID	Bdy Edge	Island EList	Island NList	Geom
1	255	−	−	...
2	−199	−	−	...
3	−446	−	−	...
4	−344	−	−	...
5	577	−	−	...
6	−577	−	−	...

Topological Data Models. Figure 7. Example of an edge and its four wings in the winged-edge data structure of Baumgart.

explicit topological data models within a relational database system. The edge record is the primal structure in this representation. For each edge, the identifiers of the start and end nodes (termed $node_{prev}$ and $node_{next}$ in Fig. 7), and the two faces, one in the clockwise traversal orientation ($face_{cw}$), and the other in the counterclockwise orientation ($face_{ccw}$) are represented. Finally, identifiers of four connected edges, two at each node, are stored. The four edges correspond to the previous and next edges found when traversing the two adjacent polygons in the clockwise ($edge_{pcw}$ and $edge_{ncw}$) and counterclockwise ($edge_{pccw}$ and $edge_{nccw}$) orientations.

A topological data model can be stored in a relational database. The topology model is represented using three tables – an edge table, a node table, and a face table. The edge table is essentially encoding the winged-edge data structure in addition to the geometry of the portion of the feature associated with the edge. The edge table utilizes negative identifiers to encode the orientation of the four edges found at the start or end nodes.

ArcGIS Geodatabase

The ArcGIS Geodatabase approach to modeling topology represents it as a collection of rules and relationships, coupled with a set of editing tools and techniques [8]. With the standard models (e.g., TIGER, Relational, etc.), it is possible to obtain topological primitives from feature geometry; similarly, it is possible to obtain feature geometry from topological primitives. Effectively, the geometry found in features is a dual representation of the geometry that would be found on the topological primitives. This topology model simplifies the generic explicit topology model and does not need to make both representations persistent. The process of topological integration (validation) results in vertex equality where features share underlying topological primitives. Given vertex equality, reconstruction of topological primitives is straightforward. Vertices on feature geometries in this scheme play the same role as that assigned to embedded foreign keys in data structures that explicitly model topological primitives.

Topological primitives and relationships are only instantiated during the process of topological validation or when required by the client application (this is similar to Intergraph's MGE where topology is selectively built but the topological primitives are not persisted in the RDBMS). The primary reason for this alternative approach is that it is easier (faster, more scalable) to recreate an index (e.g., the topological primitives) than to do all the bookkeeping necessary to make the topological primitives persistent and retrieve the primitives from the database while preserving the database transaction model. Additionally, it is frequently the case that the portion of the topological

primitives necessary for an operation is small relative to the entire topology (e.g., editing a few block groups in a localized area within TIGER).

In order for this approach to be viable from a performance standpoint, it is critical that there exists a high performance topology engine that validates the portion of the topology in question as well as instantiate the topological primitives for the given collection of features within the topology [15].

At a high level, this topology model consists of a collection of feature classes (homogeneous collections of features), topology rules, and other metadata used to support the validation model. This metadata includes dirty areas (areas that have not been validated following updates or edits), topology errors, and the cluster tolerance (i.e., the distance range in which vertices and boundaries are considered identical or coincident). Topological integrity is defined with respect to a collection of topology rules. Topology rules are used to define constraints on the permissible topological relationships between features in one or more feature classes that participate in the topology. The collection of topology rules that are associated with the topology are selected on the basis of which topological relationships are important for the user's model.

The validation process is a fundamental operation on a topology performed by a topology engine. The validation process on a topology is responsible for ensuring that the first three of Milenkovic's five normalization rules [10] on all spatial objects participating in the topology are respected:

1. No two vertices are closer than ε.
2. No vertex is closer than ε to an edge of which it is not an endpoint.
3. No two edges intersect except at their endpoints.

In addition, the validation process is responsible for checking all specified topology rules and generating topology errors at locations where rules are violated.

Topology rules are checked when the topology is validated. When a topology rule is violated, a topology error object is generated. This topology error may be represented as a special type of feature that may itself be persisted. At a later point following the validation, the user may then review the topology error objects, and the error conditions may be corrected. Topology rule violations do not prevent the validation operation from completing successfully.

Examples of topological rules that may be applied to polygon features include:

- The interiors of polygons in a feature class must not overlap (they may however share edges or vertices).
- Polygons must not have voids within themselves or between adjacent polygons (they may share edges, vertices, or interior areas).
- Polygons of one feature class must share all their area with polygons in another feature class (i.e., they must cover each other).

Key Applications

Cadastral databases, land use information systems, overlay processing, geoprocessing, topological analysis, dataset quality assurance/quality control (QA/QC).

Cross-references

▶ Geographic Information System
▶ Geographical Information Retrieval
▶ Network Data Model
▶ Spatial Data Analysis
▶ Spatial Data Types
▶ Spatial Network Databases
▶ Spatial Operations and Map Operations
▶ Topological Relationships

Recommended Reading

1. Baumgart B. A polyhedron representation for computer vision. In National Computer Conf., 1975, pp. 589–596.
2. Census Bureau. The DIME Geocoding System. Report No. 4, Census Use Study, US Department of Commerce, Bureau of the Census, 1970.
3. Cooke D. and Maxfield W. The development of a geographic base file and its uses for mapping. In Proc. 5th Annual Conf. Urban and Regional Information System Association, 1967, pp. 207–218.
4. Corbett J. Topological Principles in Cartography. Technical Paper 48. Bureau of the Census, Washington, DC, 1979.
5. Egenhofer M., Clementini E., and Di Felice P. Topological relations between regions with holes. Int. J. Geograph. Inform. Syst., 8(2):129–142, 1994.
6. Güting R. and Schneider M. Realm-based spatial data types: the ROSE algebra. VLDB J., 4(2):243–286, 1995.
7. Herring J. TIGRIS: topologically integrated geographic information system. In Proc. 8th Int. Symp. on Computer Assisted Cartography, 1987, pp. 282–291.
8. Hoel E., Menon S., and Morehouse S. Building a robust relational implementation of topology. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 508–524.
9. Marx R. The TIGER system: automating the geographic structure of the United States. In Introductory Readings in

Geographic Information Systems. Peuquet Marble Taylor & Francis, London, 1990.

10. Milenkovic V. Verifiable implementations of geometric algorithms using finite precision arithmetic. Artif. Intell., 37(1–3):377–401, 1988.

11. Mitchell W., Guptill S., Anderson K., Fegeas R., and Hallam C. GIRAS: A geographic information retrieval and analysis system for handling land use and land cover data: US Geological Survey Professional Paper 1059, GPO, Washington, DC, 1977.

12. Morehouse S. ARC/INFO: a geo-relational model for spatial information. In Proc. 7th Int. Symp. on Computer Assisted Cartography, 1985, pp. 388–397.

13. Morehouse S. and Broekhuysen M. ODYSSEY User's Manual. Laboratory for Computer Graphics and Spatial Analysis, Harvard Graduate School of Design, Cambridge, MA, 1982.

14. Peucker T. and Chrisman N. Cartographic data structures. Am. Cartograph., 2(1): 55–69, 1975.

15. van Roessel J. A new approach to plane-sweep overlay: topological structuring and line-segment classification. Cartograph. Geograph. Inform. Syst., 18(1), 1991.

Topological Data Structure

Topological Fabric

Topological Relationships

Paolino Di Felice, Eliseo Clementini
University of L'Aquila, L'Aquila, Italy

Definition

Topological relationships describe qualitative properties that characterize the relative position of spatial objects. *disjoint, meet, overlap,* and *inside* are few examples (Fig. 1).

Topology is considered the most primitive kind of spatial information, since a change in topology implies a change in other geometric aspects, while the opposite is not true. Generally speaking, topological properties are those that do not change after transformations like rotation, translation, scaling, and rubber sheeting.

Historical Background

Topological relationships have been studied extensively in a number of diverse disciplines since the beginning of the 1990s, achieving theoretical results which have constituted the basis for the definition of most of the topological operators today being part of the SQL dialects supported by commercial DBMSs (e.g., IBM-DB2, Oracle, PostGIS/PostgreSQL, . . .). The implementations are all based on the OpenGIS Consortium specifications [8] and ISO/TC 211 standard.

Foundations

The mathematical background behind the published contributions about topological relationships is constituted either by the *point set topology* or *spatial logic.* The study of topological relationships also depends on the embedding space, prevalently assumed to be the two-dimensional Euclidean space.

The major results that appeared in the literature can be clustered in the *three* main groups briefly discussed below.

Topological Relationships for Simple Objects

At the conceptual level, spatial objects can be modeled as *points, lines,* and *areas. Simple lines* are one-dimensional, continuous features embedded in the plane with two end points; *simple areas* are two-dimensional point sets topologically equivalent to a closed disc (Fig. 2).
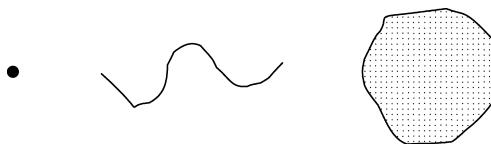
Topological Relationships. Figure 1. Examples of binary topological relationships between objects A and B (the biggest): (a) <A, *disjoint*, B>, (b) <A, *meet*, B>, (c) <A, *overlap*, B>, and (d) <A, *inside*, B>.

The two conceptual approaches, upon which almost all publications in this field have been based, are the *9-Intersection model* [5] and the *RCC model* [3]. Despite rather different foundations (the former relies on point set topology [6], the latter on spatial logic), both methods come to very similar results. Further relevant contributions belonging to this group are: [1,2,4].
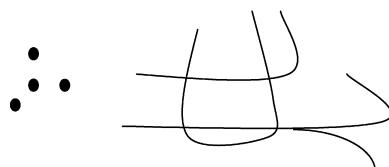
The model proposed, in 1991, by Egenhofer and Franzosa, [4], for classifying topological relationships between pairs of 2D area features represents the starting point of the research in the field and the basis for the efficient implementation of the theory on top of commercial query languages [10]. Their classification is based on the cross intersection of the boundaries and interiors of the two features. The four values are collected into a two-by-two matrix, called the 4-intersection; while the approach is called the *4-IntersectionMethod* (4IM).

Topological Relationships for Complex Objects

An important advancement of the results about topological relationships, with respect to those based on the assumption of simple objects, was achieved by extending the definitions of point, line, and area, in order to take into account finite sets of isolated points as a single complex point feature, lines having separations, more than two end-points, and possibly self-intersections, and, finally, complex areas having both separations and holes (Fig. 3). In the reality, complex features are far more common than simple ones: [1,11].

Topological Relationships for Objects with Vague Boundary

The models belonging to the previous two groups are applicable only to features whose geometry is exactly known (often called *crisp* spatial objects). Examples of crisp objects are mainly man-made artifacts like land parcels, buildings, and roads. But the reality reveals that the boundaries and extent of most spatial objects cannot be precisely determined. Examples of non-crisp features are: population density, vegetation, oceans, clouds, soil type, Spanish speaking areas, etc.

Three main alternatives have been proposed to model non-crisp spatial objects:

1. Models based on *fuzzy sets*. They allow a fine-grained modeling of vague spatial objects but are computationally rather expensive with respect to data structures and algorithms.
2. Models based on *rough sets*. They work with lower and upper approximations of spatial objects.
3. Models based on *crisp spatial objects*. They extend data models, type systems, and concepts for crisp spatial objects to vague spatial objects.

A discussion of the differences of these approaches can be found in [9], together with links to pertinent references.
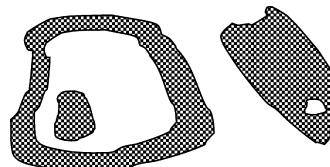
Table 1 summarizes the different subfields in the study of topological relationships discussed above.

Key Applications

The following applications are some examples among the many that benefit from dealing with topological relationships.

Geographic Information Systems (GISs)

Topological queries are relevant when dealing with spatial data. Today's GIS applications use a huge amount of spatial data. Formal models and efficient algorithms are of primary importance to reach optimal solutions.

Topological Relationships. Figure 2. Examples of a simple point, a simple line, and a simple area.

Topological Relationships. Figure 3. Examples of a complex point, a complex line, and a complex area.

Topological Relationships. Table 1. The different subfields behind the study of topological relationships

Geometry	Boundary
Simple	Crisp
	Vague
Complex	Crisp
	Vague

Qualitative Spatial Reasoning (QSR)

Topological relations capture the everyday common-sense knowledge of space. QSR makes this knowledge explicit, so that, given appropriate reasoning techniques, a computer can make predictions about spatial relations in a qualitative manner, without recourse to an intractable or unavailable quantitative model [3].

Geospatial Semantic Web (GSW)

The Geospatial Semantic Web has become one of the most prominent research themes in geographic information science. In fact, the wide availability of geo-referenced data on the web would make it possible to index and query information based on the spatial attributes. This approach would be facilitated by using qualitative spatial relations, such as topological relations, since people are much more inclined to query web pages through natural language, instead of metric measurements.

Future Directions

Despite of the huge amount of theoretical studies about topological relationships done so far, there is still a lot of work to be done.

For example, the mapping of the topological relationships into operators to be included in future releases of spatial query languages, as well as the development of processing strategies for their efficient evaluation are still open issues. A notable contribution in this direction is constituted by a very recent paper by Praing and Schneider, [10].

Furthermore, major attention needs to be paid with respect to complex objects characterized by uncertainty in order to: a) identify suitable spatial data types for their modeling, b) design a minimal set of operations and predicates defined on top of them, and c) proceed to their integration into the query language of existing DBMSs.

Experimental Results

The IBM DB2 UDB system supports the modelling and the management of spatial data through the so-called *Spatial Extender* (briefly, SE) subsystem [7] which is fully conformant with the OGC Simple Features Specification for SQL [8], starting with version 8.2.

DB2 SE supports four different spatial data formats: a) Well-known text (WKT) representation, b) Well-known binary (WKB) representation, c) Shape representation, and d) Geography Markup Language (GML) representation.

Table below provides two examples according to the WKT text representation (the numerical values represent X-Y coordinates).

Geometry type	WKT representation	Comment
Point	`point(10,20)`	A 2D point
Polygon	`polygon((0 0,0 40,40 40,40 0,0 0))`	A 2D polygon

{ST_Geometry, ST_Point, ST_LineString, ST_Polygon, ST_GeometryCollection, ST_MultiLineString,ST_MultiPolygon ST_MultiPoint} is the set of geometric data types being part of the SQL/DDL language running under DB2 SE.

DB2 SE implements a long list of spatial functions conceptually grouped into five macro-categories: a) *data exchange format functions, b) comparison functions, c) functions that return information about properties of geometries, d) functions that derive new geometries from existing ones, and e) miscellaneous functions.* The *comparison functions* implement the topological operators.

In order to give the flavour of how they look like and how easily they can be called as part of SQL statements, a spatial database storing descriptive and spatial data about sites of interest and counties is taken into account.

The SQL/DDL scripts below provide the definition of the corresponding tables according to the DB2 SE syntax:

```
CREATE TABLE sitesOf_interest (id SMAL-
LINT, geometry ST_POINT);
CREATE TABLE counties (id SMALLINT, geom-
etry ST_POLYGON);
```

The SQL/DML scripts below insert 1 and 2 tuples into the previous tables, respectively:

```
    INSERT   INTO   sitesOf_interest  (id,
geometry)
    VALUES   (1,   ST_Point(10,20,1),   (2,
ST_Point(41,41,1));
    INSERT INTO counties (id, geometry)
    VALUES (100, ST_Polygon('polygon ((0 0,0
40,40 40,40 0,0 0))', 1));
    INSERT INTO counties (id, geometry)
    VALUES (200, ST_Polygon('polygon ((1 1,1
40,40 40,40 1,1 1))', 1));
```

Notice that 1 identifies the *spatial reference system* for the resulting geometry.

The (incomplete) list of available *topological operators* are:

```
    ST_Disjoint,   ST_Touches,   ST_Equals,
ST_Contains, ST_Overlaps, ST_Crosses, etc.
```

An example of SQL usage of ST_Contains follows:
Syntax:

```
    ST_Contains(geometry1,geometry2)
```
Meaning:

```
    ST_Contains returns 1 if geometry1 contains
geometry2, 0 otherwise.
```

The query: *Determine the counties where the points of interest are located in.*

```
    SELECT poly.id AS polygon_id, pts.id AS
    point_id
    FROM sitesOf_interest pts, counties poly
    WHERE ST_Contains (poly.geometry, pts.
    geometry)
```

In summary, RDBMSs with spatial extensions like those featured by the IBM DB2 SE are:

- Reach of data types and functions to deal with geometry. This extends significantly the expressiveness of the relational data model and of SQL. It follows that writing ad hoc applications in high level programming languages, to make spatial analysis, is much easier than before
- Not problematic to be used for people accustomed to use SQL

Cross-references
▶ Dimension-Extended Topological Relationships
▶ Spatial Data Types
▶ SQL

Recommended Reading
1. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. Inf. Sci., 90(1–4):121–136, 1996.
2. Clementini E., Di Felice P., and van Oosterom P. A small set of formal topological relationships suitable for end-user interaction. In Proc. 3rd Int. Symp. Advances in Spatial Databases, 1993, pp. 277–295.
3. Cohn A.G., Bennett B., Gooday J., and Gotts N. RCC: a calculus for region based qualitative spatial reasoning. GeoInformatica, 1:275–316, 1997.
4. Egenhofer M.J. and Franzosa R. Point-set topological spatial relations. Int. J. Geogr. Inf. Syst., 5(2):161–174, 1991.
5. Egenhofer M.J. and Herring J. Categorizing binary topological relationships between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, 1991.
6. Gaal S. Point Set Topology. Academic Press, New York, NY, 1964.
7. IBM DB2. Spatial extender user's guide and reference (Vers. 8). 2004.
8. Open Geospatial Consortium. OpenGIS simple features specification for SQL. OpenGIS Project Document, 99–049, 1999.
9. Pauly A. and Schneider M. Topological predicates between vague spatial objects. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 418–432.
10. Praing R. and Schneider M. Efficient implementation techniques for topological predicates on complex spatial objects. GeoInformatica, 2007.
11. Schneider M. and Behr T. Topological relationships between complex spatial objects. ACM Trans. Database Syst., 31(1): 39–81, 2006.

Topology
▶ Topological Data Models

Toponyms
▶ Gazetteers

Tour
▶ Dynamic Graphics

TP
▶ XML Tree Pattern, XML Twig Query

TP Monitor

▶ Transactional Middleware

TPQ

▶ XML Tree Pattern, XML Twig Query

Traditional Concurrency Control for Replicated Databases

BETTINA KEMME
McGill University, Montreal, QC, Canada

Synonyms

Traditional replica and concurrency control strategies;
Traditional data replication

Definition

Since the beginnings of distributed computing, the database community has developed strategies for replicated data management. The basic idea is that each "logical" data item has one or more physical data copies, also called replicas, that are distributed across the database servers in the system. Early work on data replication provided a framework to describe transactions in a replicated environment and developed concurrency control mechanisms to control their execution. The formalism and the techniques developed in this early work have been the foundations for much of the further research on database replication. It considered strong consistency requirements where the replicated system behaves similar to a non-replicated system. *Replica control* was introduced as the task of translating the read and write operations of transactions on logical data items into operations on the physical data copies. One-copy-serializability was developed as a first – and very strong – correctness criterion defining when the concurrent execution of transactions in a replicated system is equivalent to a serial execution of these transactions over a single logical copy of the database. Replica control was combined with *concurrency control mechanisms* in order to provide one-copy-serializable transaction execution.

Historical Background

Replication became a hot topic in the early 1980s. In their book "Concurrency Control and Recovery in Database Systems" [2], Bernstein et al. presented a thorough formalism to reason about the correctness of transaction execution and concurrency control mechanisms in central, distributed and replicated database systems. Their definitions of serializability and one-copy-serializability (1SR) are still used to reason about the correctness of transactional systems. Early work on replication took as baseline concurrency control strategies used in non-replicated or distributed databases, extended them and combined them with replica control in order to provide one-copy-serializability [2,3]. Furthermore, the correctness of execution despite site or communication failures has been analyzed thoroughly [1,4]. Work done in this early phase is very visible in textbooks on database systems and distributed systems [6], and builds part of the foundations of academic education in this area. In 1996, Gray et al. [5] indicated that these traditional approaches provide poor performance and do not scale as they commit transactions only if they have executed all their operations on all (available) physical data copies. Many advanced replication schemes have been developed since then. Nevertheless, they reuse many of the base techniques developed in the traditional replication algorithms.

Foundations
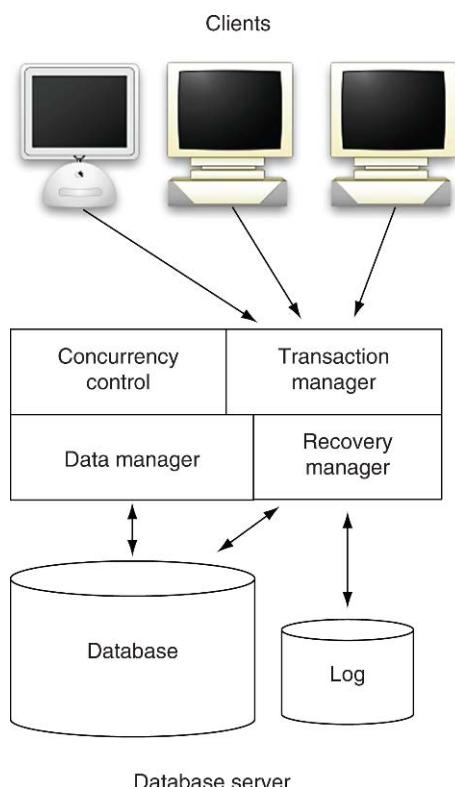
Transactions in a Non-Replicated System

The formalism that describes transactions and their execution in a replicated database is derived from the transaction model in a non-replicated system. In a non-replicated system, a database consists of a set of data items $x, y, \ldots$. A transaction T_i is a sequence of read operations $r_i(x)$ and write operations $w_i(x)$ on data items that build a logical unit. The database system should provide the transactional properties atomicity, consistency, isolation and durability (see the entry *ACID properties*). Among them, atomicity and isolation require actions in a replicated system that go beyond the tasks in a non-replicated system.

Atomicity means that a transaction T_i either terminates with a commit operation (indicated as c_i) and all its write operations remain effective in the database, or with an abort operation (indicated as a_i), in which case all already executed write operations are undone before the transaction terminates.

Isolation requires that even if transactions execute concurrently in the system, each transaction should have the impression it executes isolated on the data. In particular, when two operations conflict, i.e., they are from different transactions, want to access the same data item and at least one is a write, the execution order matters. Given a set of transactions, a history describes the order in which the database server executes the operations of these transactions. The traditional correctness criterion in a non-replicated system is *serializability*. It requires a history to be equivalent to a serial history where the same transactions are executed serially one after the other. Equivalence typically refers to executing all conflicting operations in the same order.

The execution of transactions is controlled by several components of the database system (see Fig 1). The client starts a transaction and then submits the individual operations of the transaction (including a final commit or abort request). These requests are intercepted by the transaction manager which keeps track of active transactions. The individual read and write operations are forwarded to the concurrency control module that controls when operations are executed by the data manager in order to provide serializability. The recovery manager makes changes persistent at commit time or triggers undo operations in case of abort. In real database systems, clients submit SQL statements that can access and manipulate many records of different tables. However, the abstraction into simple read and write operations on data items allows for a clear and powerful reasoning framework.

Transaction Execution in a Replicated System

In a replicated database, there is a set of database servers A, B,..., also referred to as sites, and each logical data item x has a set of physical copies x^A, x^B, ... where the index refers to the server on which the copy resides. In full replication, each data item has a copy on each server, while using partial replication, it has only copies on a subset of the servers.

Clients

Concurrency control | Transaction manager

Data manager | Recovery manager

Database

Log

Database server

Traditional Concurrency Control for Replicated Databases. Figure 1. Transaction components in non-replicated database systems.

Execution Model   As replication should be transparent to clients they continue to submit operations on the logical data items. Replica control has to map an operation $o_i(x)$, $o_i \in \{r,w\}$, of transaction T_i into operations on the physical copies of x, e.g., $o_i(x^A),o_i(x^B),....$ Given the mapping for a set of transactions, when executing these transactions in the replicated system each database server A produces a local history showing the execution order of all the operations performed on the copies maintained by A.

The most common execution model for transactions in a replicated environment is to perform a read operation on one data copy while write operations update all copies. This is referred to as ROWA (or read-one-write-all). As most database applications typically have more read than write operations is makes sense to provide fast read access and only penalize write operations.

A problem of ROWA is that if one copy is not accessible, write operations cannot be performed anymore on the data item. In order to be able to continue even if failures occur, ROWAA (read-one-write-all-available) needs to be used. It does not require to perform updates on copies that are currently not available. An alternative are quorum protocols that require both read and write operations to access a quorum of copies.

Isolation One-copy-serializability was the first correctness criterion for replicated histories. The execution of a set of transactions in a replicated environment is one-copy-serializable if it is equivalent to a serial execution over a single, non-replicated (logical) database. Defining equivalence is not straightforward since the replicated system executes on physical copies while the non-replicated on the logical data items. The handling of failures further complicates the issue.

Atomicity Guaranteeing atomicity in a replicated system requires that all sites that have performed operations on behalf of a transaction agree on the outcome (commit or abort) of a transaction. Traditional replication solutions typically achieve atomicity by running a commit protocol at the end of a transaction. Commit protocols, such as the Two-Phase-Commit Protocol, are a special form of agreement protocol in a distributed system. The challenge here is to define a protocol that works correctly in the presence of crash and network failures.

This entry does not further look at failures but focuses on providing isolation in a ROWA system.

Replica and Concurrency Control in a Replicated System
There are many ways to distribute concurrency and replica control tasks across the system. Figure 2a shows a centralized architecture. Each individual server is only responsible for data and recovery management. Additionally, there is one central transaction manager, one concurrency control module and one replica control module in the system. Together, they control the execution of all operations in the system. They could be all located in one of the data servers, or, as shown in the figure, in a special middleware. Figure 2b shows a distributed architecture where each site has its own transaction manager, concurrency and replica control modules. Decisions on where and when to execute operations are made locally at each site. A hybrid approach is shown in Fig 2c. Each database server has its traditional transaction and concurrency control modules. A middleware layer controls transactions globally and performs replica control. It relies partially on the concurrency control modules of the individual servers. As this might not be enough for a globally correct execution, the middleware performs some additional scheduling. Other combinations of distribution are also possible.

Given a concurrency control mechanism developed for a non-replicated system, there exist many ways to extend it to a replicated system. The following depicts a few examples.

Strict Two-Phase Locking (S2PL) is probably the best known concurrency control mechanism. In this case, the concurrency control module implements a lock manager. Using S2PL, a transaction has to acquire a shared lock on data item x before performing a read on x, and an exclusive lock on x before writing x. An exclusive lock on x conflicts with other shared and exclusive locks on x. If a transaction requests a lock and another transaction holds a conflicting lock, the requesting transaction has to wait. Only when a transaction terminates it releases all its locks, which then can be granted to waiting transactions. S2PL guarantees serializability because the order in which locks are granted for the first pair of conflicting operations between two transactions determines the serialization order. Deadlocks can occur. A deadlock involving two transactions can happen if the transactions have two pairs of conflicting operations and execute them in different order.
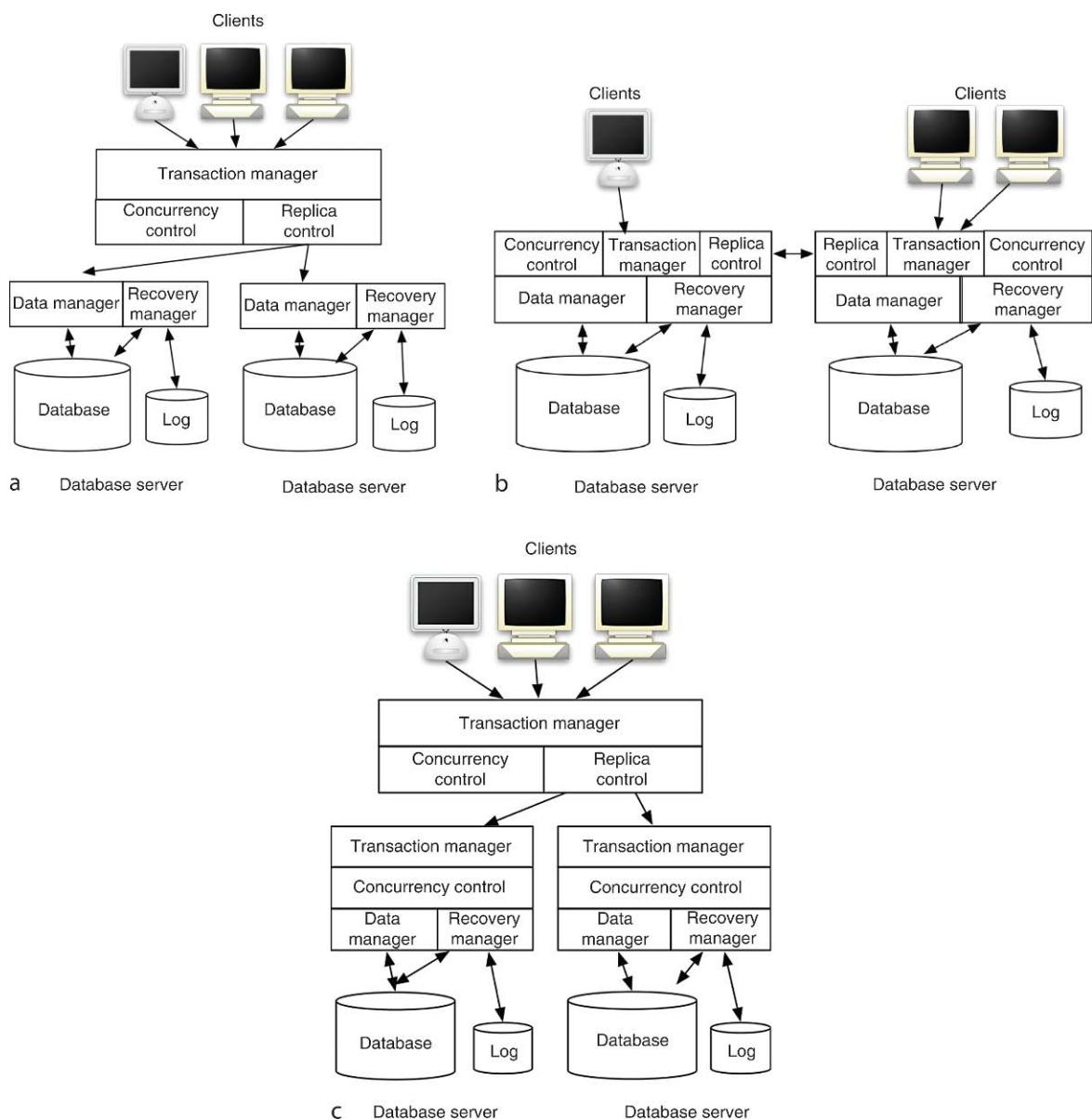
Applying S2PL in a replicated system with a centralized architecture (Fig 2a), clients submit their operations to the global transaction manager. The transaction manager gets the appropriate lock via the lock manager and then the replica control module transfers each read operation to one database server with a copy of the data item, and write operations to all servers with copies. In the distributed architecture (Fig 2b), a client connects to any site. The execution of individual operations is illustrated in Fig 3a. The figure shows the message exchange between a client and two sites. When the client submits a read operation on logical data item x to the local server A, a local shared lock is acquired on the local physical copy ($sl(x^A)$) and the read operation executes locally. If it is a write operation, an exclusive lock ($xl(x^A)$) is acquired locally and the operation executes locally. At the same time, the operation is forwarded to the server B. B, upon receiving the request, acquires a lock on the local copy, performs the operation, and sends a confirmation back to A. When A has received all confirmations, it sends the confirmation back to the client.

Architectural Comparison Comparing how well S2PL maps to the two architectures reflects well the principle trade-offs between a centralized and a decentralized architecture.
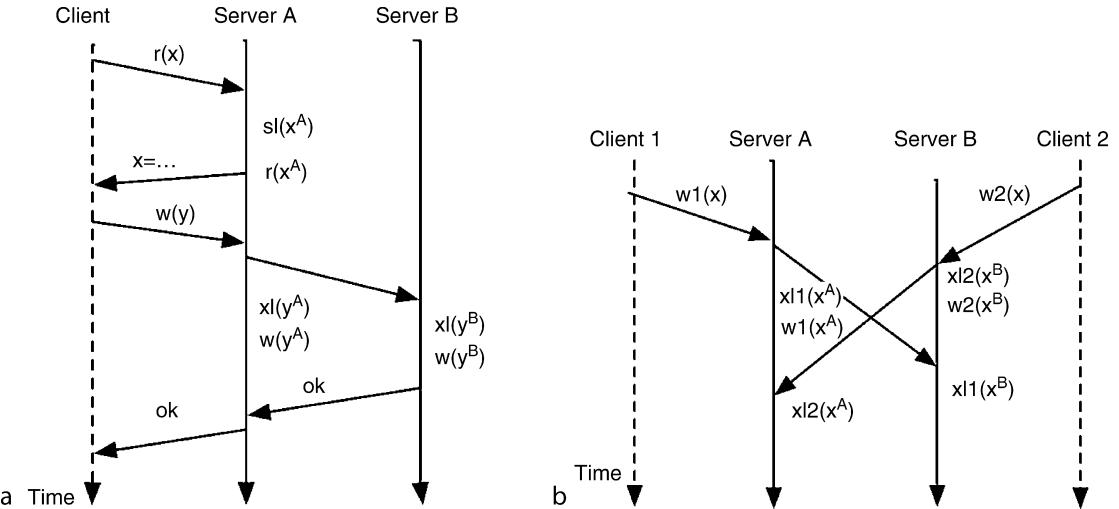
In favor of a centralized architecture. In principle, a central component makes the design of coordination algorithms often simpler. It directs the flow of execution, and has the *global knowledge* of where copies are located. One central concurrency control module serializes all operations. In the distributed architecture, the flow of execution is more complex as no single component has the full view of what is happening in the system.

While the centralized architecture acquires one lock per each operation on a logical data item, the distributed architecture acquires locks per data copies. Thus, a write operation involves many exclusive locks, adding to the complexity. The distributed architecture has the additional disadvantage of potential *distributed deadlocks*: there is no deadlock at any site locally but globally, a deadlock has occurred. Figure 3b depicts an example execution where a distributed deadlock occurs although the transactions both only access a single data item (something not even possible with the centralized architecture). *T*1 first acquires the lock on server *A* which forwards the request to server *B*. Concurrently,

Traditional Concurrency Control for Replicated Databases. Figure 2. Concurrency Control and Replica Control Architecture.

Traditional Concurrency Control for Replicated Databases. Figure 3. Distributed Transaction Execution.

$T2$ acquires the lock first on B and then requests it on A. At A, $T2$ has to wait for $T1$ to release the lock, on B, $T1$ waits for $T2$. The deadlock is distributed since no single server observes a deadlock. Such deadlocks need to be resolved via timeout or a deadlock detection mechanism, which in turn, could be implemented centrally or distributed.

In favor of a decentralized architecture. A central middleware is a potential bottleneck and a single point of failure. In contrast, in the distributed architecture, if ROWAA is used, the system can continue executing despite the failure of individual sites.

Furthermore, the middleware approach has an extra level of indirection. In the above example algorithm, this leads to four messages per read operation (a pair of messages between clients and middleware, and a pair between middleware and one database server). In contrast, the distributed architecture has two messages (between the client and one database server).

A further disadvantage is that the global controller does not have access to the data manager modules of the database servers. For example, assume clients submit SQL statements. The middleware cannot know what records will actually be accessed by simply looking at the SQL statement. Such information is only available during the execution. Thus, the central lock manager might need to set locks on the entire relation. In contrast, the distributed architecture can execute an SQL statement first locally, lock only the tuples that are

updated, and then forward the update requests on the specific records to the other servers, allowing for a finer-grained concurrency control. Generally, a tighter coupling often allows for a better optimization.

Optimistic Concurrency Control   A last example looks at optimistic concurrency control (OCC) [7]. In a non-replicated system, a write operation on x generates a local copy of x. A read on x either reads the local copy (if the transaction has previously written x) or the last committed version of x. At commit time of a transaction T_i, validation is performed. If validation succeeds, a write phase turns T_i's local copies into committed versions and T_i commits. Otherwise, T_i aborts. In the simplest form of OCC, validation and write phase are executed in a critical section. The validation order determines the serialization order. Therefore, validation of T_i fails if there is a committed transaction T_j that is concurrent to T_i (committed after T_i started), and T_j's writeset (data items written by T_j) overlaps with T_i's readset (data items read by T_i). As T_i validates after T_j it should be serialized after T_j, and thus, read what T_j has written. In the concurrent execution, however, it might have read an earlier version. Therefore, it needs to be aborted. Optimistic execution assumes conflicts are rare, and thus, it is sufficient to detect them at the end of transaction.

One possible implementation of OCC in a system using full replication uses the hybrid architecture of

Fig 2c. Upon the first operation of a transaction, the middleware starts a transaction and then executes all operations at one of the database servers. The local OCC of the server retrieves the latest committed versions and keeps track of local copies. At commit time the middleware retrieves the read- and writesets from the server at which the transaction executed, and performs validation. For that, it has to keep track of the writesets of previously committed transactions and use some timestamping mechanism to determine concurrent transactions. If validation succeeds, the write phase is triggered at all database servers with copies. Concurrency control is distributed: the middleware performs validation and ensures execution within a critical section while the local concurrency control is needed for the generation of local copies.

A distributed OCC strategy is proposed in [3]. It integrates validation into the commit protocol performed for atomicity. Therefore, a transaction can first execute completely locally at one database server and only at commit time communication takes place. This reduces the message overhead considerably.

Key Applications

Although the exact algorithms developed in this early phase of research are barely found in any system, the fundamental techniques behind the coordinated execution are used widely. For example, although few commercial solutions actually provide one-copy-serializability, the concept of ordering conflicting operations to some degree is common. Locking, timestamping, multi-version management, OCC, and distributed and centralized solutions are common place in the management of replicated data.

Experimental Results

Gray et al. [5] show that traditional approaches do not scale well. They analyzed the distributed locking approach and determined that the potential of deadlock increases quickly with the number of replicas in the system, the message overhead becomes too high, and transaction response times become too long. The goal of more recent research into replica and concurrency control has aimed at reducing the overhead by either providing lower levels of correctness or by developing more efficient ways to control the flow of execution in the system.

Cross-references

► ACID Properties
► Concurrency Control – Traditional Approaches
► Distributed Concurrency Control
► One-Copy-Serializabilty
► Replica Control
► Replicated Database Concurrency Control
► Replication based on Group Communication
► Replication for High Availability
► Replication for Scalability
► Transaction Models – the Read/Write Approach
► Two-Phase Locking

Recommended Reading

1. Bernstein P.A. and Goodman N. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. Database Syst., 9(4):596–615, 1984.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA, USA, 1987.
3. Carey M.J. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16(4):703–746, 1991.
4. El Abbadi A. and Toueg S. Availability in partitioned replicated databases. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 240–251.
5. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
6. Kindberg T., Coulouris G.F., and Dollimore J. Distributed Systems: Concepts and Design, 4th edn. Addison Wesley, Reading, MA, USA, 2005.
7. Kung H.T. and Robinson J.T. On optimistic methods for concurrency control. ACM Trans. Database Syst., 6(2):213–226, 1981.

Traditional Data Replication

► Traditional Concurrency Control for Replicated Databases

T

Traditional Replica and Concurrency Control Strategies

► Traditional Concurrency Control for Replicated Databases

Trajectory

Ralf Hartmut Güting
University of Hagen, Hagen, Germany

Synonyms
Trajectory; Moving point

Definition
Representation of a time dependent position observed over some period of time. Usually represented as a polyline in a 3D (2D + time) space for an object moving in the 2D plane.

Key Points
Trajectories describe complete histories of movement; they are stored in *moving objects databases*, sometimes called *trajectory databases* in the literature.

When operations are included, a trajectory corresponds to a value of a *moving point* data type. Queries on databases containing trajectories can be formulated using *moving object languages*. When uncertainty about an object's precise position is taken into account, an uncertain trajectory [3] results which can be viewed as a kind of cylindrical volume in the 2D + time space. There exists a lot of work on indexing trajectories [2]. Querying for trajectories similar to a given one is also an important research area [1].

Cross-references
▶ Moving Objects Databases and Tracking
▶ Spatio-Temporal Data Types

References
1. Chen L., Özsu M.T., and Oria V. Robust and fast similarity search for moving object trajectories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 491–502.
2. Mokbel M.F., Ghanem T.M., and Aref W.G. Spatio-temporal access methods. Bull. TC Data Eng., 26(2):40–49, 2003.
3. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain S. Managing uncertainty in moving objects databases. ACM Trans. Database Syst., 29(3):463–507, 2004.

Trajectory Databases

▶ Moving Objects Databases and Tracking

Trajectory Indexing

▶ Indexing Historical Spatio-Temporal Data

Transaction

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms
Transaction; ACID transaction

Definition
A transaction is a tool for application programmers to delegate the responsibility for preventing damage to data from threats such as concurrent execution, partial execution, or system crashes to the database system software; at the same time, application programmers retain the obligation to think about the impact on data consistency of the code they are writing, when executed alone and without failures. From a programmer's perspective, the power of the transaction paradigm hence lies in the fact that it reduces the task of concurrent failure-aware programming of the entire system to that of correct sequential programming of each application program separately. The transaction concept offers the ACID properties (short for *atomicity, consistency preservation, isolation,* and *durability*) and materializes through concurrency control and recovery. It is nowadays used beyond database systems.

Key Points
Database *transactions* go back to the work of Gray et al. [3,4] in the mid-1970s. Their development has been driven by applications where programs run against data stored in a single database system or a collection of such systems. There are many threats to the overall dependability of a system formed as a combination of databases and application programs; database transactions deal with the threats from concurrent execution, from incomplete execution (e.g., due to crashes or cancelations), and from system crashes that lose information from volatile buffers that has not yet been saved. The problem itself has not only been recognized in the context of database applications [1], yet has finally been solved by the notion of a transaction.

The key point of a transaction is that it comes with system-guaranteed properties collectively known as the *ACID properties* which considerably simplify the development of OLTP applications in that application programs can safely ignore a major portion of the system complexity. In particular, application programs are completely freed up from taking care of the issues of *concurrency*, i.e., effects that may result from concurrent or even parallel program executions and especially data accesses, and of *failures*, i.e., effects that would result from program executions being interrupted at random points due to process or computer failures.

In order to make this work, database systems offer the transaction concept as well as *transaction management*, commonly broken down into *concurrency control* for the synchronization of concurrent access to common data objects from multiple transactions, as well as into *recovery* for being able to restore a consistent state of the database after a crash. From a conceptual point of view, transactions can be modeled in various ways, which essentially boil down to the *page model* as well as the *object model* [5], and they have evolved from an abstraction concept into a system mechanism nowadays offered beyond database systems [2]. The page model considers transactions and transaction management at the syntactic level of disk pages that can either be read or written, while the object model considers them at a level where the semantics of operations on database objects can be taken into account.

Transactions are executed in interleavings called *schedules* or *histories*, which need to satisfy a correctness criterion that commonly comes in a form of *serializability*. Serializability is based on the perception that serial executions are correct and hence tries to make a non-serial execution "look" as if it was run serially. Similar approaches can be applied to both page-model as well as object-model transactions.

Cross-references

Recommended Reading

1. Davies C.T. Data processing spheres of control. IBM Syst. J., 17:179–198, 1978.
2. Elmagarmid A.K. Database Transaction Models for Advanced Applications. Morgan Kaufmann, San Francisco, CA, 1992.
3. Eswaran K.P., Gray J., Lorie R.A., and Traigerv I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 976.
4. Gray J., Lorie R.A., Putzolu G.R., and Traiger I.L. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975, pp. 428–451.
5. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Chopping

DENNIS SHASHA
New York University, New York, NY, USA

Definition

Transaction chopping is a technique for improving the concurrent performance of a database system by reducing the time locks are held. The idea is to break up each transaction into smaller "pieces," such that each piece executes as a transaction, but the effect is as if the original transactions executed serializably.

Key Points

Imagine an application that locks the entire database and then accesses a few rows in a table that does not fit into memory. If one looked at the resource statistics, one would find low CPU consumption and low disk utilization, yet the throughput would be very bad. For example, if ten pages were accessed even at 1 ms per page, then throughput would be only 100 transactions per second. The point is that it is possible to slow down performance greatly just because of a poor locking strategy, even if there are plenty of resources.

One might consider a less extreme case: a transaction that processes an order. The transaction might check whether there is sufficient cash available, then add the appropriate quantities to inventory, subtract the value from cash, and commit the transaction. Because any application that invokes this transaction will access the "cash" data item, that data item may become a bottleneck. Transaction chopping is a technique for circumventing such bottlenecks by dividing

transactions into smaller transactional pieces that will hold locks for only a short time, yet still preserve the serializability of the original transactions.

Assumptions

Transaction chopping makes the following main assumptions: (i) One can characterize all the transactions that will run in some time interval. The characterization may be parameterized. For example, one may know that some transactions update account balances and branch balances, whereas others check account balances. However, one need not know exactly which accounts or branches will be updated. (ii) The goal is to achieve the guarantees of serializability, while obtaining as much concurrency as possible. That is, one would like either to use degree 2 isolation, snapshot isolation, or to chop transactions into smaller pieces. The guarantee should be that the resulting execution be equivalent to one in which each original transaction executes serializably. (iii) If a transaction makes one or more calls to rollback, one knows when these occur and can arrange the transaction to move them towards the beginning.

Basic Definitions

A *chopping* partitions each T_i into *pieces* $c_{i_1}, c_{i_2}, ... c_{i_k}$. Every database access performed by T_i is in exactly one piece. A chopping of a transaction T is said to be *rollback-safe* if either T has no rollback statements or all the rollback statements of T are in its first piece. The first piece must have the property that all its statements execute before any other statements of T. This will prevent a transaction from half-committing and then rolling back. All transactions should be *rollback-safe*. Each piece will act like a transaction in the sense that each piece will acquire locks according to some standard method that guarantees the serializability of the piece.

For example, suppose T updates an account balance and then updates a branch balance. Each update might become a separate piece, acting as a separate transaction.

Correct Choppings

One may characterize the correctness of a chopping with the aid of an undirected graph having two kinds of edges. (i) *C edges*: C stands for *conflict*. Two pieces p and p' from different original transactions conflict if there is some data item x that both access and at least

one modifies. In that case, one draws an edge between p and p' and label the edge C. (ii) *S edges*: S stands for *sibling*. Two pieces p and p' are siblings if they come from the same transaction T. In this case, draw an edge between p and p' and label the edge S.

The resulting graph is called the *chopping graph*. (Note that no edge can have both an S and a C label.) A chopping graph has an *SC-cycle* if it contains a simple cycle that includes at least one S edge and at least one C edge. A chopping of $T_1, T_2, ..., T_n$ is *correct* if any execution of the chopping is equivalent to some serial execution of the original transactions. "Equivalent" is in the sense of the serializability entry.

Theorem 1: *A chopping is correct if it is rollback-safe and its chopping graph contains no SC-cycle.*

Theorem 1 shows that the goal of any chopping of a set of transactions should be to obtain a rollback-safe chopping without an SC-cycle.

Conclusion

Transaction chopping is a method to enhance concurrency by shortening the time a bottleneck resource is held locked. It works well in practice and the theory extends naturally to snapshot isolation as well as read committed isolation levels.

Cross-references

► Concurrency Control
► Locking
► Transaction Execution

Recommended Reading

1. Fekete A., Liarokapis D., O'Neil E., O'Neil P., and Shasha D. Making Snapshot Isolation Serializable. ACM Trans. Database Syst., 30(2):492–528, 2005.
2. Shasha D. and Bonnet P. Database Tuning : Principles Experiments and Troubleshooting Techniques. Morgan Kaufmann, 2002.

Transaction Commit Time

► Transaction Time

Transaction Execution

► Concurrency Control – Traditional Approaches

Transaction Management

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

Concurrency control and recovery; Transaction scheduling; Transaction processing

Definition

Transaction management [2,6] refers to the tasks of processing multiple transactions issued by various clients of a database server in such a way that the ACID contract can be fulfilled, that is, the properties of *atomicity, consistency preservation, isolation,* and *durability* of each individual transaction can be guaranteed. Transaction management is generally understood as requiring serializability-based concurrency control as well as recovery from failures. Concurrency control is the task of scheduling transactions such that their serializability can be guaranteed, while recovery has to restore a consistent database state after a system or media failure. Assuming that the database server is in charge of the "C," the former guarantees the "I" in ACID, the latter the "A" and "D" properties. Transaction management has to be highly efficient, as modern transaction servers need to accommodate thousands of transactions per minute. This is achieved by a comprehensive combination and interplay of theoretical research and practical developments.

Historical Background

Transaction management emerged in the 1970s in early database management systems [5], and has become an indispensable part of a database server [1,3]. Its goal is to devise efficient algorithms for handling transactions, the essential "contract" between an application program and transactional server that combines a number of requests to the server into a logical unit. Over the years, transaction management has received theoretical underpinnings as well as system implementations in various ways, the former of which have allowed it to extend beyond simple database objects (pages) [7]. Indeed, transaction management can essentially be "positioned" at any level of abstraction within the functional layers of a data server, where the price to pay for expressiveness typically is efficiency. Nevertheless, transaction management has also been extended into areas beyond database management, among them operating systems and, more recently, programming languages.

Foundations

To recognize the essence of what transaction management is about, consider the operation of a bank that uses a relational database to keep track of its account business. The database may contain a table *Accounts* which describes bank accounts in terms of their account id, associated customer name, identification of the respective bank branch, balance, and possibly other data. Transactions in the bank are either *withdrawals* or *deposits* (debit/credit transactions) applied to *Accounts*, and these operations are often combined into *funds transfers*, i.e., withdrawals from one account and immediate deposit into another. With a huge number of clients potentially issuing simultaneous requests to the bank's database server, a *concurrent* execution of multiple debit/credit transactions is mandatory in order to exploit the server's hardware resources and to achieve processing speeds acceptable to clients. Concurrently executing transactions are typically modifying the underlying database of the banking application (table *Accounts*) frequently. In order to be able to ignore the potential fallacies of this concurrency, each transaction would ideally be executed as if there were no other transactions, but that would mean no concurrency. This tradeoff between concurrency for the sake of performance on the one hand, and potential sequential execution for the sake of correctness on the other, is one aspect of transaction processing, formally captured through notions of *serializability* and reconciled by the *concurrency control techniques* of a transactional server.

Figure 1 illustrates that concurrency may indeed be tricky, since it may have a disastrous impact on the consistency of the underlying data. Consider two debit/credit transactions t_1 and t_2 that are concurrently executed and that are both operating on the same account x (which could be the account id). To distinguish the two different instances of the local variable "balance" that temporarily holds the value of the account balance, they can be referred to as "balance1" for transaction t_1 and "balance2" for t_2. The first transaction then intends to withdraw $30, the second transaction intends to deposit $20, and it is assumed that the initial account balance is $100. The table in Fig. 1 shows those

Transaction t_1	Time	Transaction t_2
/* balance1 = 0, x.Account_balance = 100, balance2 = 0 */		
Select account_balance Int:balance1 from account Where account_Id = x	1	
/* balance1 = 100, x.Account_balance = 100, balance2 = 0 */		
	2	Select account_balance Int:balance2 from account Where account_Id = x
/* balance1 = 100, x.Account_balance = 100, balance2 = 100 */		
balance1 = balance1 - 30	3	
/* balance1 = 70, x.Account_balance = 100, balance2 = 100 */		
	4	balance2 = balance2 + 20
/* balance1 = 70, x.Account_balance = 100, balance2 = 120 */		
Update account Set account_balance = :balance Where account_Id = x	5	
/* balance1 = 70, x.Account_balance = 70, balance2 = 120 */		
	6	Update account Set account_balance = :balance2 Where account_Id = x
/* balance1 = 70, x.Account_balance = 120, balance2 = 120 */		

Transaction Management. Figure 1. Concurrent transactions requiring concurrency control.

parts of the two transactions that read and modify the account record. Upon completion of the execution, the balance of account *x,* as recorded in the persistent database, will be $120, although it should be $90 after execution of the two transactions. Thus, the recorded data are incorrect, a kind of "anomaly" must be prevented, and concurrent executions must be treated with care. Similar anomalies can arise from transaction failures.

A second fundamentally important point is that the various accesses a transaction has to perform need to occur *in conjunction*: Once a transaction has begun execution, its data accesses should look to the outside world as an atomic operation which is either executed completely or not at all. This property of atomicity should be guaranteed even in a failure-prone environment where individual transactions or the entire database server may fail at an arbitrary point in time. To this end, a transactional server needs to provide *recovery techniques* to cope with failures. In addition to ensuring transaction atomicity, these techniques also serve to ensure the *durability* of a transaction's effects once the transaction is completed. The scenario shown in Fig. 2 illustrates this. It shows a program which transfers a given amount of money between two accounts, by first withdrawing it from a source account and then depositing it in a target account.

The program is described in terms of SQL statements embedded into a C program. It is assumed that the funds transfer program has started executing and has already performed the withdraw operation (i.e., the first SQL Update). If there is a hardware or software failure that interrupts the program's execution at this point, the remaining second update operation will not be performed. Thus, the target account will not receive the money.

A recovery procedure, to be invoked after the system is restarted, will try to find out which updates were already made by ongoing transaction program executions and which ones were not yet done, and will try to fix the situation. However, implementing recovery procedures on a per-application case basis is a difficult task that is itself error prone because of its sheer complexity, especially because multiple transactions issued by different programs may have accessed the data at the time of the failure. So rather than programming recovery in an ad hoc manner for each application separately, a systematic approach is needed, as described, for example, in [9,2]. System-provided recovery ensures the atomicity of transactions and simplifies the understanding of the post-failure state of the data and the overall failure handling on the application side. In the sample scenario of Fig. 2, rather than being left with the inconsistent state in the middle

```
/* funds transfer program */
void main()   {
        EXEC SQL BEGIN DECLARE SECTION;
            int sourceid, targetid, amount; /* input variables */
        EXEC SQL END DECLARE SECTION;

/* read user input */
        printf("Enter Source ID, Target ID, Amount to be transfered:");
        scanf("%d %d %d", &sourceid, &targetid, &amount);

/* subtract desired amount from source */
        EXEC SQL Update Accounts
                Set Account_Balance = Account_Balance - :amount
                Where Account_Id = :sourceid;

/* add desired amount to target */
        EXEC SQL Update Accounts
                Set Account_Balance = Account_Balance + :amount
                Where Account_Id = :targetid;

        EXEC SQL Commit Work; }
```

Transaction Management. Figure 2. Sample funds transfer that needs atomicity.

of the transaction, the system recovery will restore the state as of before the transaction began. On the other hand, if the transaction had already issued its "commit transaction" call, then the system would guarantee the durability of the transaction's complete funds transfer.
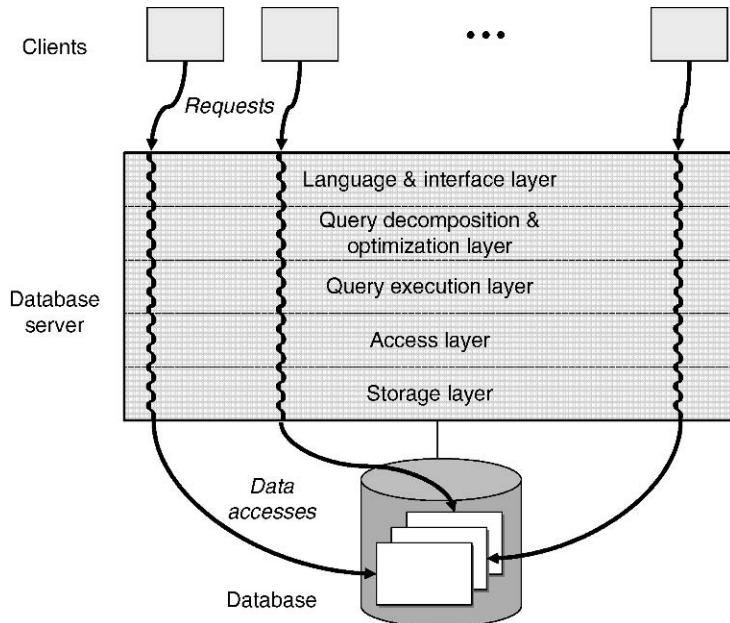
Guaranteeing the ACID properties of a transaction, which allow application developers to disregard concurrency and failures, are the major goal of transaction management; the means to accomplish this are concurrency control and recovery. These cornerstones for building highly dependable information systems can also be successfully applied outside the scope of *online transaction processing* (OLTP) applications.

Key Applications

Figure 3 shows the layered architecture shared by essentially all database servers in one form or another. When a client request arrives at the server, the server executes code that transforms the request into one or more operations at each of the underlying layers, ultimately arriving at a sequence of disk accesses (unless caching avoids the disk access). The *language and interface layer* makes various kinds of interfaces available to the application developer, usually in the form of APIs (e.g., SQL and ODBC). The *query decomposition and optimization layer* works on an internal, tree based representation of a request and is concerned with the further decomposition of the request into smaller units

that can be directly executed; this is the level where also optimizations are carried out. The execution plan chosen is often represented as an operator tree where each operator can be directly mapped to a piece of server code. This code is provided by the *query execution layer*. Next, index structures and also the capabilities for accessing and manipulating data records are provided by the *access layer*. Finally, the *storage layer* is responsible for managing the pages of a database, including disk I/O and caching.

The layered architecture in Fig. 3 does not explicitly mention transaction management, for the simple reason that the functionality of transactional concurrency control and recovery can be tied to any of the five layers shown. For many applications that run simple transactions, each of which accesses a small portion of the data only, transaction management is commonly integrated into the access and storage layers. Applications in this category include classical OLTP scenarios such as banking (see above) or reservation systems, where high availability, high throughput, and high reliability are of utmost importance. However, transactions in electronic commerce applications, where client requests may span multiple databases as well as other information sources across enterprise boundaries, or transactions arising from an execution of business processes which typically take some form of semantic information (stemming from higher-level, for example SQL-type operations) into account can also benefit

Transaction Management. Figure 3. Functional database system layers.

from transaction concepts. Through appropriate transaction models and an appropriate adaptation of the relevant concurrency control and recovery algorithms to these models, transaction management can be applied in almost any area that needs to handle concurrent requests to shared resources in such a way that the ACID properties can be guaranteed [9].

Future Directions

With constant changes in network-centric computing, including the proliferation of Web services, long-running processes across organizational boundaries, large scale peer-to-peer publish-subscribe and collaboration platforms, and ambient-intelligence environments with large numbers of mobile and embedded devices, support for handling or even masking concurrency and component failures is critical in many modern application areas, but can no longer use traditional atomicity concepts alone. In open systems applications are constructed from pre-existing components, and these components and their configurations are not known in advance and they can change on the fly. Thus, it is crucial that atomicity properties of components become composable and allow for reasoning about the behavior of the resulting system. Transaction management will thus continue to require research for the foreseeable future.

Experimental Results

Since transaction management typically has to meet high efficiency requirements, experimental evaluation and comparison of algorithmic approaches to concurrency control and recovery has always been crucial, as is tuning of the transaction management component of a database server for the system administrator. Prominent references including implementation recipes as well as experimental evaluations are [4,8].

Cross-references

▶ Control
▶ Crash Recovery
▶ Database Server
▶ Performance Analysis of Transaction Processing Systems
▶ Transaction
▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
2. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, San Francisco, CA, 1997.
3. Cellary W., Gelenbe E., and Morzy T. Concurrency Control n Distributed Database Systems. North–Holland, Amsterdam, 1988.

4. Gray J. (ed.). The Benchmark Handbook for Database and Transaction Processing Systems. 2 edn. Morgan Kaufmann, San Francisco, CA, 1993.

5. Gray J., Lorie R.A., Putzolu G.R., and Traiger I.L. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Data Bases, 1975, pp. 428–451.

6. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.

7. Lynch N., Merritt M., Weihl W., and Fekete A. Atomic Transactions. Morgan Kaufmann, San Francisco, CA, 1994.

8. Shasha D. and Bonnet Ph. Database Tuning: Principles Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.

9. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Management in Distributed Database Systems

► Distributed Transaction Management

Transaction Manager

ANDREAS REUTER[1,2]
[1]EML Research gGmbH, Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslautern, Germany

Definition

The transaction manager (TxM) is a special resource manager that implements the resource type "transaction." It handles all the state transitions that a transaction can perform. For simple ACID transactions these are: begin, savepoint, rollback, abort, prepare commit; for more refined transaction models there will be additional states. The resource managers register with the TxM when they get employed by a transaction for the first time. Thus, the TxM keeps a record for each transaction of which resource managers have been involved with it. In the same vein, it stores which sessions each transaction has used in case it has performed operations on other nodes in a distributed system. The TxM closely interacts with the concurrency control manager who can make transaction wait for other transactions in case of access conflicts. It also interacts with the logging and recovery subsystem and the communications manager.

Key Points

The TxM keeps track of all active transactions. When a transaction starts, it is assigned a transaction ID that is unique across any distributed system and will never repeat. For each transaction the TxM stores which resource managers and which sessions are associated with it; for sessions the polarity (outgoing/incoming) is also important. For each state transition of a transaction, the TxM will orchestrate the proper protocol. The most important of these protocols is the two-phase commit (2PC) protocol. Depending on the structure of a transaction, the TxM will act as a coordinator or a participant – or both. Using the information about the participating resource managers and communication sessions, the TxM calls them at their proper callback entries to perform all the necessary actions to go, for example, into the prepared state. TxMs are often implemented as interpreters of the formal description of a state machine, so they can dynamically switch between different (optimizations of) commit protocols.

Upon recovery, the TxM is the first resource manager that is reconstructed by the recovery manager. When the recovery manager has completed its initial backward scan of the online log, the table of active transactions is completely recovered. If there are any in-doubt transactions left, the TxM starts handling that particular part of the 2PC in order to resolve those transactions.

Cross-references
► Communications Manager
► Concurrency Control Manager
► Logging/Recovery Subsystem
► Storage Resource Management
► Transactional Middleware

Recommended Reading
1. Gray J. and Reuter A. Transaction Processing – Concepts and Techniques; Morgan Kaufmann, San Mateo, CA, 1993.

Transaction Model

► Transaction Models – The Read/Write Approach

Transaction Models – the Read/Write Approach

GOTTFRIED VOSSEN
University of Münster, Münster, Germany

Synonyms

Transaction model; Page model; Read/write model

Definition

The transaction concept essentially establishes an "ACID contract" in data-processing situations, and a transaction model is an abstraction concept that makes this concept amenable to realizations. Two fundamental models are the *page model* as well as the *object model*, where the former is an execution model and the latter is more a conceptual model. The page model is based on the perception that database operations ultimately are read or write operations on pages that need to be transferred between secondary storage and main memory or the database buffer. The model allows making all relevant notions (in particular interleavings of multiple transactions and schedule correctness) precise in a syntactic manner, and it forms the basis for verifying a variety of concurrency control algorithms that can be used in database management as well as other systems.

Historical Background

The read/write or page model of transactions goes back to the work of Jim Gray [5,6] and Kapali Eswaran et al. [4]. Related notions of atomic actions have been discussed by others around the same time, the mid- to late-1970s, including [8,9]. The page-model transaction concept became the subject of intensive theoretical studies, in particular in the work of Christos Papadimitriou and, independently, Phil Bernstein et al. around 1980 [2,3,10,11]. It is essentially valid in its original form until today, yet has received a number of extensions and variations over the years [13].

Foundations

The read/write model of database transactions is motivated by the observation that all operations on data (i.e., queries as well as updates) are eventually mapped into indivisible read and write operations on disk pages or on a page cache in main memory. Thus, to study the effects of concurrent executions, it essentially suffices to inspect the interleavings of the resulting page operations. The abstraction from higher-level data operations, such as SQL commands or method invocations on business objects, down to the view that a resulting transaction consists of reads and writes *only* is a strong one, yet suffices for many practical purposes. In fact, a comprehensive theory of concurrency control and of recovery can be built on it, which is directly applicable to real-world systems, albeit with some performance limitations.

Definition of a Transaction

To define the model of read/write transactions formally, a database is assumed to contain a (finite) set $D = \{x, y, z, \ldots\}$ of (indivisible and disjoint) items (*pages*) with indivisible read and write operations. Data items are often denoted by small letters from the end of the alphabet, where indices are used if necessary (e.g., x_1, y_4). A transaction can then be defined as a total or a partial order of steps; for total orders, a formal definition is as follows: A *transaction t* is a (finite) sequence of steps (actions) of the form $r(x)$ or $w(x)$, written $t = p_1 \ldots p_n$, where $n < \infty$, $p_i \in \{r(x), w(x)\}$ for $1 \leq i \leq n$, and $x \in D$ for a given database D. Here, r stands for "read" and w for "write." Thus, a transaction abstracts from the details of a program execution and instead focuses on the sequence of read and write operations that results from that execution.

Each step occurring in a transaction can be uniquely identified so that two distinct transactions do not have steps in common. Let p_j denote the jth step of a given transaction, and let p_{ij} denote the jth step of transaction i in the presence of multiple transactions. Then the following terminology is common: If $p_j = r(x)$, the interpretation is that step j reads data item x; if $p_j = w(x)$, the interpretation is that step j writes data item x. A transaction is hence a purely syntactic entity whose semantics or interpretation is unknown. In the absence of information on the semantics of the program that launches a transaction, however, the best that can be done is to devise a *Herbrand semantics*, i.e., a "syntactic" interpretation of the steps of a transaction that is as general as possible: (i) In case $p_j = r(x)$, the current value of x is assigned to a local variable v_j. (ii) In case $p_j = w(x)$, a possibly new value, computed by the respective program, is written into x. Each value written by a transaction t potentially depends on the values of *all* data items that t has previously read; if t writes x,

x is the return value of an arbitrary but unknown function f_j applied to all values read before the respective write step. The Herbrand semantics of a read/write transaction thus is a technical vehicle, only useful for making various correctness notion for transaction interleavings precise.

The *total ordering* requirement to the steps of a transaction can be relaxed into a *partial ordering*; thus, a *transaction t* becomes a partial order of steps (actions) of the form $r(x)$ or $w(x)$, where $x \in D$ and reads and writes as well as multiple writes applied to the same data item are ordered. More formally, a *transaction* is a pair $t = (\text{op}, <)$, where op is a finite set of steps of the form $r(x)$ or $w(x)$, $x \in D$, and $< \subseteq$ op $\times$ op is a partial order on set op for which the following holds: If $\{p, q\} \subseteq$ op s.t. p and q both access the same data item and at least one of them is a write step, then $p < q \vee q < p$. In other words, in the partial ordering of a transaction's steps, a read and write operation on the same data item or two write operations on the same data item need to be ordered. With "conflicting" steps inside a transaction being ordered, the "semantics" outlined above for totally ordered transactions carries over to partially ordered ones. For simplicity, however, most discussions of correctness criteria in the literature stick to total orders. Although read/write transactions are considered syntactic entities only, it is an advantage of this model that its theory can be developed in the absence of semantic information and hence can be used for *every possible interpretation* of the transactions. In other words, the read/write page model is fairly general despite its simple structure.

The model as described above allows a transaction to read or write the same data item more than once, as it is the case in the example $t = r(x)w(x)r(y)r(x)w(x)$. Here t reads and writes x twice, although it is reasonable to assume that the value of x remains available, after having been read the first time, in the local variables of the underlying program for as long as it is needed by t, and that only the last write step determines the final value of x produced by this transaction. To exclude "redundancies" of this kind, it is common to assume that (i) in each transaction each data item is read or written at most once, and (ii) no data item is read (again) after it has been written. The latter condition does not exclude the possibility of "blind writes," which is a write step on a data item that is not preceded by a read of that data item.

Schedules and Histories

The distinction between partial and total orderings is also appropriate for interleavings of transactions, i.e., for schedules and histories: Let $T = \{t_1,...,t_n\}$ be a (finite) set of transactions, where each $t_i \in T$ has the form $t_i = (\text{op}_i, <_i)$, with op_i denoting the set of operations of t_i and $<_i$ denoting their ordering, $1 \le i \le n$. A *history* for T is a pair $s = (\text{op}(s), <_s)$ s.t. (i) $\text{op}(s) \subseteq \bigcup_{i=1}^{n} \text{op}_i \cup \bigcup_{i=1}^{n} \{a_i, c_i\}$ and $\bigcup_{i=1}^{n} op_i \subseteq op(s)$, i.e., s consists of the union of the operations from the given transactions plus a termination operation, which is either a c_i (commit) or an a_i (abort), for each $t_i \in T$, (ii) $(\forall i, 1 \le i \le n)\ c_i \in \text{op}(s) \Leftrightarrow a_i \notin \text{op}(s)$, i.e., for each transaction, there is either a commit or an abort in s, but not both, (iii) $\bigcup_{i=1}^{n} <_i \subseteq <_s$, i.e., all transaction orders are contained in the partial order given by s, (iv) $(\forall i, 1 \le i \le n)(\forall p \in \text{op}_i) p <_s a_i$ or $p <_s c_i$, i.e., the commit or abort operation always appears as the last step of a transaction, and (v) any pair of operations $p,q \in \text{op}(s)$ from distinct transactions accessing the same data item s.t. at least one is a write operation is ordered in s in such a way that either $p <_s q$ or $q <_s p$. A *schedule* is a prefix of a history. A history s is *serial* if for any two transactions t_i and t_j in it, where $i \ne j$, all operations from t_i are ordered in s before all operations from t_j or vice versa. Thus, a history (for partially ordered transactions) has to contain all operations from all transactions (i), needs a distinct termination operation for every transaction (ii), preserves all orders within the transactions (iii), has the termination steps as final steps in each transaction (iv), and orders "conflicting" operations (v). The view that two operations which access the same data item and of which at least one is a write operation are in conflict is identical to the notion of conflict that will shortly be used as the basis for a notion of serializability. The notions "schedule" and "history" are not always used in the sense defined here in the literature, but are often used as synonyms.

Schedule Correctness

For transaction executions, it is important to not only have a model of interleavings and their constituents, but also to fix a notion of schedule or history *correctness*. To this end, *conflict serializability* (CSR) is the notion which is most important for the practice of transactional information systems, in particular for designing scheduling algorithms and for building schedulers. CSR is computationally easy to test, and it has a number of interesting theoretical properties which

can justify an exploitation of this concept in practice in a variety of ways. Finally, it can be generalized to other transaction models and different data settings. Conflict serializability is based on a simple notion of conflict which was mentioned already in connection with partially ordered histories, and which is appropriate for the syntactical nature of read/write transactions. Two data operations from distinct transactions are *in conflict* in a schedule if they access the same data item and at least one of them is a write. The notion of conflict gives rise to a notion of serializability: A history is *conflict serializable* if there exists a serial history with the same conflicts.

Commutativity of Operations

Conflict serializability can be characterized via acyclic conflict graphs, which gives rise to efficient testability. It is can also be characterized via *commutativity rules* for page model data operations. In these rules "$\sim$" means that the ordered pair of actions on the left-hand side can be replaced by the right-hand side, and vice versa: (C1) $r_i(x)r_j(y) \sim r_j(y)r_i(x)$ if $i \neq j$, i.e., two read steps $r_i(x)$ and $r_j(y)$, $i \neq j$, which occur in a schedule in this order and are adjacent (with no other operation in between), may be commuted. (C2) $r_i(x)w_j(y) \sim w_j(y)r_i(x)$ if $i \neq j$, $x \neq y$, i.e., a read and write step can be exchanged if they are from distinct transactions and access different data items. (C3) $w_i(x)$ $w_j(y) \sim w_j(y)w_i(x)$ if $i \neq j$, $x \neq y$, i.e., two write steps can be commuted if they refer to different data items. These commutativity rules can be applied to a given schedule or history in a stepwise fashion, for example to transform $s = w_1(x)r_2(x)w_1(y)w_1(z)r_3(z)w_2(y)w_3(y)$ $w_3(z)$ into $w_1(x)w_1(y)w_1(z)r_2(x)w_2(y)r_3(z)w_3(y)w_3(z)$, thereby proving it equivalent to the serial history $t_1t_2t_3$. The above transformations have implicitly assumed that operations in a schedule are totally ordered. With partial orders, one may need an additional transformation rule which simply states that two unordered operations can be arbitrarily ordered if they are non-conflicting.

The commutativity rules can be used for introducing another relation on schedules: Let s and s' be two schedules s.t. $op(s) = op(s')$. Define $s \sim s'$ if s' can be obtained from s by a single application of the commutativity rules to the steps of the schedule. Let "$\overset{*}{\sim}$" denote the reflexive and transitive closure of "$\sim$," i.e., $s \overset{*}{\sim} s'$ if s' can be obtained from s by a finite number of

applications of the rules. It turns out that "$\overset{*}{\sim}$" is an equivalence relation on the set of all schedules for a given set of transactions, and that finitely many applications of the rules may transform a given history into a serial one, as in the example above. More formally, a history s is *commutativity based reducible* if there is a serial history s' s.t. $s \overset{*}{\sim} s'$, i.e., s can be transformed into s' through a finite number of allowed transformation steps according to the commutativity rules. Therefore, a history s is commutativity based reducible iff s is CSR. An important application of this alternative characterization of schedule correctness is that it can immediately be generalized. Indeed, it is irrelevant to know of a schedule whether a step reads or writes or which data item it accesses, as long as it is known which steps of the schedule are in conflict. The latter suffices for deciding about the correctness of the schedules, even without any knowledge about the meaning of the operations [11]. Thus, the steps of a schedule or history to which a commutativity argument is applied may be of a completely different type, such as *increment* and *decrement* on a counter object, *push* and *pop* on a stack object, or *enqueue* and *dequeue* on a queue object. This observation can be exploited in the context of extended transaction models, where notions of serializability can be established that are based on semantic information.

The page model of transactions can also be extended to accommodate transaction recovery. Indeed, when a transaction aborts, the intuitive reaction of the underlying system is to "undo" the effects of that transaction, which can adequately be captured by inverse write operations. To execute an abort operation, the system would have to execute inverse writes in reverse order of their occurrence [1].

Key Applications

Key applications for transactions started out from debit/credit scenarios in banks and financial institutions, where multiple customer activities against shared accounts need to be synchronized, yet executed at a high throughput rate. In a debit/credit transaction, the activities are either *withdrawals* or *deposits*, and these are often combined into *funds transfers*. The typical structure of a debit/credit program is shown below, using SQL commands embedded into a C program. Note the distinction between local variables of

the invoked program and the data in the underlying database that is shared by all programs.

```
/* debit/credit program */
void main()
{EXEC SQL BEGIN DECLARE SECTION;
 int accountid, amount;
  /* input variables */
 int balance; /* intermediate variable */
 EXEC SQL END DECLARE SECTION;
 /* read user input */
 printf("Enter Account ID, Amount for
 deposit (positive) or withdrawal (nega-
 tive): ");
 scanf("%d %d", &accountid, &amount);
 /* determine current balance of the ac-
 count, reading it into a local variable
 of the program */
 EXEC SQL
 Select Account_Balance Into :balance
 From Accounts
 Where Account_Id = :accountid;
 /* add amount (negative for withdrawal) */
 balance = balance + amount;
 /* update account balance in the database */
 EXEC SQL Update Accounts
  Set Account_Balance = balance
  Where Account_Id = :accountid;
 EXEC SQL Commit Work; }
```

The crucial situation is that, say, two transactions access the same account, where one makes a withdrawal and the other a deposit, i.e., both transactions write new values of the account balance. If unsynchronized, these transaction could overwrite each other's results, thereby leaving the account in an inconsistent state. Another crucial situation is that a transaction makes a withdrawal from one account and a deposit into another (i.e., a transfer); if this transaction crashes in the middle, i.e., after the withdrawal but before the deposit, money would be lost if the system is not capable of restoring the state prior to the withdrawal. The read/write model allows for a straightforward formalization of these situations, and efficient concurrency control as well as recovery procedures can be devised to handle them.

For scenarios like the one just sketched (in particular transfers that need to appear atomic to the outside world), there are numerous applications in today's information systems landscape of electronic business over the Internet, where client requests are rarely restricted to single data servers, but often span multiple databases and other information sources across enterprise boundaries, yet the mutual consistency of all this data is crucial and thus important to maintain. Then, the resulting transactions operate in a distributed system that consists of multiple servers, often with heterogeneous software. To go one step further, a *business process* is a set of *activities* (or steps) that belong together in order to achieve a certain business goal. Business processes are also at the heart of advanced, business-to-consumer (B2C) or business-to-business (B2B) services on the Internet, for example, electronic auctions (B2C) or supply chains (B2B). Typical examples would be the processing of a credit request or an insurance claim in a bank or insurance company, respectively, the administrative procedures for real estate purchase, or the "routing" of a patient in a hospital.

Future Directions

Transactions have originally been developed in the database system community, but their usage and potential benefits are by no means limited to database management. Not surprisingly, the transaction concept is also being explored in the operating systems and programming languages communities. Recent trends include, for example, enhancing the Java language with a notion of atomic blocks that can be defined for methods of arbitrary classes. This could largely simplify the management of concurrent threads with shared objects, and potentially also the handling of failures and other exceptions. The run-time environment could be based on an extended form of *software transactional memory*. Another important direction in ongoing research is to design and reason about guarantees that resemble transactions but are weaker than the ACID properties, especially with regard to the notion of isolation. A model that is widely deployed in industrial data management systems is *snapshot isolation* (based on keeping a versioned history of data items).

Experimental Results

The read/write model of transactions has in part been so successful since it allows for highly efficient implementations of concurrency control and recovery methods, and it allows for tuning a system on the fly. Surveys are provided by [7] and in particular [12].

Cross-references

Recommended Reading

1. Alonso G., Vingralek R., Agrawal D., Breitbart Y., El Abbadi A., Schek H.-J., and Weikum G. Unifying concurrency control and recovery of transactions. Inform. Syst., 19:101–115, 1994.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
3. Bernstein P.A., Shipman D.W., and Wong W.S. Formal aspects of serializability in database concurrency control. IEEE Trans. Software Eng., SE-5:203–216, 1979.
4. Eswaran K.P., Gray J., Lorie R.A., and Traiger I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 1976.
5. Gray J. Notes on database operating systems. In: Operating Systems: An Advanced Course, LNCS, Vol. 60, R. Bayer, M.R. Graham, G. Seegmüller (eds.). Springer, Berlin Heidelberg New York, 1978, pp. 393–481.
6. Gray J. The transaction concept: virtues and limitations. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 144–154.
7. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
8. Lampson B.W. Atomic transactions. In: Distributed Systems – Architecture and Implementation: An Advanced Course, LNCS, Vol. 105, B.W. Lampson, M. Paul, H.J. Siegert (eds.). Springer, Berlin Heidelberg New York, 1981.
9. Lomet D.B. Process structuring, synchronization, and recovery using atomic actions. ACM SIGPLAN Notices, 12(3):128–137, 1977.
10. Papadimitriou C.H. The Serializability of concurrent database updates. J. ACM, 26:631–653, 1979.
11. Papadimitriou C.H. The Theory of Database Concurrency Control. Computer Science, Rockville, MD, 1986.
12. Shasha D., Bonnet Ph. Database Tuning – Principles, Experiments, and Troubleshooting Techniques. San Francisco, CA, Morgan Kaufmann, 2003.
13. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Processing

Transaction Scheduling

Transaction Service

Transaction Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Registration time; Extrinsic time; Physical time; Transaction commit time; Belief time

Definition

A database fact is stored in a database at some point in time, and after it is stored, it remains current, or part of the current database state, until it is logically deleted. The *transaction time* of a database fact is the time when the fact is current in the database. As a consequence, the transaction time of a fact is generally not a time instant, but rather has duration.

The transaction time of a fact cannot extend into the future. Also, as it is impossible to change the past, meaning that (past) transaction times cannot be changed.

In the context of a database management system that supports user transactions, the transaction times of facts are consistent with the serialization order of the transactions that inserted or logically deleted them. Transaction times may be implemented using transaction commit times, and are system-generated and -supplied.

Key Points

A database is normally understood to contain statements that can be assigned a truth value, also called facts, that are about the reality modeled by the database and that hold true during some non-empty part of the time domain. Transaction times, like valid times, may be associated with such facts. It may also be noted that it is possible for a database to contain the following different, albeit related, facts: a non-timestamped fact and that fact timestamped with a valid time. The first would belong to a snapshot relation, and the second would

belong to a valid-time relation. Both of these facts may be assigned a transaction timestamp. The resulting facts would then be stored in relations that also support transaction time.

A transaction time database is append-only and thus ever-growing. To remove data from such a database, *temporal vacuuming* may be applied.

The term "transaction time" has the advantage of being almost universally accepted and it has no conflicts with the other important temporal aspect of data, valid time.

The Oracle DBMS explicitly supports transaction time. Applications can access prior transaction-time states of their database, by means of transaction time-slice queries. Database modifications and conventional queries are temporally upward compatible.

Concerning the alternatives, the term "registration time" seems to be straightforward. However, this term may leave the impression that the transaction time is only the time instant when a fact is inserted into the database. "Extrinsic time" is rarely used. "Physical time" is also used infrequently and seems vague. "Transaction commit time" is lengthy, but more importantly, the term appears to indicate that the transaction time associated with a fact must be identical to the time when that fact is committed to the database, which is an unnecessary restriction. The term is also misleading because the transaction time of a fact is not a single time instant as implied. The term "belief time" stems from the view that the current database state represents the current belief about the aspects of reality being captured by the database. This term is used infrequently.

Cross-references
► Supporting Transaction Time Databases
► Temporal Compatibility
► Temporal Database
► Temporal Generalization
► Temporal Specialization
► Temporal Vacuuming
► Time Domain
► Timeslice Operator
► Transaction
► Transaction-Time Indexing
► User-Defined Time
► Valid Time

Recommended Reading
1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Snodgrass R.T. and Ahn I. A taxonomy of time in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 236–246.
3. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, September 1986.

Transactional Business Processes
► Transactional Processes
► Workflow Transactions

Transactional Consistency in a Replicated Database
► One-Copy-Serializability

Transactional Middleware

GUSTAVO ALONSO
ETH Zurich, Zurich, Switzerland

Synonyms
TP monitor; Object monitor; Application server; Transaction manager; Transaction service

Definition
Transactional Middleware is a generic term used to refer to the IT infrastructure that supports the execution of electronic transactions in a distributed setting. The best known form of transactional middleware is Transaction Processing Monitors (TP Monitors or TPM), which have been around for more than 3 decades (e.g., CICS of IBM). Today, TP Monitors are at the heart of most application servers and are a key component of any enterprise computing architecture. The main role of these systems is to run transactions, i.e., to support the illusion that certain distributed operations are executed atomically. This makes the design of complex systems easier for the programmer, who does not need to implement this functionality but can rely on the transactional middleware to ensure that groups of operations are executed in their entirety or not all, with the transactional middleware taking care of all the work necessary to do so. Historically, transactional middleware has often provided much more functionality than just

transactions and, in many ways, they were some of the earliest forms of middleware. They were already used in early mainframes to connect different applications.

Historical Background

The background for transactional middleware is TP Monitors. TP Monitors originated with the first commercial applications running on mainframes and the need to provide additional functionality not provided by the operating system. As the needs of applications and developers flourished, so did the capabilities and features of TP Monitors. This is how TP Monitors became a general term to refer to complex infrastructure and collection of tools that provide not only transactional support but also specialized networking capabilities, batch jobs, job control languages, multithreading, transactional extensions to programming languages, load-balancing capabilities, transactional message queues, etc. When enterprise applications broke away from the mainframe and started running on mixed environments (e.g., a large database in the mainframe, workstations for processing, and PCs for the clients), TP Monitors proved to be the ideal platform to develop and operate such distributed systems. During the 1990s, TP Monitors dominated the middleware arena and were not only the most prominent but also the clearly dominant form of middleware in the market.

The conventional TP Monitors with extensive functionality were called TP-Heavy. These were standalone products were intended for the development of distributed, transactional applications with high performance requirements. A simpler form of TP Monitor was the so called TP-Light systems. Typically, these were extensions to database engines to support interactions with the database through RPC calls and the addition of some basic data processing capabilities to the database engine without having to resort to a full middleware layer such as that required by TP-Heavy approaches. The dominance and importance of TP-Monitors is illustrated by the struggles of CORBA systems to implement the Object Transactional Services described in the CORBA specification. Object Transactional Services are exactly the transactional functionality provided by a TP-Monitor and in many commercial systems it was implemented by using an already existing TP Monitor. Such combined systems enjoyed a short period of popularity under the name of Object Monitors or Object Transaction Monitors.

TP Monitors today are still widely used although the name TP Monitor has been replaced by others, more all-encompassing terms. For instance, applications servers have a TP Monitor as one of the central components. Platforms like .NET o J2EE also rely heavily on TP Monitor functionality. Hence, today these systems are collectively referred to as transactional middleware.

Foundations

To understand transactional middleware, one needs to understand that it is not just about transactions (or, at least, historically it was not just about transactions). Nevertheless, the core functionality of transactional middleware is to run distributed transactions in an efficient manner. Thus, the discussion starts by covering this aspect of transactional middleware and postpone the system issues till later.

Transactions are an abstraction used in database engines to implement the so called ACID properties (Atomicity, Consistency, Isolation, and Durability). Unlike what is commonly found in many textbooks, the vast majority of transactional middleware systems are not there to support the four ACID properties but only one: atomicity in distributed transactions and in the interaction between applications and databases. To the extent that records of the transaction executed are kept, transactional middleware can also provide durability but in a different sense than database durability and not as one of its main operational features. Some systems also supported isolation (e.g., Encina), but such systems run into the same difficulties that all other solutions for concurrent programming have faced and are still facing today.

Atomicity in the context of transactional middleware translates into running a 2 Phase Commit protocol among all the entities involved in executing parts of a transaction. The interactions and interfaces required to do so were standardized in the early 1990s as part of the X/Open models for transactions and distributed transactions, which also defined the XA interface. The XA interface describes how a transaction manager (the transactional middleware) interacts with a resource manager (e.g., a database) to run a 2 Phase Commit protocol. Once standardized, the XA interface allowed TP Monitors to execute transactions across heterogeneous systems, greatly simplifying the task of writing code integrating several systems.

The standard and generic procedure for running a transaction in a transactional middleware platform

involves invoking some primitive that tells the middleware that a transaction needs to be started. This primitive typically hides a call to a transaction manager that records the transaction and return and identifier and a context for the transaction. The application then proceeds to make calls as part of this transaction. Each call is augmented with the transaction identifier and context, which allows the recipient of the call to realize that its execution has become part of a transaction and tells it of which one. The recipient of the call uses the transactional context it has received to register with the transactional manager. Part of this registration procedure also involves telling the transaction manager where the actual transaction will be executed (typically the part of the system that supports the XA interface). After registration, the transaction manager knows who is involved in running parts of the transaction, information that maintains in a transaction record that contains all participants in that transaction. When the original application invokes a commit on the transaction, the transaction manager uses the transaction record to start a 2 Phase Commit protocol between all the participants that have registered for this transaction. At the end of the protocol, the transaction manager informs the original application of the success or failure of the commit. If the commit was successful, the original application has the guarantee that the transaction has been executed and completed at all sites where it was invoked. If the commit fails, the original application has the guarantee that all side effects and changes made by the transaction have been rolled back. Of course, these guarantees hold only if the code that is being invoked does not cause any side effects or persistent changes outside the transaction (e.g., on a sub-system that does not support the XA interface) as those changes will not be rolled back for obvious reasons.

The system side of transactional middleware involves all the machinery necessary to run transactions plus a great deal of additional functionality to cope with distribution. Historically, transactional middleware also had to provide a wide range of functionality to compensate for the lack of support at the operating system level. A good example is support for multi-threading, which is necessary to allow the system to cope with concurrent requests to the same service. Other examples include name and directory services, load balancing, life cycle management for services, logging, domain definition, and authentication. For performance reasons, many commercial products also supported specialized network interfaces (e.g., to communicate with a mainframe) and provided additional sub-systems such as message queuing or transactional file systems. Providing such a wealth of functionality made transactional middleware, especially TP Monitors, very generic tools where sometimes the support for transaction management was not the primary reason to use the system. To a large extent, especially during the 1980s and the first half of the 1990s, transactional middleware was the most efficient way to build a distributed system and TP Monitors became not only middleware but also development platforms for distributed applications. A good example of this was Encina (the commercial version of Camelot [2]), which provided its own versions or C or C++ (Transactional C and Transactional C++).

Key Applications
High performance transaction processing (financial, on line trading, banking).

Application servers (web shops, multi tier systems).

Future Directions
TP Monitors are and will remain a key component of any enterprise computing solution. As the key functionality of TP Monitors (the ability to efficiently process large volumes of transactions) is embedded deeper and deeper within larger systems, some of the additional functionality that conventional TP Monitors always provided has migrated to other platforms (e.g., application servers) or become independent systems on their own (e.g., message brokers). Hence the generic name of transactional middleware when referring to such systems. A challenge in the future will be providing similar transactional semantics on Service Oriented Architectures where the interactions might be based on asynchronous messages instead of through blocking calls (RPC/RMI). What the proper transactional semantics are for an asynchronous enterprise bus or an event based architecture is a topic that is still open and needs attention.

Cross-references
▶ ACID Properties
▶ Advanced Transaction Models
▶ Transaction
▶ Transaction Tuning
▶ Two-Phase Commit

T

Recommended Reading

1. Bernstein P.A. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, Los Altos, CA, 1997.
2. Eppinger J.L., Mummert L.B., and Spector A.Z (eds.). Camelot and Avalon. Morgan Kaufmann, Los Altos, CA, 1991.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, (3rd edn.). Prentice Hall, Englewood Cliffs, NJ, 2009.
4. Reuter A. and Gray J. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, Los Altos, CA, 1993.
5. Weikum G. and Vossen G. Transactional Information Systems. Morgan Kaufmann, Los Altos, CA, 2001.

Transactional Processes

HEIKO SCHULDT
University of Basel, Basel, Switzerland

Synonyms

Transactional workflows; Transactional business processes

Definition

A *Transactional Process* is a partially ordered sequence of activities which is executed in a way that guarantees transactional consistency for all activities or a subset of them. Activities can be either transactional (e.g., they are again transactional processes or conventional database transactions) or non-transactional (e.g., invocations of application services). Activities are ordered by means of control flow and data flow dependencies.

Key Points

In most cases, transactional consistency for processes focuses on notions of atomicity which go beyond the "all-or-nothing" semantics of ACID database transactions. This is done by supporting a process designer in explicitly specifying how consistency has to be achieved. Models for transactional processes include support for failure handling which can either be integrated into control flow dependencies (i.e., distinguishing between regular execution and alternative executions in case of failures) or, in a rather programmatic style, as exception handlers. Failure handling also needs to take into account that certain activities can neither be compensated after having been successfully executed (i.e., to semantically undo their effects) nor be deferred until the end of a process, due to control flow dependencies. For this, backward recovery (partial compensation in case of failures) and forward recovery (alternative executions) need to be combined. Forward recovery also requires to make the state of a process persistent during execution so that it can be resumed in case of system failures. For applications in which also concurrent executions of transactional processes need to be controlled, isolation and atomicity can be jointly considered.

Commercial tools for transactional processes distinguish between microflows (short running processes where all activities are transactional) and macroflows (long-running processes). The former can actually be considered conventional database transactions and are executed using protocols such as two phase commit (2PC) while the latter have to be treated as transactional processes. Several Web services standards define protocols that allow to execute Web services in a transactional context and thus facilitate their integration into transactional processes.

Cross-references

▶ Distributed Transaction Management
▶ Generalization of ACID Properties
▶ Multilevel Transactions and Object-Model Transactions
▶ Transactional Middleware
▶ Workflow Transactions

Recommended Reading

1. Alonso G. Transactional business processes. In Process-Aware Information Systems. M. Dumas, W. Aalst, and van der A. ter Hofstede (eds.), Wiley, New York, 2005, pp. 257–278.
2. Leymann F. Supporting business transactions via partial backward recovery in workflow management systems. In Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95), Informatik Aktuell, Dresden, Germany, March 1995. Springer Verlag, Berlin, 1995, pp. 51–70.
3. Schuldt H., Alonso G., Beeri C., and Schek H.-J. Atomicity and isolation in transactional processes. ACM Trans. Database Syst., 27(1):63–116, March 2002.
4. Zhang A., Nodine M., and Bhargava B. Global scheduling for flexible transactions in heterogeneous distributed database systems. IEEE Trans. Knowl. Data Eng., 13(3):439–450, 2001.

Transactional Workflows

▶ Transactional Processes
▶ Workflow Transactions

Transaction-Time Access Methods

Transaction-Time Algebras

Transaction-Time Data Model

Transaction-Time Indexing

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]The Federal University of Rio Grande do Sol, Porte Alegre, Brazil
[2]University of California-Riverside, Riverside, CA, USA

Synonyms

Transaction-time access methods

Definition

A transaction-time index is a temporal index that enables fast access to transaction-time datasets. In a traditional database, an index is used for selection queries. When accessing transaction-time databases, selection queries also involve the transaction-time dimension. The characteristics of the transaction-time axis imply various properties that such temporal index should have to be efficient. As with traditional indices, the performance is described by three costs: (i) storage cost (i.e., the number of pages the index occupies on the disk), (ii) update cost (the number of pages accessed to perform an update on the index; for example when adding, deleting or updating a record), and (iii) query cost (the number of pages accessed for the index to answer a query).

Historical Background

Most of the early work on temporal indexing has concentrated on providing solutions for transaction-time databases. A basic property of transaction-time is that it always *increases*. This is consistent with the serialization order of transactions in a database system. Each newly recorded piece of data are time-stamped with a new, larger, transaction time. The immediate implication of this property is that previous transaction times *cannot* be changed (since every new change must be stamped with a new, larger, transaction time). This is useful for applications where every action must be registered and maintained unchanged after registration, as in auditing, billing, etc. Note that a transaction-time database records the history of a database activity rather than "real" world history. As such it can "rollback" to, or answer queries for, any of its previous states.

Consider, for example, a query on a temporal relation as it was at a given transaction time. There are two obvious but inefficient approaches to support this query, namely the "copy" and "log" approaches. In the "copy" approach, the whole relation is "flushed" (copied) to disk for every transaction for which a new record is added or modified. Answering the above rollback query is simple: the system has to then query the copy that has the largest transaction-time less or equal to the requested time. Nevertheless, this approach is inefficient for its storage and update costs (the storage can easily become quadratic to the number of records in the temporal database and the update is linear, since the whole temporal relation needs to be flushed to disk for a single record update). In contrast, the "log" solution simply maintains a log of the updates to the temporal database. Clearly, this approach uses minimal space (linear to the number of updates) and minimal update cost (simply add an update record at the end of the log), but the query time is prohibitively large since the whole log may need to be traversed for reconstructing a past state of the temporal database. Various early works on transaction-time indexing behave asymptotically like the "log" or the "copy" approaches. For a worst-case comparison of these methods see [7]. Later on, two methodologies were proposed to construct more efficient transaction-time indices, namely the (i) *overlapping* [3,10] and (ii) (partially) *persistent* approaches [1,6,9,]. These methodologies attempt to combine the benefits of the fast query time from the "copy"

approach with the low space and update costs of the "log" approach.

Foundations
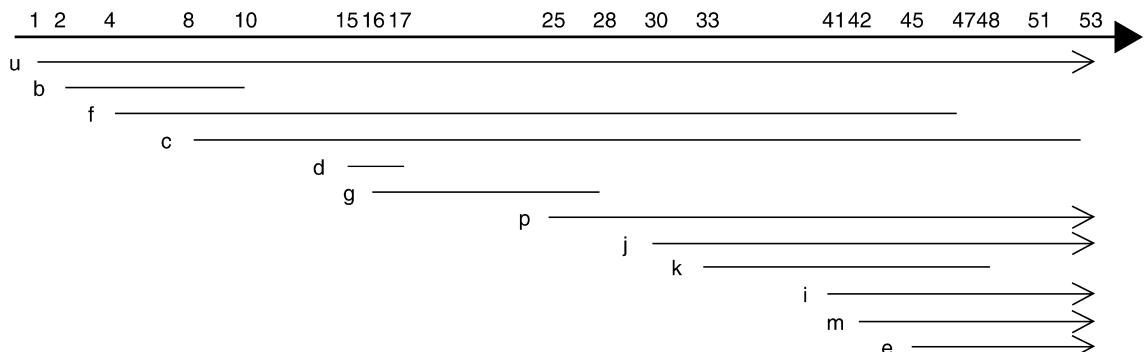
The distinct properties of the transaction-time dimension and their implications to the index design are discussed through an example; this discussion has been influenced by [8]. Consider an initially empty collection of objects. This collection evolves over time as changes are applied. Time is assumed discrete and always increasing. A change is the addition or deletion of an object, or the value change of an object's attribute. A real life example would be the evolution of the employees in a company. Each employee has a surrogate (*ssn*) and a salary attribute. The changes include additions of new employees (as they are hired or re-hired), salary changes or employee deletions (as they retire or leave the company). Each change is time-stamped with the time it occurs (if more than one change happen at a given time, all of them get the same timestamp). Note that an object attribute value change can be simply "seen" as the artificial deletion of the object followed by the simultaneous rebirth (at the same time instant) of this object having the modified attribute value. Hence, the following discussion concentrates only on object additions or deletions.

In this example, an object is called "alive" from the time that it is added in the collection until (if ever) it is deleted from it. The set $s(t)$, consisting of all alive objects at time t, forms the state of the evolving collection at t. Figure 1 illustrates a representative evolution shown as of time $t = 53$. Lines ending to ">" correspond to objects that have not yet been deleted at $t = 53$. For simplicity, at most one change per time instant is assumed. For example, at time $t = 10$ the state is $s(10) = \{u, f, c\}$. The interval created by the consecutive time instants an object is alive is the "lifetime" interval for this object. Note that the term "interval" is used here to mean a "convex subset of the time domain" (and not a "directed duration"). This concept has also been named a "period" in this discussion however, only the term "interval" is used. In Fig. 1, the lifetime interval for object b is [2,10). An object can have many non-overlapping lifetime intervals.

Note that in the above evolving set example, changes are always applied to the current state $s(t)$, i.e., past states *cannot* be changed. That is, at time $t = 7$, the deletion of object d is applied to $s(16) = \{u, f, c, d, g\}$ to create $s(17) = \{u, f, c, g\}$. This implies that, at time $t = 54$, no object can be retroactively added to state $s(5)$, neither the interval of object d can be changed to become [15,25]. All such changes are not allowed as they would affect previous states and not the most current state $s(53)$.

Assume that all the states $s(t)$ of the above evolution need to be stored in a database. Since time is always increasing and the past is unchanged, a transaction time database can be utilized with the implicit updating assumption that when an object is added or deleted from the evolving set at time t, a transaction updates the database system about this change at the same time, i.e., this transaction has commit timestamp t. When a new object is added in the collection at time t, a record representing this object is stored in the database accompanied by a transaction-time interval of the form *[t, UC)*. In this setting, *UC* (Until Changed) is a variable representing the fact that at the time the object is added in the collection, it is not yet known when (if ever) it will be deleted from it. If this object is later deleted at time t, the transaction-time interval

Transaction-Time Indexing. Figure 1. An example of a transaction-time evolution.

of the corresponding record is updated to *[t, t')*. A real-world object deletion is thus represented in the database as a "logical" deletion: the record of the deleted object is still retained in the database, accompanied by an appropriate transaction-time interval. Since the past is kept, a transaction-time database conceptually stores, and can thus answer queries about, any past state *s(t)*.

Based on the above discussion, an index for a transaction-time database should have the following properties: (i) store past logical states, (ii) support addition/deletion/modification changes on the objects of the current logical state, and (iii) efficiently access and query any database state.

Since a fact can be entered in the database at a different time than when it happened in reality, the transaction-time interval associated with a record is actually related to the process of updating the database (the database activity) and may not accurately represent the times the corresponding object was valid in reality. Note that a valid-time database has a different abstraction, which can be visualized as a dynamic collection of "interval-objects." The term interval-object is used to emphasize that the object carries a valid-time interval to represent the temporal validity of some object property. Reality is more accurately represented if both time dimensions are supported. A bi-temporal database has the characteristics of both approaches. Its abstraction maintains the evolution (through the support of transaction-time) of a dynamic collection of (valid-time) interval-objects.

Traditional indices like the B$^+$-tree or the R-tree are not efficient for transaction-time databases because they do not take advantage of the special characteristics of transaction time (i.e., that transaction time is always increasing and that changes are always applied on the latest database state). There are various index proposals that are based on the (partially) *persistent* data-structure approach; examples are the Time-Split B-tree (TSB) [6], the Multiversion B-tree (MVBT) [1], the Multiversion Access Structure [11], the Snapshot Index [9], etc. It should be noted that all the above approaches facilitate "time-splits": when a page gets full, current records from this page are copied to a new page (this operation is explained in detail below). Time-splits were first introduced in the Write-Once B-tree (WOBT), a B-tree index proposed for write-once disks [5]. Later, the Time-Split B-tree used time-splits for read-write media and also introduced

other splitting policies (e.g., splitting by other than the current time, key splits etc.) Both the WOBT and TSB use deletion markers when records are deleted and do not consolidate pages with few current records. The MVBT uses the time splitting approach of the WOBT, drops the deletion markers and consolidated pages with few current records. It thus achieves the best asymptotic behavior and it is discussed in detail below. Among the index solutions based on the *overlapping* data-structure approach [2], the Overlapping B-tree [10] is used as a representative and discussed further.

For the purposes of this discussion, the so-called *range-timeslice* query is considered, which provides a key range and a specific time instant selection. For example: "find all objects with keys in range $[K_1, K_2]$ whose lifetimes contain time instant *t*." This corresponds to a query "find the employees with ids in range $[100,..,500]$ whose entries were in the database on July 1, 2007." Let *n* be the total number of updates in a transaction-time database; note that *n* corresponds to the minimal information needed for storing the whole evolution. [7] presents a lower bound for answering a range-timeslice query. In particular, any method that uses linear space (i.e, O(*n/B*) pages, where *B* is the number of object records that fit in a page) would need O($\log_B n + s/B$) I/O's to answer such a query (where an I/O transfers one page, and *s* corresponds to the size of the answer, i.e., the number of objects that satisfy the query).

The Multiversion B-tree (MVBT): The MVBT approach transforms a timeslice query to a partial persistence problem. In particular, a data structure is called *persistent* [4] if an update creates a new version of the data structure while the previous version is still retained and can be accessed. Otherwise, if old versions of the structure are discarded, the structure is termed *ephemeral*. *Partial* persistence implies that only the newest version of the structure can be modified to create a new version.

The key observation is that partial persistence "suits" nicely transaction-time evolution since these changes are always applied on the latest state *s(t)* of the evolving set (Fig. 1). To support key range queries on a given *s(t)*, one could use an ordinary B$^+$-tree to index the objects in *s(t)* (that is, the keys of the objects in *s(t)* appear in the data pages of the B$^+$-tree). As *s(t)* evolves over time through object changes, so does its corresponding B$^+$-tree. Storing copies of all the states that the B$^+$-tree took during the evolution of *s(t)*

is clearly inefficient. Instead, one should "see" the evolution of the B$^+$-tree as a partial persistence problem, i.e., as a set of updates that create subsequent versions of the B$^+$-tree.

Conceptually, the MVBT stores all the states assumed by the B$^+$-tree through its transaction-time evolution. Its structure is a directed acyclic graph of pages. This graph embeds many B$^+$-trees and has a number of root pages. Each root is responsible for providing access to a subsequent part of the B$^+$-tree's evolution. Data records in the MVBT leaf pages maintain the transaction-time evolution of the corresponding B$^+$-tree data records (that is, of the objects in $s(t)$). Each record is thus extended to include an interval [*insertion-time, deletion-time*), representing the transaction-times that the corresponding object was inserted/deleted from $s(t)$. During this interval the data-record is termed *alive*. Hence, the MVBT directly represents object deletions. Index records in the non-leaf pages of the MVBT maintain the evolution of the corresponding index records of the B$^+$-tree and are also augmented with insertion-time and deletion-time fields.

Assume that each page in the MVBT has a capacity of holding B records. A page is called *alive* if it has not been *time-split* (see below). With the exception of root pages, for all transaction-times t that a page is alive, it must have at least q records that are alive at t ($q < B$). This requirement enables clustering of the alive objects at a given time in a small number of pages, which in turn will minimize the query I/O. Conceptually, a data page forms a rectangle in the time-key space; for any time in this rectangle the page should contain at least q alive records. As a result, if the search algorithm accesses this page for any time during its rectangle, it is guaranteed to find at least q alive records. That is, when a page is accessed, it contributes enough records for the query answer.
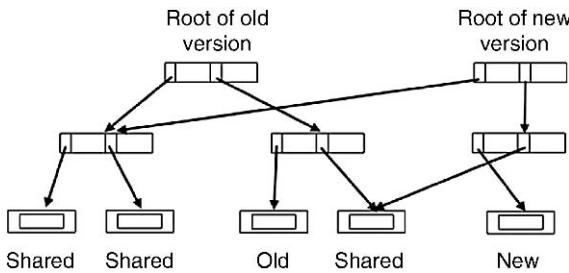
The first step of an update (insertion or deletion) at the transaction time t locates the target leaf page in a way similar to the corresponding operations in an ordinary B$^+$-tree. Note that, only the latest state of the B$^+$-tree is traversed in this step. An update leads to a *structural change* if at least one new page is created. *Non-structural* are those updates which are handled within an existing page.

After locating the target leaf page, an insert operation at the current transaction time t adds a data record with a transaction interval of [t, UC) to the target leaf page. This may trigger a structural change in the MVBT, if the target leaf page already has B records. Similarly, a delete operation at transaction time t finds the target data record and changes the record's interval to [insertion-time, t). This may trigger a structural change if the resulting page ends up having less than q alive records at the current transaction time. The former structural change is called a *page overflow*, and the latter is a *weak version underflow* [1]. Page overflow and weak version underflow need special handling: a *time-split* is performed on the target leaf-page. The time-split on a page x at time t is performed by copying to a new page y the records alive in page x at t. Page x is considered *dead* after time t. Then the resulting new page has to be incorporated in the structure [1].

Since updates can propagate to ancestors, a root page may become full and time-split. This creates a new root page, which in turn may be split at a later transaction time to create another root and so on. By construction, each root of the MVBT is alive for a subsequent, non-intersecting transaction-time interval. Efficient access to the root that was alive at time t is possible by keeping an index on the roots, indexed by their time-split times. Since time-split times are in order, this root index is easily kept (this index is called the root* in [1]). In general, not many splits propagate to the top, so the number of root splits is small and the root* structure can be kept in main memory. If this is not the case, a small index can be created on top of the root* structure.

Answering a range-timeslice query on transaction time t has two parts. First, using the root index, the root alive at t is found. This part is conceptually equivalent to accessing $s(t)$ or, more explicitly, accessing the B$^+$-tree indexing the objects of $s(t)$. Second, the answer is found by searching this tree in a top-down fashion as in a B$^+$-tree. This search considers the record transaction interval. The transaction interval of every record returned or traversed should include the transaction time t, while its key attribute should satisfy the key query predicate. A range-timeslice query takes $O(\log_B n + s/B)$ I/O's while the space is linear to n (i.e., $O(n/B)$). Hence, the MVBT optimally solves the range-timeslice query. The update processing is $O(\log_B m)$ per change, where m is the size of $s(t)$ when the change took place. This is because a change that occurred at time t traverses what is logically a B$^+$-tree on the m elements of $s(t)$.

Transaction-Time Indexing. Figure 2. The Overlapping B-tree.

The Overlapping B-tree: Similar to the MVBT approach, the evolving set $s(t)$ is indexed by a B^+-tree. The intuition behind the Overlapping B-tree [2,10] is that the B^+-trees of subsequent versions (states) of $s(t)$ will not differ much. A similar approach was taken in the EXODUS DBMS [3]. The Overlapping B-tree is thus a graph structure that superimposes many ordinary B^+-trees (Fig. 2). An update at some time t creates a new version of $s(t)$ and a new root in the structure. If a subtree does not change between subsequent versions, it will be shared by the new root. Sharing common subtrees among subsequent B^+-trees is done through index nodes of the new B^+-tree that point to nodes of previous B^+-tree(s). An update will create a new copy of the data page it refers to. This implies that a new path is also created leading to the new page; this path is indexed under the new root.

To address a range-timeslice query for a given transaction time t, the latest root that was created before or at t must be found. Hence, roots are time-stamped with the transaction time of their creation. These timestamps can be easily indexed on a separate B^+-tree. This is similar to the MVBT root* structure; however, the Overlapping B-tree creates a new root per version. After the appropriate root is found, the search continues traversing the tree under this root, as if an ordinary B^+-tree was present for state $s(t)$.

Updating the Overlapping B-tree involves traversing the structure from the current root version and locating the data page that needs to be updated. Then a copy of the page is created as well as a new path to this page. This implies $O(\log_B m)$ I/O's per update, where m is the current size of the evolving set. An advantage of the Overlapping structure is in the simplicity of its implementation. Note that except the roots, the other nodes do not involve any time-stamping. Such time-stamping is not needed because pages do not have to share records from various versions. Even if a single

record changes in a page, the page cannot be shared by different versions; rather a new copy of the page is created. This, however, comes at the expense of the space performance. The Overlapping B-tree occupies $O(n \log_B n)$ pages since, in the worst case, every version creates an extra tree path. Further performance results on this access method can be found in [10].

Key Applications
The characteristics of transaction-time make such databases ideal for applications that need to maintain their past; examples are: billing, accounting, tax-related etc.

Cross-references
► Bi-Temporal Indexing
► B+-Tree
► Temporal Database
► Transaction Time
► Valid Time
► Valid-Time Indexing

Recommended Reading
1. Becker B., Gschwind S., Ohler T., Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J., 5(4):264–275, 1996.
2. Burton F.W., Huntbach M.M., and Kollias J.G. Multiple generation text files using overlapping tree structures. Comput. J., 28(4):414–416, 1985.
3. Carey M.J., DeWitt D.J., Richardson J.E., and Shekita E.J. Object and file management in the EXODUS extensible database system. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 91–100.
4. Driscoll J.R., Sarnak N., Sleator D.D., and Tarjan R.E. Making data structures persistent. J. Comput. Syst. Sci., 38(1):86–124, 1989.
5. Easton M.C. Key-sequence data sets on inedible storage. IBM J. Res. Dev., 30(3):230–241, 1986.
6. Lomet D. and Salzberg B. Access methods for multiversion data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 315–324.
7. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
8. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.
9. Tsotras V.J. and Kangelaris N. The snapshot index: an I/O-optimal access method for timeslice queries. Inf. Syst., 20(3):237–260, 1995.
10. Tzouramanis T., Manolopoulos Y., and Lorentzos N.A. Overlapping B+-trees: an implementation of a transaction time access method. Data Knowl. Eng., 29(3):381–404, 1999.
11. Varman P.J. and Verma R.M. An efficient multiversion access structure. IEEE Trans. Knowl. Data Eng., 9(3):391–409, 1997.

Transcriptional Networks

Transformation

Transformation Engines

Translation Lookaside Buffer (TLB)

Translingual Information Retrieval

Tree Drawing

Tree Pattern Queries

Tree-based Indexing

Yannis Manolopoulos[1], Yannis Theodoridis[2],
Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki,
Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Riverside, Riverside,
CA, USA

Synonyms

Index Sequential Access Method (ISAM); B+-tree;
R-tree

Definition

Consider a relation R with some numeric attribute A taking values over an (ordered) domain D. A *range query* retrieves all tuples in R whose attribute A has values in the interval [*low, high*]. That is, $low \leq R.A \leq high$. To enable fast processing of range selection queries, an access method that maintains order is needed. Such an index has the form of a tree, where each node corresponds to a page. Leaf nodes contain (or index) the actual values of A, while index nodes provide ordered access to the nodes underneath. Examples of tree-based indexing are the B+-tree and the R-tree (for single- and multi-dimensional ranges, respectively).

Key Points

A major performance goal of a database management system is to minimize the number of I/O's (i.e., blocks or pages transferred) between the disk and main memory. One way to achieve this goal is to minimize the number of I/O's when answering a query. Consider for example relation *Employee* (*ssn, name, salary, dept, address*); the query: "Find the employees who reside in Santa Monica" references only a fraction of *Employee* records. It would be very inefficient to have the database system sequentially read all the pages of the *Employee* file and check the *address* field of each employee record for the value "Santa Monica." Instead the system should be able to locate the pages with "Santa Monica" employee records directly.

To allow such fast access additional data structures called access methods (or indices) are designed per database file. The term *search-key* is used to identify the attribute(s) on which the access method is built. There are two fundamental access methods, namely tree-based and hash-based indexing. They differ on the kind of queries that they can efficiently address. Tree-based indexing maintains the order of the search-key values. It is thus applicable to attributes that are *numeric* and hence it can be used to address *range* queries (and also *equality* queries, when the range query interval is reduced to one value). Hash-based indexing on the other hand does not assume any ordering; rather it is based on mapping the search-key values on a collection of buckets. Therefore it can only address *equality* (or *membership*) queries. Since a tree-based index maintains the order of the indexed values in its leaf pages, a (one-dimensional) range query is implemented as a search for the leaf page

with the lower value of the range interval, followed by the accessing of sibling pages until a page that contains the higher value of the range interval is reached.

Tree-based indices are further categorized by whether their search-key ordering is the same with the file's logical order (if any). Note that a file may or may not be ordered according to the value of an (a sequence of) attribute(s). A file stored without any logical order is called an unordered file or heap. If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. Since such an index also clusters the values of the file, the term *clustering* index has also been used. The search-key of a primary index is usually the file's primary key, however this is not necessary. An index built on any non-ordering attribute of a file is called *secondary*. A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index and the rest are secondary ones (obviously, a file can have at most a single logical order since it is physically stored once).

Historically, the first tree-based index used in relational databases was the ISAM file (a static tree). Currently, the most widely used tree-based index is the B+-tree which is a dynamic structure (i.e., its size increases or decreases as the size of the indexed file changes by record insertions/deletions). As with any data structure, the performance of an access method is characterized by three costs, namely: query time, update time and space. Query time corresponds to the number of page accesses (I/O's) needed to answer a query. Update time counts the number of I/O's needed for updating the method when a record is updated, inserted or deleted. Space measures the number of pages occupied by the method's data structures. The B + -tree uses logarithmic update and query costs while its space requirements are linear to the size of the indexed file.

Cross-references

Recommended Reading

1. Elmasri R.A., and Navathe S.B. Fundamentals of Database Systems (5th edn.). Addison-Wesley, Reading, MA, 2007.
2. Manolopoulos, Theodoridis Y., Tsotras. Y., and Vassilis. J., Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Ramakrishnan R. and Gehrke J. Database Management Systems (3rd edn.). McGraw-Hill, NY, 2003.

Treemaps

JEAN-DANIEL FEKETE
INRIA, LRI University Paris Sud, Orsay Cedex, France
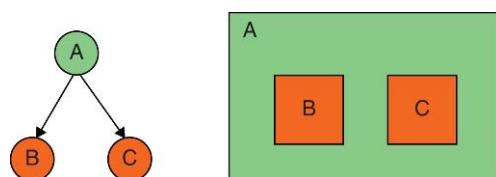
Definition

Treemaps [5] have been designed to visualize rooted trees using containment to express the hierarchy (Fig. 1 right). It is a space-filling technique and uses all the available space to render the tree. One important feature of Treemaps is their use of surface to represent an attribute value that adds up with the hierarchy. The number of leaves under a node is the simplest of these attributes. This number is one for a leaf and, for an interior node, it is the sum of the number of leaves of all its children.

This additive property is frequent on real trees. In an organizational chart, the salary of employees adds up, in an administrative geographical decomposition such as the one used by the census, population adds up, as well as income.

The traditional visual representation of rooted trees is called a *node-link diagram* (Fig. 1 left). Treemaps are not as good as standard node-link diagrams to represent the hierarchy depth on the tree; if an important task on the tree is to compare the depth of two nodes, node-link diagrams are more appropriate.

However, Treemaps can be applied on trees ranging from tens to hundred-thousands of nodes, much more

Treemaps. Figure 1. Rendition of a rooted tree as a node-link diagram on the left and as a Treemap on the right.

than standard node-link diagrams. Computing the layout of Treemaps is fast and their reading is easy with some training.

Historical Background

Treemaps were introduced by Shneiderman in 1992 [5]. They are restrictions of Euler diagrams to represent inclusion. Furthermore, when Euler diagrams are usually drawn with circles, ellipses, or curved shapes, Treemaps are drawn with rectangles to better use the screen real-estate.

The original Treemap algorithm is called "Slice and Dice" because at each level in the hierarchy, the children of a node cut the node's rectangle in slices horizontally or vertically, flipping direction at each level. Further refinements of Treemaps have tried to improve two main features: the aspect ratio of the rectangles and their order.

With the original "Slice and Dice" algorithm, the ratio between width and height of nodes can vary widely in the same Treemap. Some slices can become very thin while others can be almost square. Comparing visually the surface of rectangles with different aspect ratio is difficult for the human vision so one important improvement was to try to create nodes as square as possible. The most popular algorithm for achieving this property is due to van Wijk and is called "Squarified Treemap" [3].

Neither the "slice and Dice," nor the squarified algorithm maintains the order of children visually. When the leaf order is important, Bederson et al. have compared several algorithms and found that a simple variant of the squarified algorithm called "Ordered Treemaps" provided a good tradeoff between complexity and performance [2].

Further research have been conducted to study the limits of Treemaps in term of tree size and density [4] or to enhance their readability. Variant layouts have also been proposed, such as circular Treemaps (not space efficient) and Voronoï Treemaps [1] that are space efficient but costly to compute.

Foundations

As in all the visualization techniques, Treemaps visualizations consist in computing a *layout* for the tree – here a rectangle associated with each node – and drawing these rectangles using adequate visual attributes.

A rooted tree T is defined as a set of nodes $n \in T$ and three functions *root*, *parent* and *children*. It is convenient to define the element NIL to represent an undefined node. A rooted tree verifies the following axioms:

$$\text{parent}(T, n) \in \ T \cup \{\text{NIL}\}$$
$$\text{parent}(T, n) = \text{NIL} \Leftrightarrow n = root(T)$$
$$\text{children}(T, n) = \{c_1, c_2, ... c_k\} \in T^k$$
$$\text{parent}(T, n) = p \Leftrightarrow n \in \text{children}(T, p)$$

Algorithm 1: Slice and Dice Treemap Algorithm

```
procedure DRAWSLIDEANDDICE(T, r)                      ▷ Draws T in rectangle r
    if width(r) > height(r) then
        DRAWSLIDEANDDICE(root(T), r, horizontal)
    else
        DRAWSLIDEANDDICE(root(T), r[vertical])
    end if
end procedure
procedure DRAWSLIDEANDDICE(n, r, d)
    c ← children(n)
    if c = θ then
        DRAWRECTANGLE(r)                              ▷ Leaf node, just draw it
    else if d = vertical then
        len ← height(r)
        dw ← len/ weight(n)
        y = miny(r)
        for all m ∈ c do
            w = weight(m) × dw
            nr ← [xmin(r), xmax(r), y, y + w]
            DRAWSLIDEANDDICE(m, nr, FLIP(d))
            y ← y + w
        end for
    else                                              ▷Symmetrical to the horizontal case
    end if
end procedure
```

Treemaps. Figure 2. SequoiaView using the Cushion Treemap rendition technique for a large file hierarchy.

Treemaps. Figure 3. A Treemap of a one million file Web server, using intensity for depth perception and smooth shading to render the rectangles.

Algorithm 2: Squarified Treemap Algorithm

```
procedure DRAWSQUARIFIED(n, r)
    if is Leaf (n) then                          ▷ Leaf node, just draw it
        DRAWRECTANGLE(r)
        return
    end if
    scale ← width(r) * height(r) / weight(n)
    sorted = SORTBYDECREASINGWEIGHT(children(n))
    while sorted ≠ nil do
        if width(d) > height(`) then             ▷ Create vertical strips
            y = miny(r)
            len = height(r)
            (strip, sorted) = SQUARIFY(sorted, len, scale)
            width ← ∑c∈strip weight(c) × scale/ len
            for all child ∈ strip do
                h = weight(child) × scale / width
                nr ← [xmin(r), xmax(r),y, y + h]
                DRAWSQUARIFIED(child, nr)
                y ← y + h
            end for
            r ← [xmin(r) + width, xmax(r), ymin(r), ymax(r)]
        else
                                                 ▷ Symmetrical for the horizontal case
        endif
    endwhile
end procedure
function SQUARIFY(children, w, scale)            ▷ w is the strip width
    strip ← [pop(children)]                      ▷ Strip starts with one node
    while children ≠ ∅ do
        c ← head(children)
        if WORST(strip, w, scale) ≤ WORST(strip + c, w, scale) then
            strip ← strip + c
            pop(children)
            c ← head(children)
        else
            break
        end if
    end while
    return (strip, children)
end function
```

For convenience, define the degree of a node deg(T,n) to be the number of nodes in its children set and the *is Leaf*(T, m) predicate is deg(T, n) = 0.

Drawing a Treemap requires a positive function weight on the tree such that:

$$\text{weight}(T, n) = \sum_{c \in \text{children}(T,n)} \text{weight}(T, c) \land$$
$$\text{weight}(T, n) > 0 \tag{1}$$

A rectangular portion of the screen is used to display the whole tree. The layout algorithm is recursive and uses the *weight* function to allocate space. The algorithm for "Slice and Dice" is simple (Algorithm 1). The "flip" function and the accessor function for the rectangle type should be self explanatory.

The squarified algorithm is more complex. It tries to assign a rectangle to each node with the ratio w/h close to 1. Computing the optimal configuration is equivalent to bin-packing and is known to be NP-complete so it uses a simple heuristics. It sorts the children by decreasing weights and fills the parent rectangle with strips of children. Each node rectangle is added to the strip until adding the next rectangle decreases the quality of the rectangles already in the strip. Then a new strip is created (see Algorithm 2). The quality is usually the worst aspect ratio of the rectangles in the strip. It is computed as follows: given R a list of areas and s their total sum:

$$worst(R, w) = \max_{r \in R}(\max(w^2 r/s^2, s^2/(w^2 r)))$$

In the algorithm 2, a scale argument is passed to the function *worst* to transform the weights in surfaces since in the definition (equation 1), the weights are in arbitrary units. The computation of strips can be done incrementally as described in [3].

Sorted Treemaps only change two lines to the Algorithm 2: line 7 is removed so the nodes are in their natural order and line 9 is removed so the strips are always horizontal.

Key Applications

Treemaps have been used on any kinds of trees. They became visible to a large audience when the company SmartMoney used Treemaps to visualize a "map of the market" (still visible at www.smartmoney.com/mapofthemarket. Other applications include browsers for file systems where one can visualize the whole hierarchy and navigate on it by file sizes and using sophisticated color rendition for the rectangles, such as "cushion" (Fig. 2) or "smooth shading" (Fig. 3).

They are also very useful to visualize data tables using a flexible hierarchy. A set of column is chosen and ordered. The first level of the tree consists in creating a node for each values of the first column (partitioning according to the first column). The second level is built by sub-partitioning by values of the second column etc. This method is well suited to OLAP databases where the partitioning is already available. By interactively changing the column order, different Treemap views of the table can be explored.

Cross-references

▶ Data Visualization
▶ Dense Pixel Displays
▶ Graphic Information Processing
▶ Graphical Perception
▶ Hierarchical Data Model
▶ Hierarchy
▶ Human-Computer Interaction
▶ Visual Analytics
▶ Visual Data Mining
▶ Visual Representation
▶ Visualization for Information Retrieval
▶ Visualizing Hierarchical Data

Recommended Reading

1. Balzer M. and Deussen O. Voronoi treemaps. In Proc. IEEE Symp. on Information Visualization. 2005, pp. 48–56.
2. Bederson B.B., Shneiderman B., and Wattenberg M. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. ACM Trans. Graph., 21(4):833–854, 2002.
3. Bruls M., Huizing K., and van Wijk J.J. Squarified treemaps. In Data Visualization 2000, Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization, W. de Leeuw and R. van Liere (eds.). Springer, Vienna, 2000, pp. 33–42.
4. Fekete J.D. and Plaisant C. Interactive information visualization of a million items. In Proc. IEEE Symp. on Information Visualization, 2002, pp. 117–124.
5. Shneiderman B. Tree visualization with tree-maps: 2-d space-filling approach. ACM Trans. Graph., 11(1):92–99, 1992.

Tree-Structured Classifier

▶ Scalable Decision Tree Construction

Triangular Norms

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Synonyms

t-Norm

Definition

Triangular norms (briefly t-norms) are special binary operations $\mathbf{T} : [0,1]^2 \rightarrow [0,1]$. They are interesting for fuzzy logic because they preserve the fundamental properties of the logical conjunction "and" (to hold at the same time), namely commutativity, monotonicity, associativity, and boundedness and thus, they serve as a natural generalization of the classical conjunction in many-valued logical systems.

A concept associated with the t-norm is the triangular conorm (t-conorm) $\mathbf{S} : [0,1]^2 \rightarrow [0,1]$. This corresponds to the behaviour of truth values when joined by the logical connective "or."

Key Points

A t-norm is a binary operation $\mathbf{T} : [0,1]^2 \rightarrow [0,1]$ such that the following axioms are satisfied for all $a, b, c \in [0,1]$:

(commutativity)	$a \mathbf{T} b = b \mathbf{T} a$,
(associativity)	$a \mathbf{T}(b \mathbf{T} c) = (a \mathbf{T} b)\mathbf{T} c$,
(monotonicity)	$a \leq b$ implies $a \mathbf{T} c \leq b \mathbf{T} c$,
(boundary condition)	$1 \mathbf{T} a = a$.

The most important t-norms are *minimum*, *product* and *Łukasiewicz conjunction* defined by

$$a \; \mathbf{T}_\mathrm{L} b = \max\{0, a + b - 1\}.$$

A t-conorm is a binary operation $\mathbf{S} : [0,1]^2 \to [0,1]$, which is commutative, associative, monotone, and for all $a \in [0,1]$ it fulfils the following boundary condition:

$$0 \; \mathbf{S} \; a = a.$$

The most important t-conorms are *maximum*, *probabilistic sum*

$$a \; \mathbf{S}_P \; b = a + b - ab$$

and *Łukasiewicz disjunction*

$$a \; \mathbf{S}_\mathrm{L} b = \min\{1, a + b\}.$$

A t-conorm is dual to the given t-norm $\mathbf{T}$ if $a \, \mathbf{S} \, b = 1 - (1-a) \, \mathbf{T} \, (1-b)$ holds for all $a, b \in [0,1]$. There are many classes of t-norms and their structure is extremely complicated.

Cross-references

► Approximate Reasoning
► Fuzzy IF-THEN Rules
► Fuzzy Relation
► Fuzzy Set
► Residuated Lattice

Recommended Reading

1.　Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.

Triangulated Irregular Network

Leila De Floriani, Paola Magillo
University of Genova, Genova, Italy

Synonyms

Triangulated terrains; TIN

Definition

A Triangulated Irregular Network (TIN) is a special case of a Digital Elevation Model (DEM).

A terrain can be mathematically modeled as a function $z = f(x, y)$ mapping a point (x, y) in a domain D in the plane to its elevation value $f(x, y)$. In practice, the value of function f is known at a finite set S of points within D. A DEM provides an estimated value for function f at any point (x, y) of the domain, based on the values at the points of S. A DEM consists of a subdivision of the domain into cells and of a piece-wise interpolating function defined on such cells.

A TIN is a DEM in which the domain subdivision is a triangle mesh, i.e., a set T of triangles such that: (i) the set of vertices of T is S, (ii) the interiors of any two triangles of T do not intersect, (iii) if the boundaries of two triangles intersect, then the intersection is either a common vertex, or a common edge.

Usually, a linear interpolating function is defined on the triangles of T, thus providing a continuous terrain approximation. Given a triangle $t = P_1 P_2 P_3$ and a point $P = (x, y)$ inside triangle t, the following function estimates the elevation value z at P. Let $P_1 = (x_1, y_1, z_1)$ $P_2 = (x_2, y_2, z_2)$ and $P_3 = (x_3, y_3, z_3)$ be the coordinates of the three vertices of t. Then,

$$z = z_1 - (a(x - x_1) + b(y - y_1))/c,$$

where (a, b, c) are the components of the normal vector to the triangle $P_1 P_2 P_3$ in 3D space:

$$a = (y_1 - y_2)(z_1 - z_3) - (z_1 - z_2)(y_1 - y_3)$$
$$b = (z_1 - z_2)(x_1 - x_3) - (x_1 - x_2)(z_1 - z_3)$$
$$c = (x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)$$

Key Points

TINs have been extensively studied in Geographic Information Systems (GISs), in Computational Geometry, and in Computer Graphics. Several data structures and algorithms for representing, constructing and manipulating triangle meshes have been proposed.

The quality of the terrain approximation provided by a TIN depends on the underlying triangle mesh. Note that a point set S does not define a unique triangle mesh. The most widely used triangle mesh is the *Delaunay* one, in which the circumcircle of each triangle does not contain any data point in its interior. This means that the triangles of a Delaunay mesh are as much equiangular as possible. It has also been proven that the use of a Delaunay mesh as the basis of a TIN improves the quality of the terrain approximation and enhances numerical stability in computations. Other triangulation criteria have been proposed which consider the triangles of the mesh in 3D space.

Often, not only points, but also lines need to be included in a TIN. Such lines may represent

morphological terrain features (coast lines, rivers, ridges), man-made structures (roads, railways, gas lines), political or administrative boundaries, or contour lines. The Delaunay criterion has been modified to deal with lines in two different ways: (i) in the *constrained Delaunay triangulation*, the lines appear as triangle edges (but it may present sliver triangles); (ii) in the *conforming Delaunay triangulation*, each line is discretized by adding points on it (but a large number of points may need to be added).

TINs are used in multiresolution terrain modeling.

Cross-references

Recommended Reading

1. de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. Computational Geometry – Algorithms and Applications. 2nd ed. Springer-Verlag, Berlin, 2000.
2. De Floriani L., Magillo P., and Puppo E. Applications of computational geometry to Geographic Information Systems. Chapter 7 in Handbook of Computational Geometry, J.R. Sack, J. Urrutia (eds.). Elsevier Science, 1999, pp. 333–388.
3. van Kreveld M. Digital elevation models and TIN algorithms. In Algorithmic Foundations of Geographic Information Systems, M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer (eds.). Springer-Verlag, Berlin, 1997, pp. 37–78.

Triangulated Terrains

Trie

Maxime Crochemore[1,2], Thierry Lecroq[3]
[1]King's College London, London, UK
[2]University of Paris-East, Paris, France
[3]University of Rouen, Rouen, France

Synonyms

Prefix tree

Definition

A trie is a rooted tree used for storing associative arrays where keys are usually strings. Edges are often labeled by individual symbols. Then common prefixes are factorized. Each node of the trie is associated with a prefix of a string of the set of strings: concatenation of the labels of the path from the root to the node. The root is associated with the empty string. Strings of the set are stored in terminal nodes (leaves) but not in internal nodes. A trie can be seen as a Deterministic Finite Automaton.

Tries can be compacted. To get a compact trie from a trie, internal nodes with exactly one successor are removed. Then labels of edges between remaining nodes are concatenated. Thus:

- Edges are labeled by strings.
- Internal nodes have at least two children.
- Edges outgoing an internal node are labeled by strings starting with different symbols.

Historical Background

Tries were first recommended by de la Briandais [1]. The word "trie" comes from information re*trie*val and was suggested by Fredkin [3]. Tries enable to store and retrieve information that consists of key-element pairs. Fredkin suggested, in 1960, that it is an alternative way of storage to unordered lists, ordered lists or pigeonholes. The reader can refer to [6] for further details on tries.
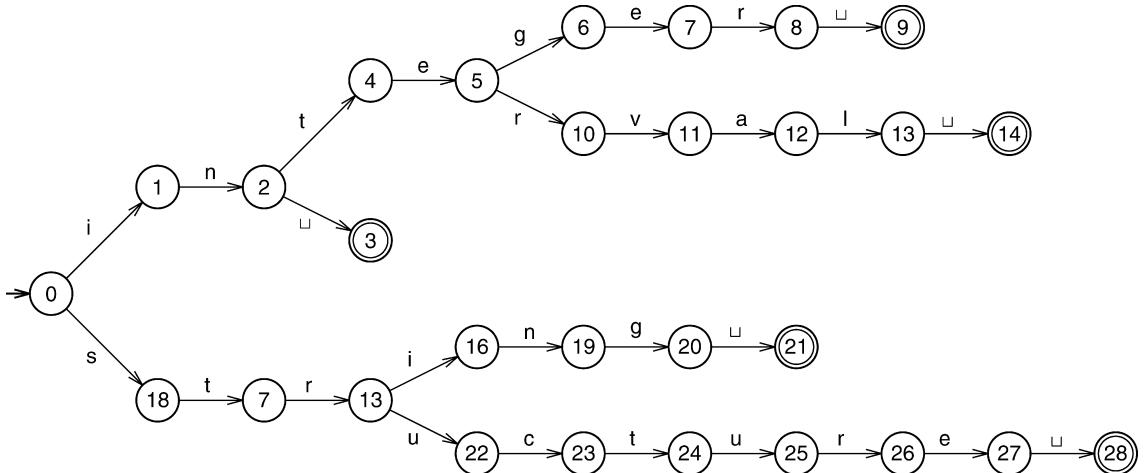
Foundations

In binary search trees, keys are stored in all the nodes of the tree and the search method is based on comparison between keys. In tries, keys are stored in the leaves and the search method involves left-to-right comparison of prefixes of the keys.

The trie of the set of strings $X =$ {in, integer, interval, string, structure} is presented Fig. 1. Note that the space sign ⊔ has been added at the end of each of the strings of X so that no string of X is a prefix of another string of X. Then each string of X is associated with a leaf of the trie (not with an internal node). The trie for the set X can be implemented in linear space with respect to the total length of the strings in X. The compact trie for the set X can be implemented in linear space with respect to the total number of the strings in X, which dramatically reduces the size of the structure.

Consider that the strings are build over an alphabet of size σ.

Flajolet and Sedgewick [8] provide an average case analysis of tries.

Trie. Figure 1. Trie of X = {in, integer, interval, string, structure}.

Construction

The algorithm TRIE(X) shown in Fig. 2, builds the trie containing all the strings in the set X. It uses a function TARGET(p, a) that gives the successor of a node p for a symbol a or the value special NIL if such a node does not exist. It works by inserting all the strings of X successively in the trie starting with the tree consisting with a single node (the root). Then for each string $x \in X$ it spells the longest prefix of x corresponding to an existing path from the root of the trie. When this longest prefix is found, it creates the nodes and the edges of the remaining suffix of x.

The sum of the lengths of all the strings of X is denoted by M. Then, the algorithm TRIE(X) run in time $O(M)$ when the branching time from a node with a given symbol is constant or $O(M \times \log \sigma)$ when the branching time depends on the alphabet size (see Implementation below).

Searching

The algorithm ISINTRIE($root$, x), see Fig. 3, tests if the string x is present in the trie and consequently if the string x is a prefix of strings represented by the trie. It works, similarly to the creation of the trie, by spelling, from the root of the trie, the longest prefix of x corresponding to a branch in the trie. If this longest prefix is x itself, then the algorithm returns TRUE and the string x belongs to the trie, otherwise the algorithm returns FALSE and the string is not a prefix of any string in the set. The algorithm ISINTRIE($root$, x) works in time $O(|x|)$ or $O(|x| \times \log \sigma)$ depending on the branching time.

```
TRIE(X)
  1  root ← new node
  2  for each string x ∈ X do
  3      p ← root
  4      for i ← 0 to |x| − 1 do
  5          q ← TARGET (p, x[i])
  6          if q = NIL then
  7              q ← new node
  8              Succ[p] ← Succ[p] ∪ {(X[i],q)}
  9          p ← q
 10      info[p] ← x
 11  return root
```

Trie. Figure 2. Algorithm that builds the trie containing of the string of a set X.

```
ISINTRIE(p,x)
  1  if x is empty then
  2      return TRUE
  3  else q ← TARGET(p,x[0])
  4      if q = NIL then
  5          return FALSE
  6  else return ISINTRIE(q,x[1..|x| − 1])
```

Trie. Figure 3. Algorithm that enables to test if a string x belongs to a trie.

Sorting

A trie can be used to sort a set of strings by doing the following: insert all the strings in the trie and output them in lexicographically increasing order by applying a pre-order traversal of the trie (respectively lexicographically decreasing order by applying a post-order traversal of the trie).

Implementation

There exist different possible representations of a trie, each with different branching times. It is possible to use a transition table whose size is the product of the number of nodes times the size of the alphabet. A trie can then be implemented in linear space with respect to the total length of all the strings it represents. The branching time, for finding the successor of a given node with a given symbol, is then constant.

The use of adjacency lists has the advantage to minimize the required space but the branching time depends on the alphabet size. It can be as low as $\log(\sigma)$ if the alphabet is ordered and outgoing edges are stored in a balanced search tree. A representation by binary tree is achieved by using a "first child – right sibling" representation of the trie.

Hashing techniques can be used as a good trade-off between transition tables and adjacency lists. The branching time is then constant on average.

A mixed technique known as the "deep shallow" technique consists of representing the nodes up to a certain level k with a transition table and to use adjacency lists for the nodes with level greater than k. Most of the times nodes with small level have many successors while nodes with large level have only one successor.
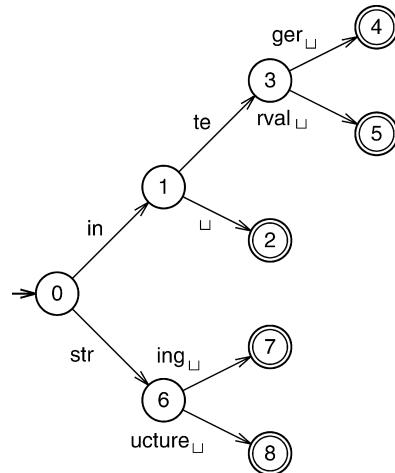
However when storage space is an issue and tries do not entirely fit into the central memory, it is possible to use compact tries as described below. Accesses to individual edges are only a bit more involved than in noncompact tries.

Compact Tries

The compact trie of the set of strings $X = \{in, integer, interval, string, structure\}$ is presented Fig. 4. It is obtained from the trie of Fig. 1 by removing internal nodes with exactly one successor. Then labels of edges between remaining nodes are concatenated.

Patricia Trees

Morrison [7] designed specific compact tries known as PATRICIA trees. PATRICIA trees are efficient especially for extremely long variable-length strings. The PATRICIA tree of the set of strings $X = \{in_\sqcup, integer_\sqcup, interval_\sqcup, string_\sqcup, structure_\sqcup\}$ is presented Fig. 5. PATRICIA trees are binary trees that consider the binary encoding of the strings. In Fig. 5 nodes 0, 1 and 3 actually point to $in_\sqcup$, nodes 4 and 7 point to $integer_\sqcup$, node 8 points to $interval_\sqcup$,

Trie. Figure 4. Compact trie of $X = \{in_\sqcup, integer_\sqcup, interval_\sqcup, string_\sqcup, structure_\sqcup\}$.

nodes 2 and 5 point to $string_\sqcup$ and node 6 points to $structure_\sqcup$. Small numbers close to the nodes are skip numbers, they indicate on which bit the branching has to be decided. The skip number of node 0 is 1: it corresponds to bit at position 1 (bolded in columns 1, 18, 26, and 34 on Fig. 5), which is the leftmost difference in the bit strings representing the strings of X. The three strings $in_\sqcup$, $integer_\sqcup$ and $interval_\sqcup$ possess a 0 so they are on the left and $string_\sqcup$ and $structure_\sqcup$ possess a 1 so they are on the right. The skip number of node 1 is 18: it corresponds to bit at position 18 (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the three strings $in_\sqcup$, $integer_\sqcup$ and $interval_\sqcup$. The string $in_\sqcup$ has a 0 so it is on the left and $integer_\sqcup$ and $interval_\sqcup$ possess a 1 so they are on the right. The skip number of node 2 is 26: it corresponds to bit at position 26, (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the two strings $string_\sqcup$ and $structure_\sqcup$. The string $string_\sqcup$ has a 0 so it is on the left and $structure_\sqcup$ possesses a 1 so it is on the right. The skip number of node 4 is 34: it corresponds to bit at position 34 (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the two strings $integer_\sqcup$ and $interval_\sqcup$. The string $interval_\sqcup$ has a 0 so it is on the left and $integer_\sqcup$ possesses a 1 so it is on the right.

The skip number of a node (different from the root) in a PATRICIA tree is never larger than its parent skip number.
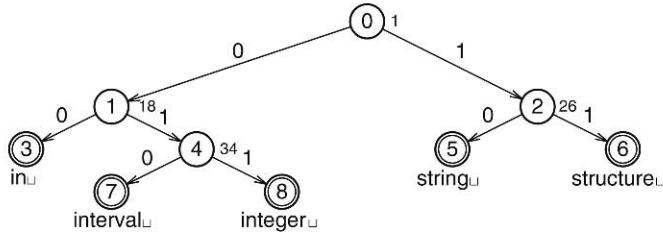
Decimal and binary ASCII codes of the symbols

		7 6 5 4 3 2 1 0				7 6 5 4 3 2 1 0				7 6 5 4 3 2 1 0
␣	32	0 0 1 0 0 0 0 0	i	105	0 1 1 0 1 0 0 1			
a	97	0 1 1 0 0 0 0 1	l	108	0 1 1 0 1 1 0 0	t	116	0 1 1 1 0 1 0 0
c	99	0 1 1 0 0 0 1 1	n	110	0 1 1 0 1 1 1 0	u	117	0 1 1 1 0 1 0 1
e	101	0 1 1 0 0 1 0 1	r	114	0 1 1 1 0 0 1 0	v	118	0 1 1 1 0 1 1 0
g	103	0 1 1 0 0 1 1 1	s	115	0 1 1 1 0 0 1 1			

Strings of *X* as sequences of bits

	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31	32 33 34 35
in␣	1 0 0 1 0 1 1 0	0 1 1 1 0 1 1 0	0 0 0 0 0 1 0 0		
integer␣	1 0 0 1 0 1 1 0	0 1 1 1 0 1 1 0	0 0 1 0 1 1 1 0	1 0 1 0 0 1 1 0	0 1 1 0 ···
interval␣	1 0 0 1 0 1 1 0	0 1 1 1 0 1 1 0	0 0 1 0 1 1 1 0	1 0 1 0 0 1 1 0	0 1 0 0 ···
string␣	1 1 0 0 1 1 1 0	0 0 1 0 1 1 1 0	0 1 0 0 1 1 1 0	1 0 0 1 0 1 1 0	0 1 1 1 ···
structure␣	1 1 0 0 1 1 1 0	0 0 1 0 1 1 1 0	0 1 0 0 1 1 1 0	1 0 1 0 1 1 1 0	1 1 0 0 ···

PATRICIA tree of *X*

Trie. Figure 5. PATRICIA tree of *X* = {in␣, integer␣, interval␣, string␣, structure␣}.

The reader can refer to [4] for further details on PATRICA trees.

Key Applications

Tries are used for dictionary representation including spell checking systems or predictive text for mobile phones (T9 system for instance). They are used for text compression (Ziv and Lempel compression schemes), multiple string matching (or multi key search). They are at the basis of the Burstsort for sorting large sets of strings. Also, it should be mentioned that when *X* is the set of suffixes of a string, the structures that are then called Suffix Trie or Suffix Tree are intensively used for Pattern Matching [2,5].

Cross-references

▶ Data Dictionary
▶ Suffix Tree
▶ Text Compression

Recommended Reading

1. de la Briandais R., File searching using variable length keys. In Proc. Western Joint Computer Conference., 1959, pp. 295–298.
2. Crochemore M., Hancart C., and Lecroq T. Algorithms on Strings. Cambridge University Press, Cambridge, 2007.
3. Fredkin E. Trie memory. Commun. ACM, 3(9):490–499, 1960.
4. Gonnet G.H. and Baeza-Yates R. Handbook of Algorithms and Data Structures – In Pascal and C, 2nd edn. Addison-Wesley, 1991.
5. Gusfield D. Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge, 1997.
6. Knuth D.E. The Art of Computer Programming, Volume 3: Sorting and Searching, 3rd edn. Addison-Wesley, 1997, Section 6.3: Digital Searching, pp. 492–512.
7. Morrison D.R. PATRICIA – Practical Algorithm to Retrieve Information Coded in Alphanumeric, J. ACM, 15(4):514–534, 1968.
8. Sedgewick R. and Flajolet Ph. An Introduction to the Analysis of Algorithms, Addison-Wesley, 1996.

Triggers

▶ Database Trigger
▶ ECA Rules

True Answer (Maybe Answer)

▶ Certain (and Possible) Answers

Trust and Reputation in Peer-to-Peer Systems

Zoran Despotovic
NTT DoCoMo Communications Laboratories Europe, Munich, Germany

Synonyms

Feedback systems; Word of mouth

Definition

Trust means reliance on something or someone's action. As such, it involves risks on the side of the subject of trust, i.e., trustor. Reducing these risks is the main goal of a trust management system. A possible way to do this is through reputation management, i.e., reputation systems.

In a typical large scale online setting, be it on the Web or in P2P networks, it is necessary to learn more about prospective transaction partners prior to engaging in a transaction with them. The size of such systems makes it highly improbable to meet the same partner repeatedly, so own experience is of little use. The types of the performed transactions are often such that well-established forms of quality assurance (e.g., contracts) are highly inefficient. Under such circumstances, reputation systems ("word of mouth" [3]) turn out to be the only viable mechanism to encourage trustworthy behavior and guide people to decide whom to trust and to what degree. They do this through collecting, distributing, and aggregating feedback about the participants past behavior," as Resnick et al. [8] explain. The key presumptions of a reputation system are that the participants of the considered online community engage in repeated interactions and that the information about their past doings is informative of their future performance and as such will influence it. Thus, collecting, processing, and disseminating the feedback about the participants' past behavior is expected to boost their trustworthiness.

Historical Background
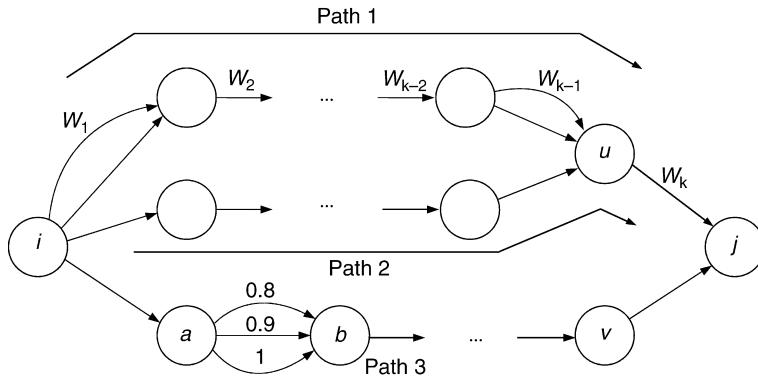
The concept of reputation is almost as old as human society. It was present in ancient times as a key enabler of trade and a broad range of other activities [3]. In the millennia to come, a new form of interaction between interested parties emerged, involving contractual agreements. When necessary, they are enforced by third parties, be it a local feudal sovereign in the medieval time or state, as nowadays. But the need for reputation did not disappear. Quite often, it is not possible to foresee what can go wrong, so that it can be specified in a contract. More important, binding contracts incur transaction costs which sometimes offset the prospective benefits from the interaction. In such cases, people resort to streamlining their interactions in informal ways, one of them being the use of reputation.

Reputation has long been a subject of study in economics. Economists find it vital. Many markets would not exist or they would be highly inefficient without reputation. Consider a market in which sellers sell goods of different qualities. Buyers cannot observe the quality of any good. Thus, they tend to undervalue high quality goods, as they may end up purchasing low quality. But then high quality sellers cannot achieve good prices and may withdraw from the market. So only low quality goods will be traded. This is what George Akerlof calls the "market for lemons" [1]. Information asymmetry between sellers and buyers is critical here. A mechanism to break it is needed. Reputation systems may be such a mechanism.

Today's perception of the P2P systems are old about eight years as of this writing. But that is quite enough for a turbulent history, which demonstrated the need for reputation management, among others. There were numerous reports of viruses spreading through well known P2P file sharing applications such as Gnutella or Kazaa. A quick inspection in Google reveals millions of entries returned to the query (Gnutella OR Kazaa) AND virus. As an example, Wired reports as of September 2004 that "forty-five percent of the executable files downloaded through Kazaa, the most popular file-sharing program, contain malicious code like viruses and Trojan horses, according to a new study."

Although these examples illustrate the point, they are benign in the sense that one can select different ways to download content, e.g., through well known web sites. But the P2P paradigm does not coincide with simple file swapping. It aims at making a more serious impact on the online world through offering a range of useful applications. No matter what these applications are, they will need reputation management. The reason is that P2P applications must be implemented through cooperation of their users, which can be unskilled or dishonest.

Trust and Reputation in Peer-to-Peer Systems. Figure 1. A trust graph.

Foundations

The core of any P2P reputation system is how it solves the following problem: how can a peer use the information on experiences between the peers to evaluate the trustworthiness of any other peer? A possible strategy might be as follows. The peer (call it also trustor) can ask its friends to report on their experiences with the unknown peer (trustee). However, the friends might not have any experience with the peer in question. This happens quite frequently in large scale systems, in which virtually every interaction is with a new peer, not seen before. As a result, the peer may search for some other unknown peers which happened to interact with the trustee. Their opinion might help. However, they might lie for whatever reason. So the problem is now how to assess the credibility of their reports. A possible solution is to continue with the search, this time looking for peers who interacted with the feedback providers and so on until enough peers are found with whom the trustor had enough experiences so that it knows their credibility. The whole process is depicted in Figure 1. The figure shows a set of peers, which are shown as vertices in the graph. The arcs represent the interactions among them, i.e., services they provide to each other. For example, peer b provided three services to peer a. The weights next to the arcs represent the level of satisfaction of the service consumer with the provided service. The structure that is formed in this way is called a trust graph.

There are two classes of reputation systems: *signaling and sanctioning reputation systems* [3]. They make different assumptions on the underlying behavior and also use different amounts of available reputation data,

i.e., different fractions of the trust graph. In a signaling reputation system, the interacting entities are presented with signals of what can go wrong in the interactions if they behave in specific ways. Having appropriate signals, the entities should decide what behavior is most appropriate for them. An important assumption of the signaling reputation systems is that the involved entities do not change their behavior in response to a change of their reputation. As an example, the system may just provide a prospective buyer with indications of the probability that the seller will fail to deliver a purchased item. This probability is the main property of the seller. It can change with time, but independently of the seller's reputation.

The other possibility is *sanctioning reputation systems*. The main assumption they make is that the involved entities are aware of the effect the reputation has on their benefits and thus adjust their behavior dynamically as their reputation changes. The main task of a reputation system in this case is to *sanction* misbehavior through providing correlation between the feedback the agent receives and the long-run profit made. The distinction between signaling and sanctioning reputation systems is made explicit in the following discussion.

A typical signaling approach involves the following three-step procedure: (i) enumerating all paths from the trustor to the trustee, (ii) aggregating the feedback along the paths and (iii) merging the obtained values. There is a nice theory developed on this subject. It is due to Richardson et al. [9]. Consider a trust graph and assume that there is only one directed arc from i to j, for any pair of vertices i and j. Multiple arcs have been merged somehow. It is not important

how exactly this merging is done. Consider the matrix $M \equiv [M_{ij}]_{i,j=1}^{N}$ (N is the number of the peers) corresponding to the trust graph and assume that it has been normalized so that for any $1 \leq i, j \leq N$: $0 \leq M_{ij} \leq 1$ and $\sum_{k=1}^{N} M_{ik} = 1$.

Define two binary operators: trust concatenation, the symbol $\circ$ will be used to denote it, and trust aggregation, denoted by $\diamond$. The former is applied on two consecutive edges in a chain of trust, while the latter applies to two chains. Simple multiplication and addition are good examples of these operators. Define now a matrix "multiplication" operation $\bullet$ as $C = A \bullet B$ such that $C_{ij} = \diamond(\forall k : A_{ik} \circ B_{kj})$. If $A = B \equiv M$, where M is the matrix representation of a given trust graph, then the interpretation of C_{ij} is aggregated trust that i puts on j over all chains of length 2. Again, if $\circ$ and $\diamond$ are ordinary multiplication and addition then $\bullet$ becomes the ordinary matrix multiplication. This is what [6] proposes.

Now, the most interesting result is that if $\diamond$ is commutative and associative and $\circ$ is associative and distributes over $\diamond$ then the aggregated value of all paths (of any length) between any pair of users can be obtained by the following simple algorithm:

$$Q^{(0)} = M, Q^{(k)} = M \bullet Q^{(k-1)} \text{ until } Q^{(k)} = Q^{(k-1)}. \quad (1)$$

The computation will converge after a finite number of steps if the matrix M (or the trust graph) is irreducible and aperiodic. It is important to see that the computation can be performed locally, after each iteration k all the peers can retrieve from their neighbors the currently computed opinions of those neighbors about all other peers and then do the computation of the step $k + 1$. It turns out that this algorithm requires at most $O(N^3)$ computations per peer.

In a similar vein, Xiong and Liu [10] compute the trustworthiness of a given peer as the average feedback about it weighted by the trustworthiness of the feedback originators themselves. This can be expressed by the formula:

$$t_j = \sum_{e \in incoming(j)} w_e \cdot \frac{t_{source(e)}}{\sum_{f \in incoming(j)} t_{source(f)}},$$

where $incoming(j)$ is the set of all edges ending at node j, w_e is the feedback belonging to the edge e and $t_{source(e)}$ the trustworthiness of the originator of this feedback.

The formula can be computed by using an iterative computation, similar to (1).

[4] makes explicit the assumption about the peer behavior. The gain is a more efficient algorithm. There is a loss as well; the mechanism is not as robust as [10]. Assume that a peer can perform trustworthy or untrustworthy (1 or 0) in its interactions with others. More precisely, each peer's behavior is modeled as two Bernoulli random variable, i.e., each peer has innate probabilities of performing trustworthy when serving other peers (denote this probability θ_k for peer p_k) and reporting truthfully its experiences with other peers (let l_k be the probability for peer p_k). The distributions of these variables are independent across peers. Consider a peer p_j that interacted with peers $p_1,...,p_n$ and its performances in these interactions were $x_1,...,x_n$, where $x_i \in \{0,1\}$. When asked to report on peer p_j's performances witnesses $p_1, p_2,...,p_n$ may misreport. This happens with their associated misreporting probabilities. The probability of observing report y_k from peer p_k can be calculated as:

$$P[Y_k = y_k] = [l_k(1 - \theta_j) + (1 - l_k)\theta_j]^{y_k}[l_k\theta_j + (1 - l_k)(1 - \theta_j)]^{1-y_k}. \quad (2)$$

Given a sample of independent reports $y_1, y_2,...,y_n$, the likelihood function of the sample is computed:

$$L(\theta_j) = P[Y_1 = y_1] \cdot P[Y_2 = y_2] \cdots P[Y_n = y_n]. \quad (3)$$

The final step is finding θ_j that maximizes this expression, i.e., the maximum likelihood estimate of the unknown probability. To do this, one needs to know l_k's in (1.3). [4] proposes a simple method to approximate them. Peer p_i deduce them from its own performances, by comparing own performances with the reports about them. If peer p_i has sufficiently many experiences with peers $p_1, p_2,...,p_n$ as reporters of its performances then it can use them to estimate the misreporting probabilities of those peers. If not, then it can opt to estimate the misreporting probability at the level of the entire network. In this case, all l_k's have the same approximate value, denote it by l, and the maximum likelihood estimate of θ_j becomes $\frac{\bar{y}-l}{1-2l}$, where $\bar{y} = \frac{y_1+\cdots y_n}{n}$.

[10] and [6] on the one hand and [4], on the other, represent different tradeoffs between the computation efficiency and robustness.

So far, an important assumption was that the peer behavior is static. The peers were characterized

by probability distributions whose parameters never change. This is an unrealistic assumption. It is dropped in sanctioning reputation systems. As a special case of these systems, game theoretic models of reputation go to the other extreme with respect to the peer behavior. They assume that the involved agents maximize their utilities across their entire lifetime. This means that, at any time instant, the agents condition their actions by the histories of previous play, both theirs and their opponents. The concept of repeated games deals with such long term interactions. However, the game-theoretic models of reputation need one more ingredient. Players are associated with different types (normally, every type assumes a different utility function of a player). Every player knows his type but the others are uncertain about which types of their opponents they face. The game-theoretic tool for modeling such uncertainties is that of games with incomplete information (Bayesian games). Fudenberg and Tirole [5] offer an extensive introduction to the subject.

There are a number of problems with a potential application of game-theoretic reputation models to P2P systems. One deals with human behavior. There are evidences that humans do not behave as utility maximizers (or if they do, it is hard to grasp their utilities within simple economic models). The other one is related to the complexity of game-theoretic models. The problem is that feedback has to be processed by the peers themselves. This leads to enlarged strategy spaces of the peers and complicates the task substantially.

Reputation models of evolutionary game theory present another area of active research. Their assumptions are as follows. A game, most often the celebrated Prisoner's dilemma, is played repeatedly among the players. The play is divided into epochs. In each epoch the players make a choice of (repeated game) strategies. When going to the next epoch, the choice of the players' strategies is biased by their scores in the previous epoch. Hence the name "evolutionary models." Axelrod [2] performed experiments in which he found that one strategy performs particularly well for the Prisoner's dilemma game. He called it "tit-for-tat" - a- it plays at any round whatever the opponent played in the previous round (and starts with cooperation). The setting for his experiments was that in each epoch the same two players played against each other. The question is how much his result extends to setting where the opponents are matched randomly, i.e., their interactions within an epoch are one-shot rather than long-term. Ohtsuki and Ywasa [7] offer an answer in which reputation data plays a critical role. They assume that a public label is associated with each player. All players can read the label, and all players except the owner of the label are allowed to change it. When a pair of players interacts, their labels are modified according to their actions. The behavior of a player can be described with two functions: the *action function* and the *assessment function*. The action function takes the label of self and the opponent and produces the decision to either cooperate or defect. The assessment function is executed after the actions of both agents have taken place. The assessment function takes the label of self, the label of the opponent and the action of the opponent and produces the new value for the opponent's label. There are 16 possible action functions and 256 possible assessment functions. This gives 4096 possible behaviors. [7] performed a systematic experimental study of these behaviors and found 8 of them (termed "the leading eight") evolutionary stable, see Table 1.

A population of agents using one of these strategies is able to sustain cooperation and drive out of existence any small population of defectors and/or reputation liars (i.e., players that set the labels to "bad" value even though their opponent cooperated).

Interestingly, there is a remarkable similarity between tit-for-tat and the leading eight strategies. The leading eight strategies exhibit all the properties of tit-for-tat, found to be important in Axelrod's experiments.

One cannot apply the above reasoning to P2P systems directly. There is a serious problem to solve.

assessment function:					action function:			
	GG	BG	GB	BB		GG	GB	BG	BB
C	G	*	G	*		C	D	C	*
D	B	G	B	*					

Trust and Reputation in Peer-to-Peer Systems. Table 1. The "leading eight" in the evolutionary indirect reciprocity game. G and B stand for good and bad reputation labels respectively. C and D stand for cooperation and defection. The GG, GB, BG, BB encode four possible states of the labels (of the two players). The three asterisks in the assessment function can take any value, hence eight possible assessment functions are possible (The value at the asterisk in the action function is uniquely determined given the choice of one of the eight assessment functions.)

How to maintain the reputation data when there is no trusted third party to do that, i.e., how to enforce the specific reputation aggregation strategy in a decentralized system? Problems like this constitute an active research area.

Key Applications

P2P applications are provided by participating peers collaboratively. For example, in a file sharing applications, any peer may provide a file that others can download. In a publish-subscribe application, anyone can act as supplier of information that the other peers can use. At the same time, peers are not well established and reputable institutions whose trustworthiness is not questionable. Most often, they are unknown individuals hidden behind meaningless identifiers. This means that reputation management is a natural need in P2P systems. Different P2P applications have varying degrees of need for reputation management, but literally all of them need it.

Interestingly, the operation of core P2P protocols can also benefit from reputation management. When routing messages, peers can take reputation of their neighbors into account and select only those which do not drop messages. This way, reputation management improves routing performance.

Cross-references
▶ Distributed Hash Table
▶ P2P Database
▶ Peer-to-Peer System
▶ Similarity and Ranking Operations
▶ Social Networks
▶ Trust in Blogosphere

Recommended Reading

1. Akerlof G. The market for "lemons": quality uncertainty and the market mechanism. Quart. J. Econom., 84:488–500, 1970.
2. Axelrod R. The Evolution of Cooperation. Basic Books, New York, 1984.
3. Dellarocas C. The digitization of word-of-mouth: promise and challenges of online reputation systems. Manage. Sci., 49(10):1407–1424, October 2003.
4. Despotovic Z. and Aberer K. Probabilistic prediction of peers performances in P2P networks. Int. J. Eng. Appl. Artif. Intell., 18(7):771–780, Elsevier, October 2005.
5. Fudenberg D. and Tirole J. Game Theory. MIT, Cambridge, MA, USA, 1991.
6. Kamvar S., Schlosser M., and Garcia-Molina H. EigenRep: reputation management in P2P networks. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 640–651.
7. Ohtsuki H. and Iwasa Y. How should we define goodness? – reputation dynamics in indirect reciprocity. J. Theor. Biol., 231:107–120, 2004.
8. Resnick P., Zeckhauser R., Friedman E., and Kuwabara K., Reputation systems. Commun. ACM, 43(12):45–48, 2000.
9. Richardson M., Agrawal R., and Domingos P. Trust management for the semantic web. In Proc. 2nd Int. Semantic Web Conf. SanibelIsland, FL, 2003, pp. 351–368.
10. Xiong L. and Liu L. Peertrust: supporting reputation-based trust in peer-to-peer communities. IEEE Trans. Knowl. Data Eng., 16(7):843–857, 2004.

Trust in Blogosphere

Nitin Agarwal, Huan Liu
Arizona State University, Tempe, AZ, USA

Synonyms
Reputation; Relationship of reliance

Definition

Trust can be defined as the relationship of reliance between two parties or individuals. *Alice* trusts *Bob* implies *Alice*'s reliance on the actions of *Bob*, based on what they know about each other. Trust is basically prediction of an otherwise unknown decision made by a party or an individual based on the actions of another party or individual. Trust is always directional and asymmetric. *Alice* trusts *Bob* does not imply *Bob* also trusts *Alice*.

From a sociological perspective, trust is the measure of belief of one party in another's honesty, benevolence, and competence. Absence of any of these properties causes failure of trust. From a psychological perspective, trust can be defined as the ability of a party or an individual to influence the other. The more trusting someone is the more easily (s)he can be influenced.

The past several years witnessed significant changes in the interactions between the individuals and groups. Individuals flock on the Internet and engage in complex social relationships, termed as social networks. Social networking has changed the paradigm of interactions and content generation. Former information consumers are now producers (or, Prosumers). Social networking has given a humongous thrust to online communities, like Blogosphere. Blogosphere is the universe of all the blog sites which contains blog posts in

reverse chronological order. Each blog post is a discourse of an individual's opinions, ideas, thoughts on some subject matter. These could be journals of personal experiences. Nonetheless, blog posts could be easily considered as a collection of semi-structured text. This opens up many research opportunities for existing text mining techniques to be adapted for this domain. Trust is highly important in a virtual world because of its low barriers to credibility. Profiles and identities could be easily faked and trust could be compromised, leading to severely critical losses, physical and/or mental.

Historical Background

Many existing works have identified the need for handling the trust aspect in social networks. Social networks can be further divided into friendship networks and the blogosphere. In social friendship networks it is important not only to detect the influential members or experts in case of knowledge sharing communities but also to assess to what extent some of the members are recognized as experts by their colleagues in the community. This leads to the estimation of trust and reputation of these experts. Some social friendship networks like Orkut allow users to assign trust ratings as a more explicit notion of trust. Whereas some websites have an implicit notion of trust where creating a link to a person on a webpage implies some amount of business trust for the person. In other cases, Trust and reputation of experts could be typically assessed as a function of the quality of their response to other members' knowledge solicitations. Pujol et al. [5] proposed a *NodeMatching* algorithm to compute the authority or reputation of a node based on its location in the social friendship network. A node's authority depends upon the authority of the nodes that relate to this node and also on other nodes that this node relates to. The basic idea is to propagate the reputation of nodes in the social friendship network. This is very similar to the *PageRank* and *HITS* algorithm in the traditional web search. However, authors point out the differences between their algorithm and *Pagerank* and *HITS*. For *PageRank* and *HITS* the transition probability matrix and variance-covariance matrix respectively have to be known previously, unlike *NodeMatching* algorithm. This becomes infeasible for very large graphs. Moreover, *PageRank* assumes a fixed graph topology by stratifying the range of transition probability which is different in *NodeMatching*

which can automatically adapt to the topology since it depends upon the authority of the related nodes.

While Pujol et al. [5] proposed an approach to establish reputation based on the position of each member in the social friendship network [8], developed a model for reputation management based on the Dampster-Shafer theory of evidence in the wake of spurious testimonies provided by malicious members of the social friendship network. Each member of a social friendship network is called an agent. Each agent has a set of acquaintances a subset of which forms its neighbors. Each agent builds a model for its acquaintances to quantify their expertise and sociability. These models are dynamic and change based on the agent's direct interactions with the given acquaintance, interactions with agents referred to by the acquaintance, and on the ratings this acquaintance received from other agents. The authors point out a significant problem with this approach which arises if some acquaintances or other agents generate spurious ratings or exaggerate positive or negative ratings, or offer testimonies that are outright false. Yu and Singh [8] study the problem of deception using the Dampster-Shafer belief functions so as to capture uncertainty in the rankings caused by malicious agents. A variant of majority weighted function is applied to belief functions and simple deception models were studied to detect deception in the ratings.

Sabater and Sierra [6] propose a combination of reputation scores on three different dimensions. They combined reputation scores not only through social relations governed by a social friendship network, termed as social dimension but also past experiences based on individual interactions, termed as individual dimension and reputation scores based on other dimensions, termed as ontological dimension. For large social friendship networks it is not always possible to get reputation scores based on just the individual dimension, so they can use the social dimension and ontological dimension would enhance the reputation estimation by considering different contexts. The ontological dimension is very similar to the work proposed in [7], where the authors recommend collaboration in social friendship networks based on several factors. They explain the importance of context in recommending a member of social friendship network for collaboration.

In [2], authors consider those social friendship networking sites where users explicitly provide trust

ratings to other members. However, for large social friendship networks it is infeasible to assign trust ratings to each and every member so they propose an inferring mechanism which would assign binary trust ratings (trustworthy/non-trustworthy) to those who have not been assigned one. They demonstrate the use of these trust values in email filtering application domain and report encouraging results. Authors also assume three crucial properties of trust for their approach to work: transitivity, asymmetry, and personalization. These trust scores are often transitive, meaning, if Alice trusts Bob and Bob trusts Charles then Alice can trust Charles. Asymmetry says that for two people involved in a relationship, trust is not necessarily identical in both directions. This is contrary to what was proposed in [8]. They assume symmetric trust values in the social friendship network between two members. Personalization of trust means that a member could have different trust values with respect to different members. Trust of a member is absolutely a personal opinion. Consolidating the trust scores for a member might not give a reasonable estimation, so authors propose trust propagation mechanism. Authors define source as the node which is seeking trust value of another node called sink. If there is a direct edge between source and sink then the value is directly transferred, otherwise the trust value is inferred based on the source's neighbors. Source polls each of its neighbors whom it has given a positive trust rating. The neighbors also use this procedure to compute the trust rating of the sink. Hence gradually sink's trust scores propagate to the source. They demonstrate the trust rating in filtering emails with the help of a prototype TrustMail and using Enron email dataset. (http://www.cs.cmu.edu/~enron/). Guha et al. [3] proposed another trust propagation scheme in social friendship networks based on a series of matrix operations but they included the element of distrust along with the trust scores.

The works discussed above rely on one or the other form of network centrality measures (like degree centrality, closeness centrality, betweenness centrality, eigenvector centrality) to evaluate trustworthy nodes and how trust propagates in the network. Nonetheless, blog networks have very sparse trust information between different pairs of nodes. Using trust propagation approaches for such a sparse network would be highly inaccurate and unreliable. Although not much research has been published that exploits text mining to evaluate trust in Blogosphere, authors in [4] have proposed to use sentiment analysis of the text around the links to other blogs in the network. They study the link polarity and label the sentiment as "positive," "negative," or "neutral." This information mined from the blogs is coupled with Guha et al.'s [3] trust and distrust propagation approach to derive trust values between node pairs in the blog network. They further use this model to identify trustworthy nodes in the blog network and also identify clusters of like-minded blogs.

Foundations

Quantifying and computing trust in social networks is hard because concepts like trust are fuzzy, and is being expressed in a social way. The definitions and properties are not mathematical formalisms but social ones. The two main components of defining trust are belief and commitment. The extent to which someone trusts another is illustrated by the belief and trusted individuals maintain that with their commitment. Note that trust is highly subjective, nevertheless some characteristic properties are pointed:

Transitivity: Trust can propagate through different nodes following transitive property. However, the degree of trust does not remain same. It may decrease as the path length through which trust propagates increases.
Asymmetry: Trust is asymmetric, in the sense that if A trusts B then it is not necessary that B also trusts A. Some existing works relax this assumption and consider trust as symmetric.
Personalization: Trust is a personalized concept. Everyone has a different conception of trust with respect to some other individual. Assigning a global trust value to an individual is highly unrealistic. Trustworthiness of an individual is always evaluated with respect to some other individual.

Trust can be considered as binary-valued with 1 indicating trust and 0 indicating "not-trusted." Trust can also be evaluated as continuous-valued. Moreover, binary-valued trust is little more complicated than meets the eye. A value of 0 could be a little vague as it could represent both "no-opinion" or "distrust." To qualify this notion, often researchers use -1 to represent distrust and 0 as missing value or "no-opinion." Researchers model the propagation of distrust the same way as the propagation of trust. Propagation of trust (T) and distrust (D) could be governed by the set of rules illustrated in Table 1. Here *A*, *B*, and *C* are

Trust in Blogosphere. Table 1. Rules for trust and distrust propagation

Propagation Scheme	Outcome	Comments
A $\xrightarrow{T}$ B $\xrightarrow{T}$ C	A $\xrightarrow{T}$ C	*Transitivity*
A $\xrightarrow{T}$ B $\xrightarrow{D}$ C	A $\xrightarrow{D}$ C	Don't trust someone who is distrusted by a person you trust.
A $\xrightarrow{D}$ B $\xrightarrow{T}$ C	A $\xrightarrow{D}$ C	Don't trust someone who is trusted by a person you don't trust.
A $\xrightarrow{D}$ B $\xrightarrow{D}$ C	(1) A $\xrightarrow{T}$ C	Enemy of yopur enemy is your friend.
	(2) A $\xrightarrow{D}$ C	Don't trust someone who is not trusted by a person you don't trust.

different individuals and trust or distrust relationship between *A–B* and *B–C* is known. These rules help in inferring trust or distrust between *A–B*. Propagation of distrust is a little intricate. As shown in the Table 1, if *A* distrusts *B* and *B* distrusts *C* then *A* has reasons for either trusting *C* (enemy of enemy is a friend) or distrusting *C* (do not trust someone who is not trusted by someone else that is not trusted).

In case the link between *A* and *C*, like *B* is missing, which can be used to infer the trust between *A–C*, a different strategy could be used. Trust only if someone is trusted by *k* people, i.e., if *C* is trusted by a *k* number of people then *A* could trust *C*. Do not trust anyone who is distrusted by *k′* people, i.e., if *C* is distrusted by *k′* number of people then *A* could distrust *C*. Note that the thresholds *k* and *k′* could be learned from the data.

Key Applications

Trust in social networks has several applications. Trust and reputation based spam email filters have become popular after naïve spam email filters. The social network information of senders and recipients could be exploited to study trust among them and filter emails based on these values. Trust acts as lubricant that improves information flow and promotes frankness and honesty. Trust can also be helpful in online discussion forums where users look for informative and trustworthy answers to their questions. Giving trustworthy recommendations could also improve the customer retention policies.

Future Directions

Trust is a promising area of research in social networks, especially the blogosphere, where most of the assumptions from friendship networks are absent.

1. Social friendship networks assume initial trust values are assigned to the nodes of the network. Unless some social networking websites allow their members to explicitly provide trust ratings for other members, it is a topic of research and exploration to compute initial trust scores for the members. Moreover, in Blogosphere it is even harder to implicitly compute initial trust scores.

2. Social friendship networks assume an explicit relationship between members of the network. However, in Blogosphere there is no concept of explicit relationship between bloggers. Many times, these relationships have to be anticipated using link structure in the blogs or blogging characteristics of the bloggers.

3. Existing works of trust propagation algorithms assume an initial starting point. In Blogosphere, where both network structure and initial ratings are not explicitly defined, it is challenging to tackle the trust aspect. A potential approach could be to use influential members [1] of a blog community as the seeds for trustworthy nodes.

4. Since text mining has not been sufficiently exploited in Blogosphere domain, several promising research opportunities can be explored.

Data Sets

The following datasets are widely used by many researchers in this area:

A website (http://www.epinions.com/) that maintains trust values for all the available products/services provided by the customers.

Epinions: Movie Recommendation: Netflix (http://www.netflixprize.com/) provides movie recommendation dataset and what recommendations were followed by the customers. Research works have engineered this dataset to evaluate trust among customers.

Enron Email Dataset: A collection of emails that contains both genuine and spam emails. Researchers

constructed social network between senders and recipients of the email and studied trust aspect.

Cross-references

Recommended Reading

1. Agarwal N., Liu H., Tang L., and Yu P.S. Identifying the influential bloggers in a community. In Proc. Int. Conf. Web Search and Web Data Mining, 2008, pp. 207–218.
2. Golbeck J. and Hendler J. Inferring binary trust relationships in web-based social networks. ACM Trans. Inter. Tech., 6(4):497–529, 2006.
3. Guha R., Kumar R., Raghavan P., and Tomkins A. Propagation of trust and distrust. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 403–412.
4. Kale A., Karandikar A., Kolari P., Java A., Finin T., and Joshi A. Modeling trust and influence in the blogosphere using link polarity. In Proc. 1st Int.'l AAAI Conf. on Weblogs and Social Media, 2007.
5. Pujol J.M., Sangesa R., and Delgado J. Extracting reputation in multi agent systems by means of social network topology. In Proc. 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2002, pp. 467–474.
6. Sabater J. and Sierra C. Reputation and social network analysis in multi-agent systems. In Proc. 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2002, pp. 475–482.
7. Terveen L. and McDonald D.W. Social matching: a framework and research agenda. ACM Trans. Comput.-Hum. Interact., 12(3):401–434, 2005.
8. Yu B. and Singh M.P. Detecting deception in reputation management. In Proc. 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2003, pp. 73–80.

Trusted Database Systems

Trusted Hardware

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Synonyms

Tamper-proof hardware; Secure hardware

Definition

Trusted Hardware is a broad term used to denote any hardware that has been certified to perform according to a certain set of requirements. Most often however, "trusted hardware" is discussed in adversarial contexts. The term has thus been somewhat hijacked to mean "tamper-proof" hardware, i.e., hardware designed to resist direct physical access adversaries. Often trusted hardware encompasses some cryptographic abilities, i.e., performing encryption and data authentication.

Key Points

Certification. The National Institute of Standards has established a set of standards for security requirements of cryptographic modules and specifically for physical properties and tamper-resistance thereof [2]. The FIPS 140–2 Level 4 certification is at present the highest-attainable hardware security in sensitive, non-classified domains. While a plethora of devices have undergone FIPS certification, the most common types of trusted hardware in use today are TPM micro-controllers and secure CPUs: ***TPM.*** The Trusted Platform Module (TPM) specifications of the Trusted Computing Group [3] define a micro-controller that stores keys, passwords and digital certificates. In actual instances TPMs are connected to the main circuitry of computing devices (such as PC motherboards) and ensure that the stored data are secure from external software attacks. It is important to note however, that a TPM "can only act as a 'lave' to higher level services and applications by storing and reporting pre-runtime configuration information. [...]. At no time can the TCG building blocks 'control' the system or report the status of applications that are running." This passive nature limits the TPM's utility in security paradigms that require active processing.

SCPUs. Secure CPUs (SCPUs) are a term used to denote general-purpose CPUs deployed in a certified tamper-proof enclosure. Instances include the IBM 4758 PCI and the newer IBM 4764 PCI-X cryptographic coprocessors [1]. The IBM 4764 is a PowerPC-based board and runs embedded Linux. The 4758 is based on a Intel 486 architecture and is preloaded with a compact runtime environment that allows the loading of arbitrary external certified code. The CPUs can be custom programmed. Moreover, they (4758 models 2 and 23 and 4764 model 1) are compatible with the IBM Common Cryptographic Architecture (CCA) API. The CCA implements common cryptographic services

such as random number generation, key management, digital signatures, and encryption (DES/3DES, RSA). If physically attacked, the devices destroy their internal state (in a process powered by internal long-term batteries) and shut down in accordance with their FIPS 140-2 certification. It is important to note that SCPUs are generally one order of magnitude slower than main processors mainly due to heat dissipation constraints limiting the maximum allowable gate-density within the tamper-proof enclosure.

Cross-references

▶ Regulatory Compliance in Data Management

Recommended Reading

1. IBM Cryptographic Hardware. Online at http://www-03.ibm.com/security/products/, 2007.
2. NIST Federal Information Processing Standards. Online at http://csrc.nist.gov/publications/fips/, 2007.
3. Trusted Computing Group. Online at http://www.trustedcomputinggroup.org/, 2007.

TSQL2

RICHARD T. SNODGRASS
University of Arizona, Tucson, AZ, USA

Definition

TSQL2 (*Temporal Structured Query Language*) is a temporal extension of SQL-92 designed in 1993–1994 by a committee comprised of Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T. Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo and Suryanarayana M. Sripada. The goal of this language design committee was to consolidate past research on temporal query languages, by the committee members as well as many others, by developing a specification for a consensus language that could form a common core for research.

Historical Background

Temporal databases have been an active research topic since 1980. By the early 1990's, several dozen temporal query languages had been proposed, and many temporal database researchers felt that the time had come to consolidate approaches to temporal data models and calculus-based query languages to achieve a consensus query language and associated data model upon which future research could be based.

In April 1992, Richard Snodgrass circulated a white paper that proposed that a temporal extension to SQL be produced by the research community. Shortly thereafter, the temporal database community organized the ARPA/NSF International Workshop on an Infrastructure for Temporal Databases, held in Arlington Texas in June 1993 [3]. Discussions at that workshop indicated that there was substantial interest in a temporal extension to the conventional relational query language SQL-92 [2]. A general invitation was sent to the community, and about a dozen people volunteered to develop a language specification. Several people later joined the committee. The goal of this language design committee was to develop a specification for a consensus extension to SQL-92, termed the *Temporal Structured Query Language*, or TSQL2.

The group corresponded via electronic mail from early July 1993, submitting, debating, and refining proposals for the various aspects and elements of the language. In September 1993, the first draft specification, accompanied by 13 commentaries, was distributed to the committee. In December 1993, a much enlarged draft, accompanied by some 24 commentaries, was distributed to the committee. A preliminary language specification was made public in March 1994 [6], and a tutorial of the language appeared in September 1994 [7]. The final language specification and 28 commentaries were also made available via anonymous FTP in early October 2004. The final specification and commentaries appeared in a book [4] that was distributed at a temporal database workshop in summer of 1995, less than 2 years after the committee had been founded. TSQL2 is remarkable, and perhaps unique, in that it was designed entirely via electronic mail, by a committee that never met physically (in fact, no one on the committee has met every other committee member).

Work then commenced to incorporate elements and underlying insights of TSQL2 into SQL3. The first step was to propose a new part to SQL3, termed SQL/Temporal. This new part was accepted at the Ottawa meeting in January, 1995 as Part 7 of the

SQL3 specification [11]. A modification of TSQL2's PERIOD data type is included in that part.

The focus at that point changed to adding valid-time and transaction-time support to SQL/Temporal. Two change proposals, one on valid-time support and one on transaction-time support, were unanimously accepted by ANSI and forwarded to ISO [8,9]; a summary appeared shortly thereafter [10]. A comprehensive set of case studies [5] showed that while SQL-92 required 1,848 lines, only 520 lines of SQL/Temporal were required to achieve exactly the same functionality. These case studies showed that, over a wide range of data definition, query, and modification fragments, the SQL-92 version is *three* times longer in numbers of lines than the SQL/Temporal version, and many times more complex. In fact, very few SQL/Temporal statements were more than ten lines long; some statements in SQL-92 comprised literally dozens of lines of highly complex code. Due to disagreements within the ISO committee as to where temporal support in SQL should go, the project responsible for temporal support was canceled near the end of 2001. Nevertheless, concepts and constructs from SQL/Temporal have been implemented in the Oracle database management system, and other products have also included temporal support.

Oracle 9i includes support for transaction time. Its flashback queries allow an application to access prior transaction-time states of its database; they are transaction timeslice queries. Database modifications and conventional queries are temporally upward compatible. Oracle 10g extends flashback queries to retrieve all the versions of a row between two transaction times (a key-transaction-time-range query) and allows tables and databases to be rolled back to a previous transaction time, discarding all changes after that time. The Oracle 10g Workspace Manager includes the period data type, valid-time support, transaction-time support, support for bitemporal tables, and support for sequenced primary keys, sequenced uniqueness, sequenced referential integrity, and sequenced selection and projection, in a manner quite similar to that proposed in SQL/Temporal.

Foundations

The goals that underpinned the process that led to the TSQL2 language design are first considered, then the language concepts underlying TSQL2 are reviewed.

Design Goal for TSQL2

TSQL2 is a temporal query language, designed to query and manipulate time-varying data stored in a relational database. It is an upward-compatible extension of the international standard relational query language SQL-92.

The TSQL2 language design committee started their work by establishing a number of ground rules with the objective of achieving a coherent design.

- TSQL2 will be a *language* design.
- TSQL2 is to be a *relational* query language, not an object-oriented query language.
- TSQL2 should be consistent with existing standards, not another standard.
- TSQL2 should be comprehensive and should reflect areas of convergence.
- The language will have an associated algebra.

The committee enumerated the desired features of TSQL2; these guided the design of the language. The first batch concerned the data model itself.

- TSQL2 should not distinguish between snapshot equivalent instances, i.e., snapshot equivalence and identity should be synonymous.
- TSQL2 should support only one valid-time dimension.
- For simplicity, tuple timestamping should be employed.
- TSQL2 should be based on homogeneous tuples.
- Valid-time support should include support for both the past and the future.
- Timestamp values should not be limited in range or precision.

The next concerned the language proper.

- TSQL2 should be a consistent, fully upwardly compatible extension of SQL-92.
- TSQL2 should allow the restructuring of tables on any set of attributes.
- TSQL2 should allow for flexible temporal projection, but TSQL2 syntax should reveal clearly when non-standard temporal projections are being done.
- Operations in TSQL2 should not accord any explicit attributes special semantics. For example, operations should not rely on the notion of a key.

T

- Temporal support should be optional, on a per-table basis. Tables not specified as temporal should be considered as snapshot tables. It is important to be an extension of SQL-92's data model when possible, not a replacement. Hence, the schema definition language should allow the definition of snapshot tables. Similarly, it should be possible to derive a snapshot table from a temporal table.
- User-defined time support should include instants, periods, and intervals.
- Existing aggregates should have temporal analogues in TSQL2.
- Multiple calendar and multiple language support should be present in timestamp input and output, and timestamp operations. SQL-92 supports only one calendar, a particular variant of the Gregorian calendar, and one time format. The many uses of temporal databases demand much more flexibility.
- It should be possible to derive temporal and non-temporal tables from underlying temporal and non-temporal tables.

Finally, the committee agreed upon three features aimed at ease of implementation.

- TSQL2 tables should be implementable in terms of conventional first normal form tables. In particular, the language should be implementable via a data model that employs period-timestamped tuples. This is the most straightforward representation, in terms of extending current relational technology.
- TSQL2 must have an efficiently implementable algebra that allows for optimization and that is an extension of the conventional relational algebra, on which current DBMS implementations are based. The temporal algebra used with the TSQL2 temporal data model should contain temporal operators that are extensions of the operations in the relational algebra. Snapshot reducibility is also highly desired, so that, for example, optimization strategies will continue to work in the new data model.
- The language data model should *accept* implementation using other models, such as models that timestamp attribute values. The language data model should allow multiple representational data models. In particular, it would be best if the data model accommodated the major temporal data models proposed to date, including attribute timestamped models.

Language Concepts in TSQL2

The following is a brief outline of the major concepts behind TSQL2.

Time Ontology   The TSQL2 model of time is bounded on both ends. The model refrains from deciding whether time is ultimately continuous, dense, or discrete. Specifically, TSQL2 does not allow the user to ask a question that will differentiate the alternatives. Instead, the model accommodates all three alternatives by assuming that an instant on a time-line is much smaller than a chronon, which is the smallest entity that a timestamp can represent exactly (the size of a chronon is implementation-dependent). Thus, an instant can only be approximately represented. A discrete image of the represented times emerges at run-time as timestamps are scaled to user-specified (or default) granularities and as operations on those timestamps are performed to the given scale.

An instant is modeled by a timestamp coupled with an associated scale (e.g., day, year, month). A period is modeled by a pair of two instants in the same scale, with the constraint that the instant timestamp that starts the period equals or precedes (in the given scale) the instant timestamp that terminates the period.

Base Line Clock   A semantics must be given to each time that is stored in the database. SQL-92 specifies that times are given in UTC seconds, which are, however, not defined before 1958, and in any case cannot be used to date prehistoric time, as UTC is based in part on solar time. TSQL2 includes the concept of a *baseline clock*, which provides the semantics of timestamps. The baseline clock relates each second to physical phenomena and partitions the time line into a set of contiguous *periods*. Each period runs on a different clock. *Synchronization points* delimit period boundaries. The baseline clock and its representation are independent of any calendar.

Data Types   SQL-92's datetime and interval data types are augmented with a *period* datetime, of specifiable range and precision. The range and precision can be expressed as an integer (e.g., a precision of 3 fractional digits) or as an interval (e.g., a precision of a week). Operators are available to compare timestamps and to compute new timestamps, with a user-specified precision. Temporal values can be input and output

in user-specifiable formats, in a variety of natural languages. *Calendars* and *calendric systems* permit the application-dependent semantics of time to be incorporated.

A surrogate data are introduced in TSQL2. Surrogates are unique identifiers that can be compared for equality, but the values of which cannot be seen by the users. In this sense, a surrogate is "pure" identity and does not describe a property (i.e., it has no observable value). Surrogates are useful in tying together representations of multiple temporal states of the same object; they are not a replacement for keys.

Time-Lines   Three time-lines are supported in TSQL2: user-defined time, valid time, and transaction time. Hence values from disparate time-lines can be compared, at an appropriate precision. Transaction-time is bounded by `initiation`, the time when the database was created, and until changed. In addition, user-defined and valid time have two special values, beginning and forever, which are the least and greatest values in the ordering. Transaction time has the special value until changed.

Valid and user-defined times can be indeterminate. In *temporal indeterminacy*, it is known that an event stored in a temporal database did in fact occur, but it is not known exactly *when* that event occurred. An instant (interval, period) can be specified as determinate or indeterminate; if the latter then the possible mass functions, as well as the generality of the indeterminacy to be represented, can be specified. The quality of the underlying data (termed its *credibility*) and the *plausibility* of the ordering predicates expressed in the query can be controlled on a per-query or global basis.

Finally, instant timestamps can be *now-relative*. A now-relative time of "*now* – 1 day," interpreted when the query was executed on June 12, 1993, would have the *bound* value of "June 11, 1993." The users can specify whether values to be stored in the database are to be bound (i.e., not now-relative) or unbound.

Aggregates   The conventional SQL-92 aggregates are extended to take into account change across time. They are extended to return time-varying values and to permit grouping via a partitioning of the underlying time line, termed *temporal grouping*. Values can be *weighted* by their duration during the computation of

an aggregate. Finally, a new temporal aggregate, `RISING`, is added. A taxonomy of temporal aggregates [4, Chap. 21] identifies 14 possible kinds of aggregates; there are instances of all of these kinds in TSQL2.

Valid-Time Tables   The snapshot tables supported by SQL-92 continue to be available in TSQL2, which, in addition, supports *state* tables, where each tuple is timestamped with a *temporal element* that is a union of periods. As an example, the Employee table with attributes Name, Salary, and Manager could contain the tuple (Tony, 10,000, LeeAnn). The temporal element timestamp would record the maximal (non-contiguous) periods in which Tony made $10,000 and had LeeAnn as his manager. Information about other values of Tony's salary or other managers would be stored in other tuples. The timestamp is implicitly associated with each tuple; it is not another column in the table. The range, precision, and indeterminacy of a temporal element can be specified.

Temporal elements are closed under union, difference, and intersection. Timestamping tuples with temporal elements is conceptually appealing and can support multiple representational data models. Dependency theory can be extended to apply in full to this temporal data model.

TSQL2 also supports *event* tables, in which each tuple is timestamped with an *instant set*. As an example, a Hired table with attributes Name and Position could contain the tuple (LeeAnn, Manager). The instant set timestamp would record the instant(s) when LeeAnn was hired as a Manager. (Other information about her positions would be stored in separate tables.) As for state tables, the timestamps are associated implicitly with tuples.

Transaction-Time and Bitemporal Tables   Orthogonally to valid time, transaction time can be associated with tables. The transaction time of a tuple, which is a temporal element, specifies when that tuple was considered to be part of the current database state. If the tuple (Tony, 10,000, LeeAnn) was stored in the database on March 15, 1992 (say, with an `APPEND` statement) and removed from the database on June 1, 1992 (say, with a `DELETE` statement), then the transaction time of that tuple would be the period from March 15, 1992 to June 1, 1992.

Transaction timestamps have an implementation-dependent range and precision, and they are determinate.

In summary, there are six kinds of tables: snapshot (no temporal support beyond user-defined time), valid-time state tables (timestamped with valid-time elements), valid-time event tables (timestamped with valid-time instant sets), transaction-time tables (timestamped with transaction-time elements), bitemporal state tables (timestamped with bitemporal elements), and bitemporal event tables (timestamped with bitemporal instant sets).

Schema Specification   The CREATE TABLE and ALTER statements allow specification of the valid- and transaction-time aspects of temporal tables. The scale and precision of the valid timestamps can also be specified and later altered.

Restructuring   The FROM clause in TSQL2 allows tables to be *restructured* so that the temporal elements associated with tuples with identical values on a subset of the columns are coalesced. For example, to determine when Tony made a Salary of $10,000, independent of who his manager was, the Employee table could be restructured on the Name and Salary columns. The timestamp of this restructured tuple would specify the periods when Tony made $10,000, information which might be gathered from several underlying tuples specifying different managers.

Similarly, to determine when Tony had LeeAnn as his manager, independent of his salary, the table would be restructured on the Name and Manager columns. To determine when Tony was an employee, independent of how much he made or who his manager was, the table could be restructured on only the Name column.

Restructuring can also involve *partitioning* of the temporal element or instant set into its constituent maximal periods or instants, respectively. Many queries refer to a continuous property, in which maximal periods are relevant.

Temporal Selection   The valid-time timestamp of a table may participate in predicates in the WHERE clause by via VALID() applied to the table (or correlation variable) name. The transaction-time of a table can be accessed via TRANSACTION(). The operators have been extended to take temporal elements and instant sets as arguments.

Temporal Projection   Conventional snapshot tables, as well as valid-time tables, can be derived from underlying snapshot or valid-time tables. An optional VALID or VALID INTERSECT clause is used to specify the timestamp of the derived tuple. The transaction time of an appended or modified tuple is supplied by the DBMS.

Update   The update statements have been extended in a manner similar to the SELECT statement, to specify the temporal extent of the update.

Cursors   Cursors have been extended to optionally return the valid time of the retrieved tuple.

Schema Versioning   Schema *evolution*, where the schema may change, is already supported in SQL-92. However, old schemas are discarded; the data are always consistent with the current schema. Transaction time support dictates that previous schemas be accessible, which calls for *schema versioning*. TSQL2 supports a minimal level of schema versioning.

Vacuuming   Updates, including (*logical*) deletions, to transaction time tables result in insertions at the physical level. Despite the continuing decrease in cost of data storage, it is still, for various reasons, not always acceptable that all data be retained forever. TSQL2 supports a simple form of *vacuuming*, i.e., physical deletion, from such tables.

System Tables   The TABLE base table has been extended to include information on the valid and transaction time components (if present) of a table. Two other base tables have been added to the definition schema.

SQL-92 Compatibility   All aspects of TSQL2 are pure extensions of SQL-92. The user-defined time in TSQL2 is a consistent replacement for that of SQL-92. This was done to permit support of multiple calendars and literal representations. Legacy applications can be supported through a default SQL92_calendric_system.

The defaults for the new clauses used to support temporal tables were designed to satisfy snapshot reducibility, thereby ensuring that these extensions constitute a strict superset of SQL-92.

Implementation During the design of the language, considerable effort was expended to ensure that the language could be implemented with only moderate modification to a conventional, SQL-92-compliant DBMS. In particular, an algebra has been demonstrated that can be implemented in terms of a period-stamped (or instant-stamped, for event tables) tuple representational model; few extensions to the conventional algebra were required to fully support the TSQL2 constructs. This algebra is snapshot reducible to the conventional relational algebra. Support for multiple calendars, multiple languages, mixed precision, and indeterminacy have been included in prototypes that demonstrated that these extensions have little deleterious effect on execution performance. Mappings from the data model underlying TSQL2, the bitemporal conceptual data model [1], to various representational data models have been defined [4].

Key Applications

TSQL2 continues to offer a common core for temporal database research, as well as a springboard for change proposals for extensions to the SQL standard.

Future Directions

Given the dramatic decrease in code size and complexity for temporal applications that TSQL2 and SQL/Temporal offers, it is hoped that other DBMS vendors will take Oracle's lead and incorporate these proposed language constructs into their products.

Url to Code

http://www.cs.arizona.edu/people/rts/tsql2.html   This web page includes links to the ISO documents. http://www.sigmod.org/dblp/db/books/collections/snodgrass95.html

Cross-references

▶ Applicability Period
▶ Fixed Time Span
▶ Now in Temporal Databases
▶ Period-Stamped Temporal Models
▶ Span
▶ Schema Versioning
▶ Temporal Aggregation
▶ Temporal Algebras
▶ Temporal Compatibility
▶ Temporal Integrity Constraints
▶ Temporal Joins

▶ Temporal Logical Models
▶ Temporal Query Languages
▶ Temporal Vacuuming
▶ Time-Line Clock
▶ Transaction Time
▶ TUC
▶ Until Changed
▶ Valid Time
▶ Value Equivalence

Recommended Reading

1. Jensen C.S., Soo M.D. and Snodgrass R. T. Unifying Temporal Data Models via a Conceptual Model. Inf. Syst., 19(7):513–547, December 1994.
2. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, 1993.
3. Snodgrass R.T. (ed.). In Proc. Int. Workshop on an Infrastructure for Temporal Databases, 1993.
4. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer Academic, 1995.
5. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, July 1999.
6. Snodgrass R.T., Ahn I., Ariav G., Batory D.S., Clifford J., Dyreson C.E., Elmasri R., Grandi F., Jensen C.S., Käfer W., Kline N., Kulkarni K., Leung T.Y.C., Lorentzos N., Roddick J.F., Segev A., Soo M.D., and Sripada S.M., TSQL2 Language Specification. ACM SIGMOD Rec., 23(1):65–86, March 1994.
7. Snodgrass R.T., Ahn I., Ariav G., Batory D., Clifford J., Dyreson C.E., Elmasri R., Grandi F., Jensen C.S., Käfer W., Kline N., Kulkarni K., Leung T.Y.C., Lorentzos N., Roddick J.F., Segev A., Soo M.D., and Sripada S.M. A TSQL2 tutorial. ACM SIGMOD Rec., 23(3):27–33, September 1994.
8. Snodgrass R.T., Böhlen M.H., Jensen C.S. and Steiner A. Adding Transaction Time to SQL/Temporal. Change proposal, ANSI X3H2-96-502r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-147r2, November 1996.
9. Snodgrass R.T., Böhlen M.H., Jensen C.S. and Steiner A. Adding Valid Time to SQL/Temporal. change proposal, ANSI X3H2-96-501r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-146r2, November 1996.
10. Snodgrass R.T., Böhlen M.H., Jensen C.S., and Steiner A., Transitioning Temporal Support in TSQL2 to SQL3. In Temporal Databases: Research and Practice, O. E.zion, S. Jajodia, S.M. Sripada (eds.). Springer, Berlin, 1998, pp. 150–194.
11. Snodgrass R.T., Kulkarni K., Kucera H., and Mattos N. Proposal for a new SQL Part – Temporal. ISO/IEC JTC1/SC21 WG3 DBL RIO-75, X3H2-94-481, November 2, 1994.

Tug-of-War Sketch

▶ AMS Sketch

Tuning Concurrency Control

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Synonyms

Lock tuning

Definition

Database systems implement concurrency control to give users the illusion that each transaction executes correctly, in isolation from all others. The concurrency-control algorithm in predominant use is two-phase locking. Tuning concurrency control consists in improving the performance of concurrent operations by reducing the number, duration and scope of the conflicts due to locking.

Historical Background

In 1976, Jim Gray et al. identified the fundamental concurrency control trade-off between correctness and performance. They discussed different lock granularities and introduced the notion of degrees of consistency.

Foundations

Database systems attempt to give users the illusion that each transaction executes in isolation from all others. The ANSI SQL standard, for example, makes this explicit with its concept of degrees of isolation. Full isolation or *serializability* is the guarantee that each transaction that completes will appear to execute one at a time, except that its performance may be affected by other transactions. Choosing a lower level of isolation will benefit performance, possibly at the cost of correctness. The value of serializability experiment (see below in experimental results) illustrates this performance/correctness trade-off. This entry discusses basic concurrency tuning techniques.

Leveraging Application Semantics

Efficient tuning often entails understanding application semantics. A frequently required feature is to assign consecutive key values to consecutive records (e.g., customer numbers, purchase order numbers). Consider a straightforward implementation.

In the following example, the COUNTER table contains the next value which is used as a key when inserting values in the ACCOUNT table.

```
begin transaction
    NextKey:=select nextkey from COUNTER;
    insert into ACCOUNT values (nextkey, 100, 200);
    update COUNTER set nextkey=NextKey+1;
end transaction
```

When the number of such transactions issued concurrently increases, COUNTER becomes a bottleneck because all transactions read and write the value of nextkey.

An alternative approach is to use the facilities that many systems offer that reduce the length of time counter locks are held. These facilities (sequences in Oracle, autoincrement in MySQL and identity in SQL Server, DB2 UDB and Sybase Adaptive Server) enable transactions to hold a latch (see the latch definitional entry) on the counter only while accessing the counter, rather than until the transaction completes. This eliminates the counter as a bottleneck but may introduce a small problem.

Consider an insert transaction T that increments the counter then aborts. Before T aborts, a second transaction T' may increment the counter further. Thus, the counter value obtained by T will not be associated with any data item. That is, there may be gaps in the counter values. Most applications can tolerate such gaps, but some cannot for legal reasons, e.g., tax authorities prefer that invoice numbers have no gaps.

Living Dangerously

Many applications live with less than full isolation due to the high cost of holding locks during user interactions. Consider the following full-isolation transaction from an airline reservation application:

Airline Reservation Transaction
    Begin transaction
        Retrieve list of seats available;
        Reservation agent talks with customer regarding availability;
        Secure seat.
    End transaction

The performance of a system built from such transactions would be intolerably slow, because each customer would hold a lock on all available seats for a flight

while chatting with the reservations agent. This solution does, however, guarantee two conditions: (i) no two customers will be given the same seat, and (ii) any seat that the reservation agent identifies as available in view of the retrieval of seats will still be available when the customer asks to secure it.

Because of the poor performance, however, the following is done instead:

Loosely Consistent Airline Reservation
    Begin transaction
      Retrieve list of seats available;
      Reservation agent talks with customer regarding availability;
      Secure seat.
    End transaction

This design relegates lock conflicts to the secure step, thus guaranteeing that no two customers will be given the same seat. It does allow the possibility, however, that a customer will be told that a seat is available, will ask to secure it, and will then find out that it is gone.

General Rules of Thumb

By looking at blocking and (more rarely) deadlock statistics, an administrator or advanced application user can infer the existence of a concurrency control bottleneck. What should follow is careful analysis of the application to see (i) whether transactions can be redesigned to place accesses to hot items at the ends of the transactions, (ii) whether system facilities may help to reduce concurrency bottlenecks, (iii) or whether the application semantics allow a lesser form of concurrency correctness guarantee for the sake of performance. The general idea is to reduce the hold on the few critical resources that cause the concurrency bottleneck.

Key Applications

Concurrency control tuning is essential for applications having frequent modifications (inserts, deletes, and/or updates), because those applications entail lock conflicts.

Experimental Results

Value of Serializability

This experiment illustrates the correctness/performance trade-off associated to the two isolation levels, i.e.,

serializable and read committed. Consider a table of the form R(a int primary key, b int), on which an application executes two types of transactions: (i) a *sum* transaction that computes the sum of *b* values, and (ii) *swap* transactions that swap *b* values. The experiment consists in executing the two types of transactions concurrently. The parameters of the experiments are (i) the level of isolation, and (ii) the number of concurrent threads executing the swap transactions (note that the total number of transactions is kept constant throughout the experiment). Response time is measured for the total number of transactions.

The results presented below were obtained with MySQL SQL 6.0 (using InnoDB with a 1GB buffer pool), running on a Linux server equipped with an Intel Core 2 Duo processor (the cache is warm during this experiment). The SQL code and the data used for this experiment as well as the data set with the measurements are available at the URL listed at the end of this entry.
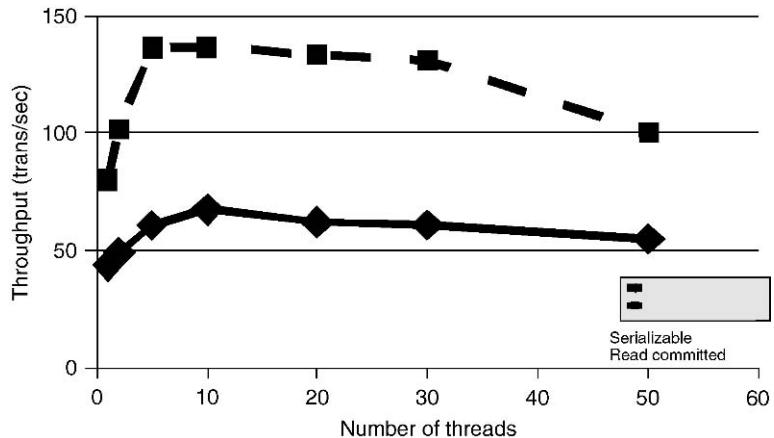
Interestingly, both the read committed and serializable isolation levels yield 100% correct answer regardless of the number of threads executing the *swap* transactions. The reason is that MySQL (like Oracle) implements snapshot isolation: the result of the *sum* transaction is obtained on the R table as it stood when that transaction began, i.e., the *swap* transactions do not impact the result of the *sum* transaction. When executed on database systems that do not implement snapshot isolation (e.g., SQLServer or DB2), only about 40% of the results were correct for the *sum* transaction [3].

Figure 1 traces throughput (i.e., total number of transactions/response time) for the serializable and read committed level as a function of the number of swap threads.
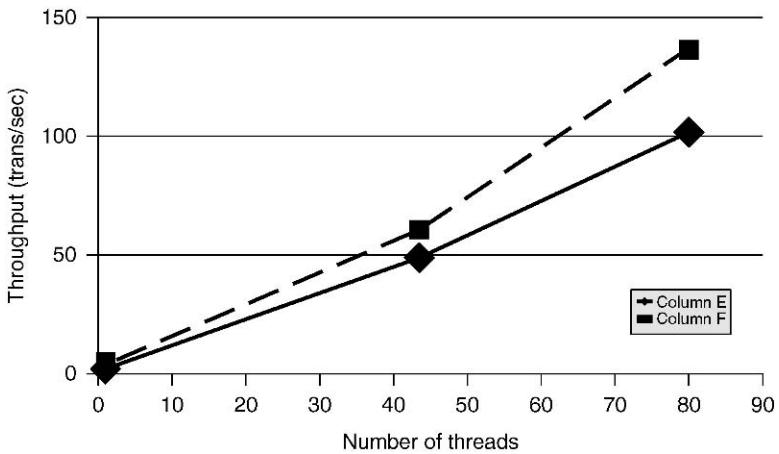
Read committed yields a higher throughput than serializable. The reason is that in the serializable isolation level, the *sum* transaction sets next-key locks while scanning the table, whereas at the read committed isolation level, the sum transaction relies on snapshot isolation, i.e., no read locks are used.

Counters

This experiment illustrates the benefit of using a system-defined counter as opposed to using an adhoc method based on a counter table. The experiment

Tuning Concurrency Control. Figure 1. Value of serializability experiment on MySQL 6.0.

Tuning Concurrency Control. Figure 2. Counter experiment on MySQL 6.0.

consists in running a fixed number of insert transactions concurrently. The number of threads used to run these transactions is the main parameter of this experiment. The experiment measures response time for the total number of transactions.

Figure 2 presents traces throughput (i.e., total number of transactions/response time) for the ad hoc and system implementation as a function of the number of swap threads. These results were obtained with the configuration used for the Value of Serializability experiment (see above). The SQL code and the data used for this experiment are available at the URL listed below.

In the experiment, the benefits of the system-based counter become more significant as the number of threads increases. The reason is that as the number of threads increase, the counter table becomes a hot spot and the benefits of using a latch released at the end of the statement (system method) over a lock held until the end of the transaction (ad hoc method) becomes significant. Note that the performance of the ad hoc method diminishes as the amount of processing in the transaction increases (i.e., as the time during which the lock is held increases).

URL to Code and Data Sets
Value of Serializability experiment: http://www.databasetuning.org/?sec=valueofserializability

Counter experiment: http://www.databasetuning.org/?sec=counter

Cross-references

Recommended Reading

1. Bernstein P., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston, MA, 1987.
2. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1992.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
4. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2001.

Tuning the Application Interface

Tuple Relational Calculus

Tuple-Generating Dependencies

Ronald Fagin
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Equality-generating dependency (egd)

Definition

Tuple-generating dependencies, or *tgds*, are one of the two major types of *database dependencies* (the other major type consists of *equality-generating dependencies*, or *egds*).

To define tgds, the notion of an *atomic formula* is first needed, which is a formula of the form $P(x_1,...,x_k)$,

where P is a k-ary relational symbol, and $x_1,...,x_k$ are variables, not necessarily distinct.

Then tgds are formulas of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where

1. $\phi(\mathbf{x})$ is a conjunction of atomic formulas, all with variables among the variables in $\mathbf{x}$.
2. every variable in $\mathbf{x}$ appears in $\phi(\mathbf{x})$ (but not necessarily in $\psi(\mathbf{x}, \mathbf{y})$).
3. $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas, all with variables among the variables in $\mathbf{x}$ and $\mathbf{y}$.

If $\mathbf{y}$ is empty, so that there are no existentially-quantified variables, then the tgd is called *full*.

Conditions (1) and (2) together are sometimes replaced by the weaker condition that $\phi(\mathbf{x})$ be an arbitrary first-order formula with free variables exactly those in $\mathbf{x}$.

Key Points

An example of a tgd is the formula

$$\forall x_1 \forall x_2 (R(x_1, x_1, x_2) \wedge S(x_2) \\ \rightarrow \exists y (R(x_1, y) \wedge T(x_2, y, x_1))).$$

Historically, tgds were introduced for the purpose of database normalization and design, with the first example being *multivalued dependencies* [2,5]. Fagin [3] defined the class of *embedded implicational dependencies*, which includes both tgds and egds, but he focused on the case where they are (i) unirelational (so that all atomic formulas involve the same relation symbol) and (ii) typed (so that no variable can appear in both the ith and jth position of an atomic formula if $i \neq j$). Beeri and Vardi [1] defined and named tgds and egds.

In recent years, tgds have been used to define *schema mappings* in *data exchange* [4], which describe how data structured under one schema (the *source schema*) is to be transformed into data structured under a second schema (the *target schema*). In this application, the atomic formulas in the premise $\phi(\mathbf{x})$ are all from the source schema, and the atomic formulas in the conclusion $\psi(\mathbf{x}, \mathbf{y})$ are all from the target schema.

Cross-references

Recommended Reading

1. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
2. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Sys., 2(3):262–278, 1977.
3. Fagin R. Horn clauses and database dependencies. J. ACM, 29(4):952–985, 1982.
4. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. Theor. Comput. Sci., 2005, pp. 89–124.
5. Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. 1976. Ph.D. Dissertation, UCLA.

Twigs

Two-Dimensional Shape Retrieval

Lei Chen
Hong Kong University of Science and Technology, Kowloon, Hong Kong, China

Definition

Shape is an important image feature, it is the geometrical information of an object after removing position, scale and rotational effects [3]. A shape is often represented by the contour map extracted from the images. Given a query 2D shape, *2D shape retrieval* retrieves a ranked list of similar 2D shapes from a collection of 2D polygonal models (contour points) based on the shape characteristics.

Figure 1 gives an example object shapes, which are represented by the extracted contour maps.
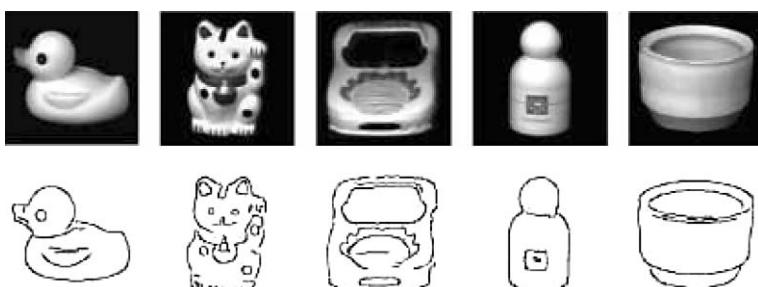
Historical Background

The study of shape retrieval can be traced back to 1980s. At that time, shape retrieval was treated as an key technique in object recognition of robot vision. Since then, shape retrieval has received much attention in the database domain due to its various application in biometrics, industry, medicine and anthropology.

Foundations

The shape retrieval problem is to retrieve shapes that are visually similar to the query shape. There are two key issues related to shape retrieval [6], *shape representation* and *similarity measure*. Given a shape, it should be represented in a form which is invariant to scaling, translation and rotation. For similarity measures, various measures are designed to meet application requirements. In fact, the two key issues are closely related to each other. Based on the different shape representations, different similarity measures are applied. Existing works represent shape in various ways including:

1. *Shape signature* is one-dimensional vector derived from the shape boundary coordinates. For example, the Euclidean distance between centroid point to the boundary points [8]. The shape of an object is represented by a set of normalized Fourier coefficients of the shape signature and the inner product

Two-Dimensional Shape Retrieval. Figure 1. Examples of shape of object represented by contour maps [4].

of the Fourier coefficients is used to measure the similarity between two shapes.

2. *Grid descriptor* is derived by overlaying the shapes with a coarse grid and assigning an '1' to a grid cell if more than 15% of the cell is covered by the shape, otherwise '0' [5]. Two shapes are compared by counting the number of different bits of corresponding normalized grid descriptors. The similarity measure conforms to human similarity perception, i.e., perceptually similar shapes have high similarity measure.

3. *Shape context* captures the distribution of the points relative to a reference point on the contour of an object, thus it offers a globally discriminative feature. Two shapes are similar if they have similar shape contexts. Therefore, the similarity between two shapes is computed by counting the number of matching corresponding points [2].

4. *Distance set* uses N nearest neighbors to represent each contour point [4]. The similarity between two shapes are measure by the cost of the cheapest correspondence relation of corresponding distance sets, which is computed by evaluating the minimum cost assignment in an associated bipartite graph.

5. *Curvature scale space* is formed by the positions of inflection points (x-axis) on the contours on every scale (y-axis) [7]. The shape representation is the positions of the maxima on these curves. The shape similarity is measures by relating the positions of the maxima of the corresponding curves.

6. *Symbolic features* refers to a shape which is represented in terms of multi-interval valued type features including shape centroid, extreme points, axis of least inertia with slope angle, and feature points on the axis of least inertia, etc. [1]. The similarity between two shapes is defined on these symbolic features as the average degree of similarity of all corresponding feature vector pairs.

7. *Shape space* refers a single point (vector) in a high-dimensional manifold. The vector is obtained by normalizing the landmark vector of the origin shape [9]. The similarity is measured as the geodesic distance between a shape and a model in the shape space.

In addition to the shape representation and similarity measure design, index structures should be built on the shape representations to allow shape similarity queries be answered efficiently.

Key Applications

Content Based Image Retrieval

Content-based image retrieval (CBIR), also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR) is the application of computer vision to the image retrieval problem, that is, the problem of searching for digital images in large databases.

Visual Surveillance

Visual surveillance in dynamic scenes is an active research topics in computer vision. The aim of visual surveillance is to make it possible that the computer can watch or monitor a scene by automatic localization, tracking and recognition.

Future Directions

Effective and robust similarity measures and efficient indexing structures.

Cross-references

► Feature-Based 3D Object Retrieval
► Image Retrieval

Recommended Reading

1. Attalla E. and Siy P. Robust shape similarity retrieval based on contour segmentation polygonal multiresolution and elastic matching. Pattern Recognit., 38(12):2229–2241, 2005.
2. Belongie S., Malik J., and Puzicha J. Shape matching and object recognition using shape contexts. IEEE Trans. Pattern Anal. Mach. Intell., 24(4):509–522, 2002.
3. Dryden I.L. and Mardia K.V. Statistical Shape Analysis. Wiley, New York, 1998.
4. Grigorescu C. and Petkov N. Distance sets for shape filters and shape recognition. IEEE Trans. Image Process., 12(10):1274–1286, 2003.
5. Lu G. and Sajjanhar A. Region-based shape representation and similarity measure suitable for content-based image retrieval. Multimedia Syst., 7(2):165–174, 1999.
6. Mehrotra R. and Gary J.E. Similar-shape retrieval in shape data management. Computer, 28(9):57–62, 1995.
7. Mokhtarian F. and Bober M. Curvature Scale Space Representation: Theory, Applications, and MPEG-7 Standardization. Kluwer, Norwell, MA, USA, 2003.
8. Zhang D. and Lu G. Evaluation of mpeg-7 shape descriptors against other shape descriptors. Multimedia Syst., 9(1):15–30, 2003.
9. Zhang J., Zhang X., Krim H., and Walter G.G. Object representation and recognition in shape spaces. Pattern Recognit., 36(5):1143–1154, 2003.

Two-Phase Commit

Yousef J. Al-Houmaily[1], George Samaras[2]
[1]Institute of Public Administration, Riyadh,
Saudi Arabia
[2]University of Cyprus, Nicosia, Cyprus

Definition

Two-phase commit (2PC) is a synchronization protocol that solves the *atomic commitment problem*, a special case of the *Byzantine Generals* problem. Essentially, it is used in distributed database systems to ensure *global atomicity* of transactions in spite of site and communication failures, assuming that a failure will be, *eventually*, repaired and each site guarantees atomicity of transactions at its local level.

Historical Background

2PC is the simplest and most studied *atomic commit protocol* (ACP). It was first published in [9] and [4]. Since then, the protocol has received much attention from the academia and industry due to its importance in distributed database systems, and the research has resulted in numerous variants and optimizations for different distributed database environments. These environments include main memory databases (e.g., [10]), real-time databases (e.g., [5]), mobile database systems (e.g., [12]), heterogeneous database systems (e.g., [1]), Web databases (e.g., [15]), besides traditional (homogeneous) distributed database systems (e.g., [13,3]).

Foundations

In a distributed database system, a transaction is decomposed into a set of *subtransactions*, each of which executes at a single participating database site. Assuming that each database site preserves atomicity of (sub)transactions at its local level, global atomicity cannot be guaranteed without taking additional measures. This is because without global synchronization a distributed transaction might end-up committing at some participating sites and aborting at others due to a site or a communication failure. Thus, jeopardizing *global* atomicity and, consequently, the consistency of the (distributed) database.

To achieve atomicity at the global level, there is a need for a synchronization protocol that ensures a unanimous final outcome for each distributed transaction and regardless of failures. Such a protocol is referred to as an *atomic commit protocol* (ACP). An ACP ensures that a distributed transaction is either *committed* and all its effects become persistent across all participating sites, or *aborted* and all its effects are obliterated as if the transaction had never executed at any site. This is the essence of the two-phase commit (2PC) protocol.
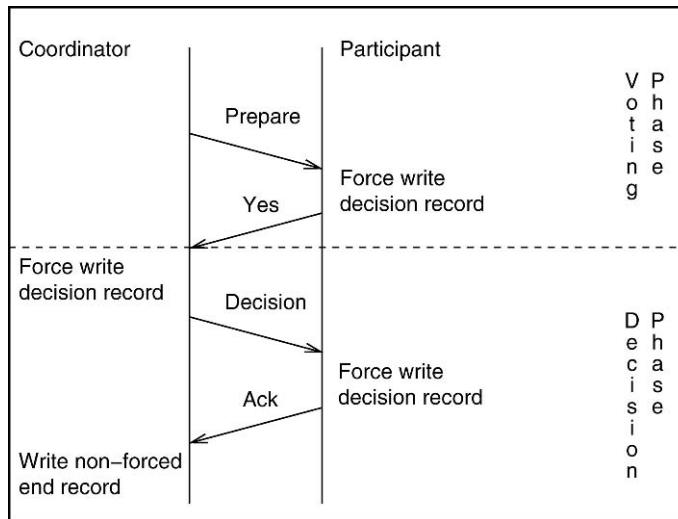
Dynamics of Two-Phase Commit

In 2PC, each transaction is associated with a designated site called the *coordinator* (or *master*). Although the coordinator of a transaction could be any of the sites participating in the transaction's execution, it is commonly the originating site of the transaction (i.e., the site where the transaction is first initiated). The rest of the sites are called *participants*, *subordinates*, *cohorts* or *slaves*. Once a transaction finishes its execution and indicates its termination point, through a commit primitive, to its coordinator, the coordinator initiates 2PC.

As the name implies, 2PC consists of two phases, namely a *voting phase* and a *decision phase*, as shown Fig. 1. During the voting phase, the coordinator requests all the sites participating in the transaction's execution to *prepare-to-commit* whereas, during the decision phase, the coordinator either decides to commit the transaction if *all* the participants are prepared to commit (voted "yes"), or to abort if any participant has decided to abort (voted "no"). On a commit decision, the coordinator sends out commit messages to *all* participants whereas, on an abort decision, it sends out abort messages to *only* those (required) participants that are prepared-to-commit (voted "yes").

When a participant receives a prepare-to-commit message for a transaction, it validates the transaction with respect to data consistency. If the transaction can be committed (i.e., it passed the validation process), the participant responds with a "yes" vote. Otherwise, it responds with a "no" vote and aborts the transaction, releasing all the resources held by the transaction.

If a participant had voted "yes", it can neither commit nor abort the transaction unilaterally and has to wait until it receives a final decision from the coordinator. In this case, the participant is said to be *blocked* for an indefinite period of time called *window of uncertainty* (or *window of vulnerability*) awaiting the coordinator's decision. When a participant receives the final decision, it complies with the decision, sends back an acknowledgement message (Ack) to the coordinator and releases all the resources held by the transaction.

Two-Phase Commit. Figure 1. The two-phase commit protocol.

When the coordinator receives Acks from all the participants that had voted "yes," it *forgets* the transaction by discarding all information pertaining to the transaction from its protocol table that is kept in main memory.

The resilience of 2PC to failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. The coordinator force writes a *decision* record prior to sending out its decision to the participants. Since a *forced write* of a log record causes a flush of the log onto a stable storage that survives system failures, the decision is not lost if the coordinator fails. Similarly, each participant force writes a *prepared* record before sending its "yes" vote and a *decision* record before acknowledging a decision. When the coordinator completes the protocol, it writes a non-forced *end* record in the volatile portion of its log that is kept in main memory. This record indicates that all (required) participants have received the decision and none of them will inquire about the transaction's status in the future. This allows the coordinator to (permanently) forget the transaction, with respect to 2PC, and garbage collect the log records of the transaction when necessary.

Recovery in Two-Phase Commit

Site and communication failures are detected by *time-outs*. When an operational site detects a failure, it invokes a recovery manager to handle the failure. In 2PC, there are four places where a communication failure might occur. The first place is when a participant is waiting for a prepare-to-commit message from the coordinator. This occurs before the participant has voted. In this case, the participant may unilaterally decide to abort the transaction. The second place is when the coordinator is waiting for the votes of the participants. Since the coordinator has not made a final decision yet and no participant could have decided to commit, the coordinator can decide to abort. The third place is when a participant had voted "yes" but has not received a commit or an abort final decision. In this case, the participant cannot make any unilateral decision because it is uncertain about the coordinator's final decision. The participant, in this case, is *blocked* until it re-establishes communication with the coordinator and, once re-established, the participant inquires the coordinator about the final decision and resumes the protocol by enforcing and, then, acknowledging the coordinator's decision. The fourth place is when the coordinator is waiting for the Acks of the participants. In this case, the coordinator re-submits its final decision to those participants that have not acknowledged the decision once it re-establishes communication with them. Notice that the coordinator cannot simply discard the information pertaining to a transaction from its protocol table or its stable log until it receives Acks from all the (required) participants.

To recover from site failures, there are two cases to consider: coordinator's failure and participant's failure.

For a coordinator's failure, the coordinator, upon its restart, scans its stable log and re-builds its protocol table to reflect the progress of 2PC for all the pending transactions prior to the failure. The coordinator has to consider only those transactions that have started 2PC and have not finished it prior to the failure (i.e., transactions that have decision log records without corresponding end log records in the stable log). For other transactions, i.e., transactions that were active at the coordinator's site prior to its failure without a decision record, the coordinator considers them as aborted transactions. Once the coordinator re-builds its protocol table, it completes the protocol for each of these transactions by re-submitting its final decision to all (required) participants whose identities are recorded in the decision record and waiting for their Acks. Since some of the participants might have already received the decision prior to the failure and enforced it, these participants might have already forgotten that the transaction had ever existed. Such participants simply reply with *blind* Acks, indicating that they have already received and enforced the decision prior to the failure.

For a participant's failure, the participant, as part of its recovery procedure, checks its log for the existence of any transaction that is in a prepared-to-commit state (i.e., has a prepared log record without a corresponding final decision one). For each such transaction, the participant inquires the transaction's coordinator about the final decision. Once the participant receives the decision from the coordinator, it completes the protocol by enforcing and, then, acknowledging the decision. Notice that a coordinator will be always able to respond to such inquires because it cannot forget a transaction before it has received the Acks of all (required) participants. However, there is a case where a participant might be in a prepared-to-commit state and the coordinator does not remember the transaction. This occurs if the coordinator fails after it has sent prepare-to-commit messages and just before it has made its decision. In this case, the coordinator will not remember the transaction after it has recovered. If a prepared-to-commit participant inquires about the transaction's status, the coordinator will *presume* that the transaction was aborted and responds with an abort message. This special case where an abort presumption is made about unremembered transactions in 2PC motivated the design of the *presumed abort* 2PC.

Underlying Assumptions

ACPs solve a special case of the problem of consensus in the presence of faults, a problem that is known in distributed systems as the *Byzantine Generals* problem [7]. This problem is, in its most general case, not solvable without some simplifying assumptions. In distributed database systems, ACPs solve the problem under the following general assumptions (among others that are sometimes ACP specific):

1. *Each site is sane*: A site is fail stop where it never deviates from its prescribed protocol. That is, a site is either operational or not but never behaves abnormally causing *commission* failures.
2. *Eventual recovery*: A failure (whether site or communication) will be, eventually, repaired.
3. *Binary outcome*: All sites unanimously agree on a single binary outcome, either commit or abort.

Performance Issues

There are three important performance issues that are associated with ACPs, which are as follows [3]:

1. *Efficiency During Normal Processing*: This refers to the cost of an ACP to provide atomicity in the absence of failures. Traditionally, this is measured using three metrics. The first metric is *message complexity* which deals with the number of messages that are needed to be exchanged between the systems participating in the execution of a transaction to reach a consistent decision regarding the final status of the transaction. The second metric is *log complexity* which accounts for the frequency at which information needs to be recorded at each participating site in order to achieve resiliency to failures. Typically, log complexity is expressed in terms of the required number of non-forced log records which are written into the log buffer (in main memory) and, more importantly, the number of forced log records which are written onto the stable log (on the disk). The third metric is *time complexity* which corresponds to the required number of rounds or sequential exchanges of messages in order for a decision to be made and propagated to the participants.
2. *Resilience to Failures*: This refers to the types of failures that an ACP can tolerate and the effects of failures on operational sites. An ACP is considered *non-blocking* if it never requires operational sites to

wait (i.e., block) until a failed site has recovered. One such protocol is 3PC.

3. *Independent Recovery*: This refers to the speed of recovery. That is, the time required for a site to recover its database and become operational, accepting new transactions after a system crash. A site can *independently* recover if it has all the necessary information needed for recovery stored locally (in its log) without requiring any communication with any other site in order to fully recover.

Most Common Two-Phase Commit Variants

Due to the costs associated with 2PC during normal transaction processing and the reliability drawbacks in the events of failures, a variety of ACPs have been proposed in the literature. These proposals can be, generally, classified as to enhance either (i) the efficiency of 2PC during normal processing or (ii) the reliability of 2PC by either reducing 2PC's blocking aspects or enhancing the degree of independent recovery. The most commonly pronounced 2PC variants are *presumed abort* (PrA) [11] and *presumed commit* (PrC) [11]. Both variants reduce the cost of 2PC during normal transaction processing, albeit for different final decisions. That is, PrA is designed to reduce the costs associated with aborting transactions whereas PrC is designed to reduce the costs associated with committing transactions.

In PrA, when a coordinator decides to abort a transaction, it does not force-write the abort decision in its log as in 2PC. It just sends abort messages to all the participants that have voted "yes" and discards all information about the transaction from its protocol table. That is, the coordinator of an aborted transaction does not have to write any log records or wait for Acks. Since the participants do not have to Ack abort decisions, they are also not required to force-write such decisions. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the participant to abort the transaction (by presumption). Thus, as the name implies, if no information is found in the log of the coordinator of a transaction, the transaction is presumed aborted.

As opposed to PrA, in which missing information about transactions at a coordinator's site is interpreted as abort decisions, in PrC, a coordinator interprets missing information about transactions as commit decisions when replying to inquiry messages. However, in PrC, a coordinator has to force write a commit *initiation* record for each transaction before sending out prepare-to-commit messages to the participants. This record ensures that missing information about a transaction will not be misinterpreted as a commit after a coordinator's site failure without an actual commit decision is made.

To commit a transaction, the coordinator force writes a commit record to logically eliminate the initiation record of the transaction and then sends out commit messages. The coordinator also discards all information pertaining to the transaction from its protocol table. When a participant receives the decision, it writes a non-forced commit record and commits the transaction without having to Ack the decision. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been committed, the coordinator, not remembering the transaction, will direct the participant to commit the transaction (by presumption).

To abort a transaction, on the other hand, the coordinator does not write the abort decision in its log. Instead, the coordinator sends out abort messages and waits for Acks before discarding all information pertaining to the transaction. When a participant receives the decision, it force writes an abort record and then acknowledges the decision, as in 2PC. In the case of a coordinator's failure, the initiation record of an interrupted transaction contains all needed information for its recovery.

Table 1 summarizes the costs associated with the three 2PC variants for the commit as well as the abort case assuming a "yes" vote from each participant: "m" is the total number of log records, "n" is the number of forced log writes, "p" is the number of messages sent from the coordinator to each participant and "q" is the number of messages sent back to the coordinator. For simplicity, these costs are calculated for the flat (two-level) execution model in which, unlike the multi-level execution model, a participant never initiates (i.e., spawns) new (sub)transactions that execute at other participants, forming a tree of communicating participants.

Compatibility of 2PC Variants

ACPs are incompatible in the sense that they cannot be used (directly) in the same environment without conflicts. This is true even for the simplest and most

Two-Phase Commit. Table 1. The costs for update transactions in 2PC and its most commonly known two variants

2PC Variant	Commit decision						Abort decision					
	Coordinator			Participant			Coordinator			Participant		
	m	n	p	m	n	q	m	n	p	m	n	q
Basic 2PC	2	1	2	2	2	2	2	1	2	2	2	2
Presumed abort	2	1	2	2	2	2	0	0	2	2	1	1
Presumed commit	2	2	2	2	1	1	2	1	2	2	2	2

closely related variants such as the basic 2PC, PrA and PrC. The analysis of ACPs shows that incompatibilities among ACPs could be due to (i) the semantics of the coordination messages (which include both their meanings as well as their existence), or (ii) the presumptions about the outcome of terminated transactions in case of failures [1].

The *presumed any* (PrAny) protocol [2] interoperates the basic 2PC, PrA, and PrC. It was proposed in the context of multidatabase systems, a special case of heterogeneous distributed databases, to demonstrate the difficulties that arise when one attempts to interoperate different ACPs in the same environment and, more importantly, to introduce the "*operational correctness criterion*" and the notion of "*safe state.*" Operational correctness means that all sites should be able, not only to reach an agreement but also, to forget the outcome of terminated transactions. On the other hand, the safe state means that, for any operationally correct ACP, the coordinator should be able to reach a state in which it can reply to the inquiry messages of the participants, in a consistent manner, without having to remember the outcome of terminated transactions forever.

In PrAny, a coordinator talks the language of the three 2PC variants and knows which variant is used by which participant. Based on that, it forgets a committed transaction once all PrA and 2PC participants Ack the commit decision, and forgets an aborted transaction once all PrC and 2PC participants Ack the abort decision. This is because only commit decisions are acknowledged in PrA whereas, in PrC, only abort decisions are acknowledged. However, unlike the other 2PC variants, in PrAny, a coordinator does not adopt a single presumption about the outcome of *all* terminated transactions. This is because, if it does so, the global atomicity of some transactions might be violated. For example, if the coordinator adopts for recovering purposes the abort presumption, it will respond with an abort message to a recovering PrC participant that inquires about a forgotten committed transaction. Similarly, if the coordinator adopts the commit presumption, it will respond with a commit message to a recovering PrA participant that inquires about a forgotten aborted transaction. Instead of using a single presumption, a coordinator in PrAny adopts the presumption of the protocol used by the inquiring participant. That is, if a participant inquires about a forgotten committed transaction, the participant has to be a PrC participant. This is because only PrC participants do not acknowledge commit decisions. Thus, the coordinator will reply with a commit message in accordance with PrC adopted by the participant. On the other hand, if a participant inquires about a forgotten aborted transaction, the participant has to be a PrA participant. This is because only PrA participants do not acknowledge abort decisions. Thus, the coordinator will reply with an abort message in accordance with PrA adopted by the participant. Knowledge about the used protocols by the participants could be recorded statically at the coordinator's site [2] or inferred dynamically by having a participant declares its used protocol in each inquiry message [15].

Key Applications

The use of 2PC (or one of its variants) is mandatory in any distributed database system in which the traditional atomicity property of transactions is to be preserved. However, the basic 2PC has never been implemented in any commercial database system due to its unnecessary costs compared to its two other most commonly known variants. Instead, PrA is considered the *de facto* standard in the industry and has been incorporated as part of the current X/Open DTP [14] and ISO OSI-TP [6] distributed transaction processing standards. PrA is chosen instead of PrC because (i) the cost of PrC for committing transactions is not symmetric with the

cost of PrA for aborting transactions (which is high-lighted in Table 1) and, more importantly, (ii) PrA is much cheaper to use with read-only transactions when complementing it with the traditional *read-only* optimization [13,3] (which is also part of the current database standards).

The above two reasons that favor PrA have been nullified with new 2PC variants and a read-only optimization called *unsolicited update-vote* (UUV). Thus, PrC is expected to become also part of future database standards, especially that the two variants can be incorporated in the same environment without any conflicts [1,15].

Cross-references

► Atomicity
► Distributed Database Systems
► Distributed Recovery
► Distributed Transaction Management

Recommended Reading

1. Al-Houmaily Y. Incompatibility dimensions and integration of atomic commit protocols. Int. Arab J. Inf. Technol., 5(4):2008.
2. Al-Houmaily Y. and Chrysanthis P. Atomicity with incompatible presumptions. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 306–315.
3. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Uppersaddle River, NJ, 1998, pp. 370–416.
4. Gray J.N. Notes on data base operating systems. In Operating Systems – An Advanced Course. M.J. Flynn et al. (eds.), LNCS, Vol. 60, Springer, London, 1978, pp. 393–481.
5. Haritsa J., Ramamritham K., and Gupta R. The PROMPT real-time commit protocol. IEEE Trans. Parallel Distributed Syst., 11(2):160–181, 2000.
6. ISO. Open systems interconnection – Distributed transaction processing – Part 1: OSI TP Model. ISO/IEC, 10026–1, 1998.
7. Lamport L., Shostak R., and Pease M. The Byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
8. Lampson B. and Lomet D. A new presumed commit optimization for two phase commit. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 630–640.
9. Lampson B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, CA, 1976.
10. Lee I. and Yeom H. A single phase distributed commit protocol for main memory database systems. In Proc. 16th Int. Parallel and Distributed Processing Symp., 2002, pp. 14–21.
11. Mohan C., Lindsay B., and Obermarck R. Transaction management in the R* distributed data base management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Nouali N., Drias H., and Doucet A. A mobility-aware two-phase commit protocol. Int. Arab J. Inf. Technol., 3(1):2006.
13. Samaras G., Britton K., Citron A., and Mohan C. Two-phase commit optimizations in a commercial distributed environment. Distrib. Parall. Databases, 3(4):325–361, 1995.
14. X/Open Company Limited. Distributed Transaction Processing: Reference Model. Version 3 (X/Open Document No. 504), 1996.
15. Yu W. and Pu C. A Dynamic Two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1):2007.

Two-Phase Commit Protocol

Jens Lechtenbörger
University of Münster, Münster, Germany

Synonyms

XA standard

Definition

The *Two-phase commit (2PC)* protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via 2PC a unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in two phases, namely the prepare (or voting) and the commit (or decision) phase, which explains the protocol's name.

The protocol is executed by a *coordinator* process, while the participating servers are called *participants*. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying – *via prepare messages* – all participants whether to abort or to commit the transaction. If all participants vote to commit then in the second phase the coordinator informs all participants to commit their share of the transaction by sending a *commit message*. Otherwise, the coordinator instructs all participants to abort their share of the transaction by sending an *abort message*. Appropriate log entries are written by coordinator as well as participants to enable restart procedures in case of failures.

Historical Background

Essentially, the 2PC protocol is modeled after general contract law, where a contract among two or more
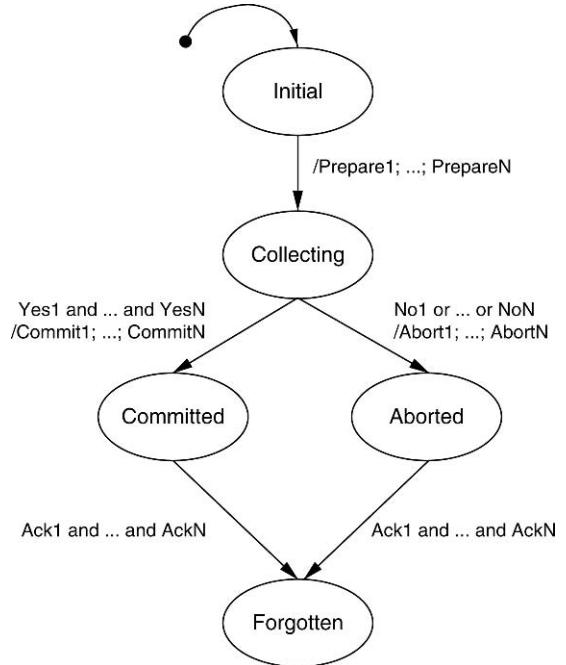
parties is only established if all parties agree; hence, the underlying idea is well-established in everyday life. According to [3] the first known implementation in a distributed system was performed by Nico Garzado for the Italian social security system in the early 1970s, while the protocol's name arose in the mid 1970s. Early scientific presentations are given by Gray [2] and by Lampson and Sturgis [4]. Since then an API for the 2PC protocol has been standardized under the name XA within the X/Open Distributed Transaction Processing (DTP) model [7], and this API has been incorporated into several middleware specifications and implemented in numerous software components.

Foundations
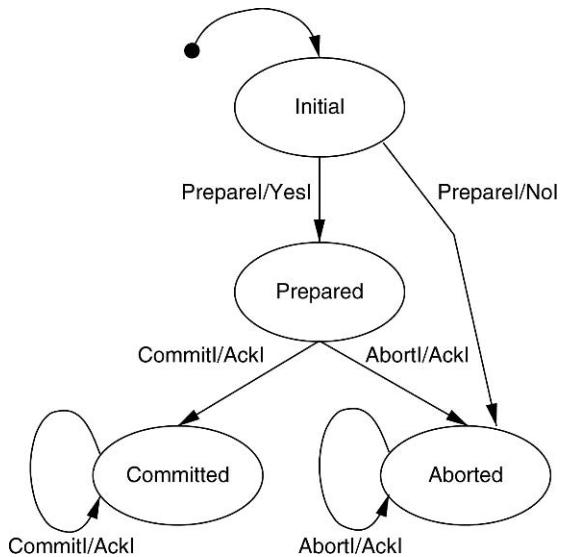
The 2PC protocol as described and analyzed in detail in [8] assumes that parts of a single (distributed) transaction involve resources hosted by multiple *resource managers* (e.g., database systems, file systems, messaging systems, persistent programming environments), which reside on possibly different nodes of a network and are called *participants* of the protocol. For every transaction one *coordinator* process, typically running on the node of that participant where the transaction was initiated, assumes responsibility for executing the 2PC protocol; alternative strategies for selecting (and transferring) the coordinator are discussed in [8]. The states through which coordinator and participants move in the course of the protocol are illustrated in Figs. 1 and 2, resp., and explained in the following. Such statecharts represent finite state automata, where ovals denote states, labeled arcs denote state transactions, and arc labels of the form "precondition/action" indicate that (i) the state transition is only enabled if the precondition is satisfied and (ii) the given action is executed when the state is changed.

Basic Protocol

As long as a transaction is still executing ordinary operations, coordinator as well as all participants operate in the Initial state. When the coordinator is requested to commit the transaction, it initiates the first phase of the 2PC protocol: To capture the state of the protocol's execution (which needs to be available in case of protocol restarts as explained below), the coordinator first forces a *begin log entry*, which includes a transaction identifier as well as a list of the transaction's participants, to a stable log. Afterwards, the

Two-Phase Commit Protocol. Figure 1. Statechart for coordinator (given *N* participants).

Two-Phase Commit Protocol. Figure 2. Statechart for participant *I*.

coordinator sends a *prepare* message to every participant, enters the Collecting state and waits for replies.

Upon receiving a prepare message, a participant decides whether it is able to commit its share of the

transaction. In either case, suitable log entries for later recovery operations as well as a *prepared log entry* indicating the vote ("Yes" or "No") are forced to a stable log, before a response message containing the vote is sent back to the coordinator. In case of a No-vote, the participant switches into the Aborted state and immediately aborts the transaction locally. In case of a Yes-vote, the participant moves into the Prepared state. In the latter case the participant is said to be *in doubt* or *blocked* as it has now given up its local autonomy and must await the final decision from the coordinator in the second phase (in particular, locks cannot be released yet).
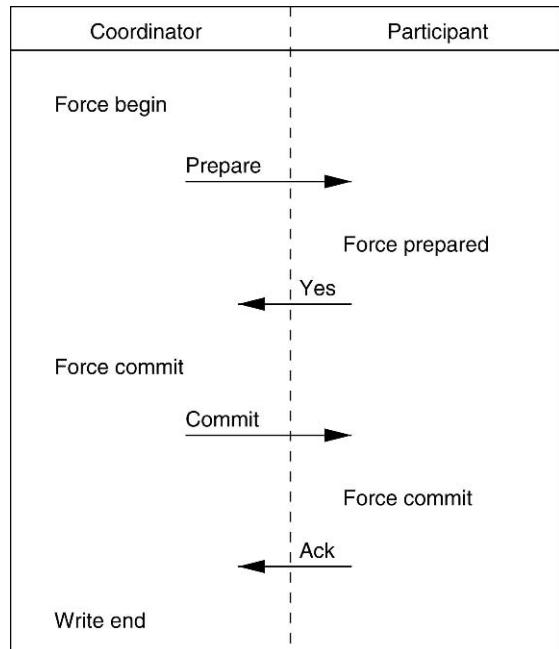
Once the coordinator has received all participants' response messages it starts the second phase of the 2PC protocol and decides how to complete the global transaction: The result is "Commit" if all participants voted to commit and "Abort" otherwise. The coordinator then forces a *commit* or *abort log entry* to the stable log, sends a message containing the final decision to all participants, and enters the corresponding state (Committed or Aborted).

Upon receipt of the decision message, a participant commits or aborts the local changes of the transaction depending on the coordinator's decision and forces suitable log entries for later recovery as well as a *commit* or *abort log entry* to a stable log. Afterwards, it sends an acknowledgment message to the coordinator and enters the corresponding final state (Committed or Aborted).

Once the coordinator has received all acknowledgment messages it ends the protocol by writing an *end log entry* to a stable log to enable later log truncation and enters the final state, Forgotten. The actions described for the overall process are summarized in Fig. 3 for the case of a transaction commit. (For multiple participants, the actions simply have to be duplicated; in case of abort, at least one of the participants votes "No", which implies that all occurrences of "commit" are replaced with "abort".)

Protocol Restart

The log entries seen so far are used to restart the 2PC protocol after so-called soft crashes of coordinators or participants, i.e., failures like process crashes which lead to a loss of main memory but which leave secondary storage intact. In particular, as participants always force log entries before sending replies, the coordinator never needs to resend messages for which replies have

Two-Phase Commit Protocol. Figure 3. Actions for transaction commit in the basic protocol.

been received. Moreover, log truncation (garbage collection) may occur once all acknowledgment messages have arrived. Finally, every log entry uniquely determines a state, and the last log entry determines the most recent state prior to a failure. Clearly, failures in the final states (Forgotten for the coordinator and Committed or Aborted for a participant) do not require any action. For the remaining states, restart procedures are as follows:

If the coordinator fails in the Initial or the Collecting state, it simply restarts the protocol in the Initial state. (Coordinators writing received votes into the log could recover differently from the Collecting state.) If it fails in the Committed or in the Aborted state, it resends the decision message to all participants, and continues waiting for acknowledgments in the previous state.

If a participant fails in the Initial state it did not yet participate in the 2PC protocol and is free to decide arbitrarily when asked later on. If it fails in the Prepared state it either waits for the coordinator to announce the decision or actively queries the coordinator or other participants for the decision.

In addition to these restart procedures, coordinator and participants also need to be able to recover from message losses. To this end, standard timeout

mechanisms are employed: Whenever a message is sent, a timer starts to run. If the timer expires before an appropriate answer is received, the message is simply resent (assuming that either original message or answer are lost; e.g., if the coordinator is missing some votes in the Collecting state, it resends a prepare message to every participant that did not answer in time). Finally, if repeated timeouts occur in the Collecting state the coordinator may decide to abort the transaction globally (as if an "Abort" vote was received), and a participant may unilaterally abort the transaction in the Initial state if no prepare message arrives.

Hierarchical and Flattened 2PC

New participants enter the 2PC protocol whenever they receive requests (e.g., to execute SQL statements) from already existing participants. In such a situation, the new participant can be regarded as child node of the requesting participant, and all such parent-child relationships form a *participant tree* with the transaction's initiator as root node. To execute the 2PC protocol, that tree may either be used directly or flattened as explained in the following.

For the *flattened* 2PC, one node in the participant tree, e.g., the root node, is chosen as coordinator, and this coordinator communicates directly with every participant contained in the tree to execute the basic 2PC protocol as described above. In contrary, in case of the hierarchical 2PC, the root node acts as global coordinator, the leaf nodes are ordinary participants, and the inner nodes are participants with respect to their parents as well as sub-coordinators for their children. Thus, when an inner node receives a 2PC message from its parent, the inner node first has to forward the message to its children before it responds on behalf of the *entire* subtree. For example, a prepare message is forwarded down the tree recursively, and an inner node first waits for all votes of its children before it decides, write a log entry, responds with a vote to the parent, and makes a transition to the Prepared (if all children voted to commit) or Aborted state.

Optimizations
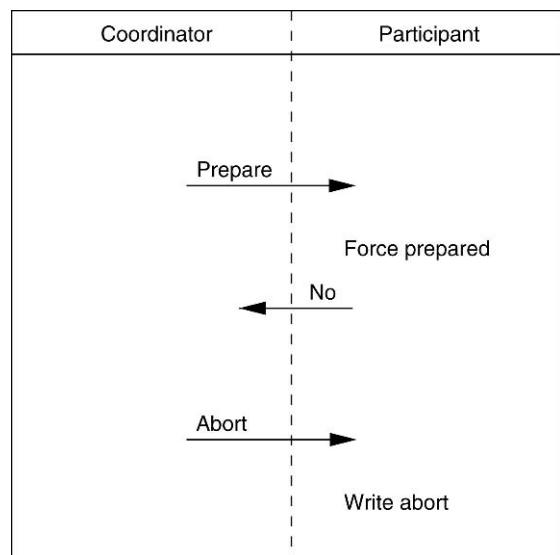
As the 2PC protocol involves costly operations such as sending messages and forcing log entries, several optimizations of the basic protocol have been proposed. In the following the most common variants based on

presumption are sketched; further details and techniques such as real-only subtree optimization, coordinator transfer, and three-phase commit (3PC) to reduce blocking are presented in [8].

The key idea for presumption based optimizations is to write less log entries and send fewer messages in a systematic way such that in case of a failure the missing information can be compensated for by suitable presumptions concerning the transaction's state. As the basic protocol described above is not based on any presumptions, it is also called *presumed-nothing protocol*. In contrast, in the *presumed-abort protocol*, which aims to optimize the case of aborted transactions, the essential idea is to omit certain information concerning transaction aborts. If that information is needed but absent later on, abort is presumed. In fact, for the presumed-abort protocol the following information is omitted:

- The Coordinator's begin and abort log entries are omitted.
- The participants' abort log entries are not forced.
- Participants do not send acknowledgment messages before entering the Aborted state.

The actions required in case of a transaction abort are summarized in Fig. 4, which indicates significant

Two-Phase Commit Protocol. Figure 4. Actions for transaction abort in the presumed-abort variant.

savings when compared with the actions for the basic protocol shown in Fig. 3. In the presumed-abort variant, if a participant fails after receiving the abort decision from the coordinator and restarts without finding a log entry, it queries the coordinator for the decision. As the coordinator does not find the appropriate log entry (which has never been written) it presumes that the transaction should be aborted and informs the participant accordingly, which leads to a globally consistent decision.

Alternatively, in the *presumed-commit protocol*, which aims to optimize the case of committed transactions, the following information is omitted:

- The participants' commit log entries are not forced.
- Participants do not send acknowledgment messages before entering the Committed state.

The actions required in case of a transaction commit are summarized in Fig. 5, which again indicates significant savings in comparison to the basic protocol shown in Fig. 3. In this variant, log entries of committed transactions can be garbage collected as missing transactions are presumed to have committed. Thus, if a participant fails after receiving the commit decision from the coordinator and restarts without finding a log entry, it queries the coordinator for the decision. If the coordinator does not find any log entry it presumes that the transaction has committed and informs the participant accordingly, which leads to a globally consistent decision.
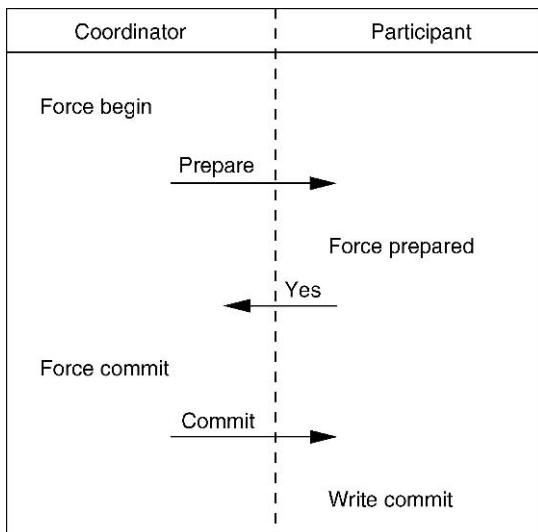
Key Applications

While there is no single key application for the 2PC protocol, it is applicable wherever decentralized data needs to be shared by multiple participants under transactional guarantees, e.g., in e-commerce or e-science settings. More specifically, the 2PC protocol is widely implemented in database systems (commercial as well as open source ones), TP monitors, and message queue systems, where it is used in the background to provide atomicity for distributed transactions. In addition, the XA interface [7] for the protocol, more precisely for the hierarchical presumed-abort variant, has been adopted in the CORBA Transaction Service specified by the OMG [5] and is used as basis for the Java Transaction API (JTA) [6]. Furthermore, the 2PC protocol is also part of the Web Services Atomic Transaction specification [1] to enable the interoperable atomic composition of Web Service invocations.

Cross-references

▶ ACID Properties
▶ Distributed Transaction Management
▶ Logging and Recovery
▶ Transaction
▶ Transaction Management

Recommended Reading

1. Cabrera L.F. et al. Web services atomic transaction, 2005.
2. Gray J. Notes on database operating systems. In Operating Systems: An Advanced Course. Lecture Notes in Computer Science. R. Bayer, M.R. Graham, G. Seegmüller (eds.). 60, Springer, Berlin Heidelberg New York, 1978, pp. 393–481.
3. Gray J. and Reuter A. Transaction processing: concepts and techniques. Morgan Kaufmann, San Francisco, CA, 1993.
4. Lampson B.W. and Sturgis H. Crash recovery in distributed data storage systems. Technical Report, Xerox Palo Alto Research Center, Palo Alto, CA.
5. OMG Transaction Service, version 1.4. http://www.omg.org/technology/documents/formal/transaction_service.htm, 2007.
6. Sun Microsystems. Java Transaction API (JTA). http://java.sun.com/jta/; http://java.sun.com/jta/, 2007.
7. The Open GROUP Distributed Transaction Processing: The XA Specification. X/Open Company Ltd, ISBN 1 872630 24 3, 1991.
8. Weikum G. and Vossen G. Transactional information systems – theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Two-Phase Commit Protocol. Figure 5. Actions for transaction commit in the presumed-commit variant.

T

Two-Phase Locking

Georg Lausen
University of Freiburg, Freiburg, Germany

Synonyms

Locking protocol; Isolation; Conflict serializability;
Pessimistic scheduler

Definition

A locked transaction is a transaction which, in addition
to read and write actions, contains lock and unlock
operations to the data items. Lock and unlock opera-
tions enable a database system to control the order of
read and write actions of a concurrent set of transac-
tions. A locking policy is a set of rules which restrict the
possible ways to introduce lock and unlock operations
into a transaction. A locking policy is safe, if, whenever
all the transactions conform to the policy, any history
of the transactions is guaranteed to be serializable.
Two-Phase Locking is a safe locking policy which is
based on the simple rule saying a transaction is not
allowed to further lock a data item once it has already
unlocked some data item.

Historical Background

Two-Phase Locking was first described in [6]. Later
the basic policy has been extended into several direc-
tions. In [1] ordered sharing of locks is proposed
which allows more than one transaction to hold a
lock on a data item as long as the actions are performed
in the same order as the locks have been acquired.
Altruistic locking has been proposed by [10]. A trans-
action may allow other transactions to access data
items it has locked if it will not access these items
later again. This protocol is designated to situations
in which long and short transactions are running con-
currently. Two-Phase Locking taking old values of a
data item into account is described in [2]. Also hybrid
protocols have been proposed which allow to apply
other techniques than Two-Phase Locking simulta-
neously [5,8].

Foundations

Locking protocols are among the earliest mechanisms
which have been developed for controlling a set of
transactions to achieve serializability. The reason is

not surprising, as locking is a very intuitive means
to control transactions: the processing of an action
may be either immediately allowed or delayed. Using
locking, the critical actions for serializability are all
those, whose order of processing might influence the
effects of the transactions and thereby affect serial-
izability. Such actions are called conflicting. Read and
write actions are conflicting, whenever they refer to
the same data item and at least one of them is a write
action. The notion of serializability which is appli-
cable under these assumptions is called conflict-
serializability.

For the following, a database is a finite set $D = \{x, y,
z,...\}$ of disjoint data items. A transaction $t = (op_t, <_t)$
consists of a set of steps op_t which are assumed to be
totally ordered by $<_t$. Read actions $r(x)$ and write
actions $w(x)$ are steps, where x is the data item on
which the respective action is processed. A history s of
a set T of concurrent transactions is a pair $s = (op_s, <_s)$,
where $op_s = \cup_{t \in T} op_t$ and $<_s$ is a total order on the steps
in op_s which preserves $<_t$ for $t \in T$. $<_s$ also is called an
interleaving of the transactions in T. For notational
simplicity, total orders $<_t$ and $<_s$, $t \in T$, s a history of
T, are also written as action sequences, where the total
order is defined by considering a sequence from left to
right. A history is called serializable, if it is equivalent to a
serial, i.e., not interleaved history of the same set of
transactions.

The decision whether or not a history is serializable
can be based on an analysis of a so called conflict graph
whose definition is based on the relative order of
conflicting actions. Let $s = (op_s, <_s)$ be a history of a
set of transactions T. The conflict graph $C(s)$ is a
directed graph with set of nodes T and edges $t_i \rightarrow t_j$,
$i \neq j$, whenever one of the following conditions is
fulfilled:

RW − conflict: $r_i(x) \in op_i, w_j(x) \in op_j, r_i(x) <_s w_j(x)$
and for all $w_k(x) \in op_s, i \neq k, j \neq k,$
there either holds $w_k(x) <_s r_i(x)$
or $w_j(x) <_s w_k(x),$

WR − conflict: $w_i(x) \in op_i, r_j(x) \in op_j, w_i(x) <_s r_j(x)$
and for all $w_k(x) \in op_s, i \neq k, j \neq k,$
there either holds $w_k(x) <_s w_i(x)$
or $r_j(x) <_s w_k(x),$

WW − conflict: $w_i(x) \in op_i, w_j(x) \in op_j, w_i(x) <_s w_j(x)$
and for all $w_k(x) \in op_s, i \neq k, j \neq k,$
there either holds $w_k(x) <_s w_i(x)$
or $w_j(x) <_s w_k(x).$

It is well known (e.g., [9,12]), whenever the conflict graph of a history is acyclic, then the history is serializable. However, this condition is not necessary, there may exist histories whose conflict graph is cyclic and still these histories are serializable. In practice, such histories are considered to be not of interest and, as a consequence, as correctness notion conflict-serializability (CSR) is used:

CSR : A history is conflict − serializable if and
only if its conflict graph is acyclic.

To keep the terminology simple, instead of conflict-serializability the notion of serializability will be used in the sequel. The serializability of a history $s = (op_s, <_s)$ of a set of transactions T depends on the relative ordering of the conflicting actions of the involved transactions. Consider two transactions t, t'. Let t, among others, contain actions p, q and t' actions p', q'. Now assume that p and p' are in conflict and q and q' as well. Let further $p <_s p'$ and $q' <_s q$. s is not serializable as the conflict graph $C(s)$ contains the cycle $t \rightarrow t' \rightarrow t$. This means that there cannot exist a serial schedule s^* containing t and t' with the same order on the conflicting actions. For such a schedule s^*, either $p <_{s^*} p'$ and $q <_{s^*} q'$, or $p' <_{s^*} p$ and $q' <_{s^*} q$; both orderings are not compatible to $<_s$. So why not enforce $q <_s q'$ once $p <_s p'$ has occurred? This means to ask for a transaction scheduler who is able to enforce certain relative orderings of conflicting actions such that not serializable orderings cannot occur. Two-Phase Locking, which is called 2PL for brevity in the sequel, is the most widely used technique which can be used to control the concurrent execution of transactions in a way which guarantees serializability.

To control a concurrent set of transactions 2PL uses lock and unlock operations. A locked transaction $t = (op_t, <_t)$ is a transaction which, in addition to the read and write actions, contains lock operations Lx and unlock operations Ux, where Lx, $Ux \in op_t$ and $<_t$ is a total order as before, however now ordering locks and unlocks as well. For a locked transaction $t = (op_t, <_t)$, whenever $p \in op_t$, $p \in \{ r(x), w(x)\}$, the following conditions must hold: (i) Lx, $Ux \in op_t$, (ii) $Lx <_t Ux$ and (iii) $Lx <_t p <_t Ux$.

Let $t = r(x)w(x)r(y)w(y)$ be a transaction without any lock and unlock operations. Under the above conditions, there still exist many ways to insert lock

and unlock operations into t, as it is demonstrated by the following examples:

$$
\begin{aligned}
&(a) Lx\ r(x)\ w(x)\ Ux\ Ly\ r(y)\ w(y)\ Uy, \\
&(b) Lx\ r(x)\ w(x)\ Ly\ Ux\ r(y)\ w(y)\ Uy, \\
&(c) Lx\ r(x)\ w(x)\ Ly\ r(y)\ w(y)\ Ux\ Uy, \quad (1) \\
&(d) Lx\ Ly\ r(x)\ w(x)\ Ux\ r(y)\ w(y)\ Uy, \\
&(e) Lx\ Ly\ r(x)\ w(x)\ r(y)\ w(y)\ Ux\ Uy.
\end{aligned}
$$

The idea of a lock operation Lx and the corresponding unlock operation Ux is to grant a transaction t the right of an exclusive access to data item x for the interval defined by the time of the processing of the lock and the processing of the succeeding unlock operation. The decision whether or not a lock can be granted typically is based on a so called lock table $\mathcal{L}$. Starting from an empty table $\mathcal{L}$, a lock operation Lx can only then be granted to a transaction t, if $\mathcal{L}$ does not contain an entry with respect to x. In that case (x, t) is inserted into $\mathcal{L}$. Otherwise transaction t has to wait until the condition is fulfilled. When later t processes the corresponding unlock operation Ux, the entry (x, t) is deleted from the lock table $\mathcal{L}$. Therefore, for any history s of locked transaction the following condition on locks and unlocks (LUL) must hold, where $t_1, t_2 \in T$:

LUL : If $L_1\ x <_s L_2\ x$, then $U_1\ x <_s L_2\ x$.

Locking by itself does not guarantee serializability. Consider the two transactions t_1, t_2:

$$t_1 : L_1 x\ r_1(x)\ w_1(x)\ U_1 x\ L_1 y\ r_1(y)\ w_1(y)\ U_2 y,$$
$$t_2 : L_2 x\ L_2 y\ r_2(x)\ w_2(x)\ r_2(y)\ w_2(y)\ U_2 x\ U_2 y,$$

and a history s representing an interleaving of t_1 and t_2:

$$s : L_1 x\ r_1(x)\ w_1(x)\ U_1 x\ L_2 x\ L_2 y\ r_2(x)$$
$$w_2(x)\ r_2(y)\ w_2(y)\ U_2 x\ U_2 y\ L_1 y\ r_1(y)\ w_1(y)\ U_1 y.$$

This schedule is not serializable; $w_1(x) <_s r_2(x)$ and $w_2(y) <_s r_1(y)$ force a cycle $t_1 \rightarrow t_2 \rightarrow t_1$ in the conflict graph which contradicts serializability. Therefore, rules are needed which define how locks and unlocks should be introduced into the single transactions such that, whatever other transactions are running concurrently, serializability is guaranteed. In other words, a locking policy has to be introduced. The most widely used locking policy is called Two-Phase Locking (2PL) and is expressed by the surprisingly simple sentence:

2PL : Whenever a transaction t has executed its *first* operation, no further lock operations *of* t are allowed.

All except the first transaction listed in (1) perfectly obey 2PL. The remaining four locked versions (b) – (e) implement different strategies for locking. Transaction (b) unlocks x as early as possible and locks y as late as possible. It therefore tries to minimize delay of other transactions by keeping the intervals of locked data items as small as possible. At a first glance, this seems to be an attractive approach. However, because of the early unlocking, another transaction may read the value written of x before the transaction has finished. Reading values of data items from transactions which have not yet reached their end is highly problematic for recovery reasons. Only after a transaction has executed its final commit successfully, a database system guarantees that the effects of the respective transactions are permanent and will survive system and transaction failures. Version (c) behaves more carefully and keeps all data items locked until the end of the transaction. The remaining versions (d) and (e) lock all data items in advance and unlock either as early as possible, respectively at the end of the transaction. Taking the possibility of failures into account, only versions (c) and (e) are acceptable. In practice, locks are kept until to the end of a transaction giving rise to the notion of strict 2PL. The following theorem states that 2PL indeed is a safe locking policy:

Theorem: *Let T be a set of transactions. If all $t \in T$ obey 2PL, then any history of the transactions in T is serializable.*

Proof: For any $t \in T$, the lock point of t is defined as t's last lock operation. Now consider a history s of the transactions in T such that all $t \in T$ obey 2PL. Under the assumption, that s is not serializable, a contradiction to 2PL can be derived thereby proving serializability of s which in turn proves the theorem.

If s is not serializable, then the conflict graph of $<_s$ contains a cycle, which, without loss of generality, is of the form $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_k \rightarrow t_1$. An edge $t \rightarrow t'$ can only then be part of the cycle, when there exists a data item x such that t and t' perform conflicting actions on x. As all transactions use locks, transaction t' can only then execute its action on x, after t has processed the unlock Ux. Before processing the respective action on

x, t' has to lock x. Applying this observation on all edges of the cycle the following restrictions on the order $<_s$ can be concluded, where $x_1, \ldots, x_k$ are data items:

$$U_1 x_1 <_s L_2 x_1,$$
$$\vdots$$
$$U_{k-1} x_{k-1} <_s L_k x_{k-1},$$
$$U_k x_k <_s L_1 x_k.$$

Let l_i be the lock point of transaction t_i, $1 \leq i \leq k$. From the structure of $<_s$ it follows $l_1 <_s l_2, \ldots, l_{k-1} <_s l_k$ and $l_k <_s l_1$. Therefore it holds $l_1 <_s l_1$, which is a contradiction to the total order $<_s$.

2PL is optimal in the sense that for any locked transaction t_1 which does not follow 2PL, a locked transaction t_2 can be constructed such that for $T = \{t_1, t_2\}$ there exists a history which is not serializable. Consider the lock and unlock operations of t_1 be given by $L_1 x <_{t_1} U_1 x <_{t_1} L_1 y <_{t_1} U_1 y$ and the lock and unlock operations of t_2 be given by $L_2 x <_{t_2} L_2 y <_{t_2} U_2 y <_{t_2} U_2 x$. t_2 follows the 2PL policy, however t_1 does not. If both transactions are running concurrently, then the following order of locks and unlocks being part of a history s may happen:

$$L_1 x <_s U_1 x <_s L_2 x <_s L_2 y <_s U_2 y <_s U_2 x <_s L_1 y <_s U_1 y.$$

Obviously, if between any pair of lock and unlock operations Lx, Ux and Ly, Uy there exist read and write actions to the respective data items, conflicts happen and history s is not serializable. Therefore, t_1 has also to follow 2PL when serializability has to be guaranteed for arbitrary sets of transactions T, $t_1 \in T$.

However, optimality in the above sense does not imply that in every serializable history of transactions without locks and unlocks, locks and unlocks can be inserted according to 2PL in a way not violating LUL. Consider a serializable history s of $T = \{t_1, t_2, t_3\}$ with order of actions

$$r_1(x) <_s r_2(x) <_s w_2(x) <_s r_3(y) <_s w_3(y) <_s w_1(y).$$

It is impossible to insert lock and unlock operations into the transactions in T such that any $t \in T$ obeys 2PL and s fulfills LUL. Because of $r_1(x) <_s r_2(x)$ it must hold $U_1 x <_s L_2 x$ and because of 2PL it must hold $L_1 y <_s U_1 x$. However, as $w_1(y) <_s U_1 y$, this implies $L_1 y <_s r_3(y) <_s w_3(y) <_s U_1 y$. Therefore, either $L_3 y <_s L_1 y <_s U_3 y$, or $L_1 y <_s L_3 y <_s U_1 y$. Both orderings

contradict the basic locking condition LUL. Therefore, 2PL is a safe locking policy, however some serializable histories of a set of transactions T may be excluded. In other words, 2PL can only accept a strict subset of the set of all serializable histories, in general.

The model for locking presented so far is overly restrictive, as it does not distinguish between read and write actions. Serializability of a history is only then an issue, when some of the involved transactions modify the database, i.e., perform write actions. Therefore, when locking it should be possible to distinguish between locks for read actions $L^R x$ and locks for write actions $L^W x$. If a transaction reads and writes a data item x, then a single write lock is in order. It should be possible that arbitrarily many transactions may lock a data item for reading as long as there is no concurrent transaction writing the same data item. For any history s of locked transaction $t_1, t_2 \in T$ the following conditions on locks and unlocks (LUL^{RW}) must hold, which do not impose any restrictions on the ordering of read locks $L_1^R x$ and $L_2^R x$:

$$\text{LUL}^{\text{RW}} : \begin{array}{l} \text{If } L_1^R x <_s L_2^W x, \text{ then } U_1 x <_s L_2^W x. \\ \text{If } L_1^W x <_s L_2^R x, \text{ then } U_1 x <_s L_2^R x. \\ \text{If } L_1^W x <_s L_2^W x, \text{ then } U_1 x <_s L_2^W x. \end{array}$$

When locking is used, deadlocks may occur. Consider transactions t_1, t_2 as follows: $t_1 = L_1 x\ r_1(x)\ L_1 y\ w_1(y)\ U_1 x\ U_1 y$ and $t_2 = L_2 y\ r_2(y)\ L_2 x\ w_2(x)\ U_2 y\ U_2 x$. A prefix s' of a history s cannot be continued to a complete history containing all the steps of t_1 and t_2, if, for example, $L_1 x <_{s'} L_2 y$ and s' does not contain $U_1 x$. To complete the prefix, $L_2 x$ and also $L_1 y$ have to be considered for $<_s$. However this is not possible, because otherwise the condition LUL would be violated. The problem of deadlocks is inherent to locking and practical systems solve deadlocks by aborting one of the involved transactions [12].

The transaction model abstracts away many aspects of real transactions running in practical systems. In particular, transactions are defined explicitly by a set of steps and not by a program whose concrete execution on a certain state of the database will define the transaction. Assume a program to process all data items which fulfill a certain predicate p. A transaction t resulting from executing the program will read all data items for which p is true. To guarantee serializability, in advance to reading such a data item, the item has to be locked. If concurrently, however later before t has finished, another transaction t' inserts a new data item x into the database which also fulfills predicate p, t has neither read nor locked x. In such situations serializability cannot be guaranteed by 2PL as described so far, because the relevant set of data items has not been locked by t. This problem is known as the phantom problem. Practical systems have found ways to live with phantoms. Instead of only locking data items, locking is applied on the set of objects fulfilling a predicate, on complete relations or index intervals.

Key Applications

For organizations of any kind database systems have become indispensable to maintain the operational data. Applications in this context typically are dominated by a potentially huge number of rather short transactions. In such a setting, commonly called online transaction processing (OLTP), reliable and efficient processing of transactions is required. 2PL has become a de facto standard in database systems for OLTP applications. However, 2PL is based on blocking transactions and therefore system throughput may be severely affected by the locking protocol. Fortunately, over the years several guidelines have been developed [3,7,11] which help a database administrator to tune the programming of the transactions and the implementation of 2PL in a way such that the requisite efficiency of the overall system can be achieved.

Cross-references
▶ ACID Properties
▶ Concurrency Control – Traditional Approaches
▶ Locking Granularity and Lock Types
▶ Multi-Version Serializability and Concurrency Control
▶ Serializability
▶ SQL Isolation Levels
▶ Transaction Chopping
▶ Transaction Models – The Read/Write Approach

Recommended Reading
1. Agrawal D. and Abbadi A.E. Constrained shared locks for increased concurrency in databases. J. Comput. Syst., Sci.51:(1) 53–63, 1995.
2. Bayer R., Heller H., and Reiser A. Parallelism and recovery in database systems. ACM Trans. Database Syst., 5:(2)139–156, 1980.
3. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for Systems Professionals. Morgan Kaufmann, San Francisco, CA, 1996.

4. Bernstein P.A., Shipman D.W., and Wong W.S. Formal Aspects of Serializability in Database Concurrency Control. IEEE Trans. Software Eng., SE-5:203–215, 1979.

5. Boral H. and Gold I. Towards a Self-adapting Centralized Concurrency Control Algorithm. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 18–32.

6. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notion of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 1976.

7. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.

8. Lausen G. Concurrency Control in Database Systems: A Step towards the Integration of Optimistic Methods and Locking. In Proc. ACM Annual Conf., 1982, pp. 64–68.

9. Papadimitriou C.H. The Serializability of Concurrent Database Updates. J. ACM, 26:631–653, 1979.

10. Salem K., Garcia-Molina H., and Shands J. Altruistic locking. ACM Trans. Database Syst., 19:(1)17–165, 1994.

11. Shasha D. Database Tuning – A Principled Approach. Prentice-Hall, USA, 1992.

12. Weikum G. and Vossen G. 1Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Two-Poisson model

Giambattista Amati
Ugo Bordoni Foundation, Rome, Italy,

Synonyms

Harter's model; Probabilistic model of indexing

Definition

The 2-Poisson model is a mixture, that is a linear combination, of two Poisson distributions:

$$\text{Prob}(X = \mathbf{tf}) = \alpha \frac{\lambda^{\mathbf{tf}} e^{-\lambda}}{\mathbf{tf}\,!} + (1 - \alpha) \frac{\mu^{\mathbf{tf}} e^{-\mu}}{\mathbf{tf}\,!} \quad [0 \le \alpha \le 1]$$

In the context of IR, the 2-Poisson is used to model the probability distribution of the frequency X of a term in a collection of documents.

Historical Background

The 2-Poisson model was given by Harter [5–7], although Bookstein [2,1] and Harter had been exchanging ideas about probabilistic models of indexing during those years. Harter coined the word "elite" to introduce his 2-Poisson model [5, pp. 68–74].

The origin of the 2-Poisson model can be traced back through all Luhn, Maroon, Damerau, Edmundson and Wyllys [3,4,5,6]. The first accounts on Poisson distribution modeling the stochastic behavior of functional words were given by Stone, Rubinoff and Damerau [3,11]. Stone, Rubinoff and Damerau observed that the words that have only a functional role in the text, can be modeled by a Poisson distribution.

Foundations

The 2-Poisson model is a probabilistic model of indexing, rather than a document retrieval model. The purpose of Harter's work is to identify the key-words likely to be informative for an arbitrary document that can be selected to build an index for a collection. Such words are called *specialty* words by Harter in contraposition to the other ones, the *non-specialty* ones, which instead are considered to occur at random in documents. In the works by Luhn, Maroon, Damerau, Edmundson and Wyllys it was observed that the divergence between the rare usage of a word across the document collection and the contrasting relative within-document frequency constitutes a revealing indication of the informative status of a word. Damerau suggests selecting the class of high status words of the index by making the assumption that Poisson distribution describes frequencies of words that are in the complementary class. If the Poisson probability of a term within a document is very small, then the word is marked as an index term. Obviously, not all words clearly fall either into one class or into the other. But nonetheless, many word tokens occur randomly in many documents while the same word tokens occur more densely and nonrational in a few documents. This set of documents called the *Elite set* of the term $\mathbf{t}$ is very likely to be the set of documents which extensively connects with the concept or the semantics related to the term. The Elite set $\mathbf{E_t}$ attracts the tokens with an expected rate $\lambda_{\mathbf{E_t}}$. Tokens fall randomly into the other documents with a lower rate $\lambda_{\overline{\mathbf{E_t}}}$. The final probability of occurrence of the term in any document is given by the mixture of these two Poisson distributions:

$$\text{Prob}(X = \mathbf{tf}) = \alpha \cdot \frac{e^{-\lambda_{\mathbf{E_t}}} \lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf!}} + (1 - \alpha) \cdot \frac{e^{-\lambda_{\overline{\mathbf{E_t}}}} \lambda_{\overline{\mathbf{E_t}}}^{\mathbf{tf}}}{\mathbf{tf!}} \quad (1)$$

The probability of term **t** to appear **tf** times in a document **d** belonging to the Elite set $\mathbf{E_t}$ is given by the conditional probability

$$\mathrm{Prob}(\mathbf{d} \in \mathbf{E_t} | X = \mathbf{tf}) = \frac{\mathrm{Prob}(X = \mathbf{tf}, \mathbf{d} \in \mathbf{E_t})}{\mathrm{Prob}(X = \mathbf{tf})}$$

$$= \frac{\alpha \cdot \frac{e^{-\lambda_{\mathbf{E_t}}} \lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf}!}}{\alpha \cdot \frac{e^{-\lambda_{\mathbf{E_t}}} \lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf}!} + (1 - \alpha) \cdot \frac{e^{-\lambda_{\overline{\mathbf{E_t}}}} \lambda_{\overline{\mathbf{E_t}}}^{\mathbf{tf}}}{\mathbf{tf}!}}$$

$$(2)$$

Thus, in the case that the word belongs to the non-specialty class:

$$\mathrm{Prob}(\mathbf{d} \in \mathbf{E_t} | X = \mathbf{tf}) = \frac{1}{1 + \beta \cdot e^{\left(\lambda_{\mathbf{E_t}} - \lambda_{\overline{\mathbf{E_t}}}\right)} \left(\frac{\lambda_{\overline{\mathbf{E_t}}}}{\lambda_{\mathbf{E_t}}}\right)^{\mathbf{tf}}} \quad (3)$$

where $\beta = \frac{1-\alpha}{\alpha}$. The conditional probability of (3) is used by Harter to generate a ranking of the most informative words. However, the 2-Poisson model requires the estimation of three parameters for each word of the vocabulary, and this is a real drawback for any direct practical application of his model to term selection or term-weighting problems.

A last remark concerns the **N**-Poisson model, the generalization of the 2-Poisson model. Any probability distribution on $(0, \infty)$ can be defined as a mixing distribution of Poisons [10]. Therefore, it is true that every word follows a **N**-Poisson distribution for some N. **N**-Poisson models thus have a practical application only when N is small, that is the 2-Poisson model or the 3-Poisson model.

The 2-Poisson is illustrated with an example. A collection of ten documents contains a word **t** occurring respectively 4, 5, 6, 0, 1, 2, 0, 0, 0, 0 times within these ten documents, the first three documents having the highest frequency of term in the collection. A useful image for this configuration could be that the first three documents constitute an *elite* or *special* subset of documents for the term. Then, these data are fitted to the 2-Poisson model using the Expectation Maximization (EM) algorithm. To set the initial values α_0, λ_0 and μ_0 for the first step of the EM algorithm the assumption is that there is a Poisson distribution generating the frequency in the elite set of the word, with a mean term-frequency $\hat{\mu}_0 = \frac{4+5+6}{3} = 5$, and a second Poisson distribution generating the term-frequency in the rest of the collection with

$\hat{\lambda}_0 = \frac{0+1+2+0+0+0+0}{7} = 0.4287$. Observe that the elite set is $\frac{3}{10} = 0.3333$ of the entire collection, and thus $\hat{\alpha}_0 = 0.3333$ is initially set. Finally, the EM algorithm converges to the values $\hat{\alpha} = 0.35$, $\hat{\mu} = 4.5$ and $\hat{\lambda} = 0.54$ with a confidence of 0.99. The probability that a document **d** belongs to the Elite set of a term occurring **tf** times in **d** is

$$\mathrm{Prob}(\mathbf{d} \in \mathbf{E_t} | X = \mathbf{tf}) = \frac{1}{1 + 1.86 \cdot e^{3.96} 0.12^{\mathbf{tf}}}$$

Key Applications

The 2-Poisson model is at the basis of two types of probabilistic models of IR: the BM25 model and the Divergence From Randomness models.

Cross-references

▶ BM25
▶ Divergence from Randomness Models

Recommended Reading

1. Bookstein A. and Kraft D. Operations research applied to document indexing and retrieval decisions. J. ACM, 24(3):418–427, 1977.
2. Bookstein A. and Swanson D. Probabilistic models for automatic indexing. J. Am. Soc. Inform. Sci., 25:312–318, 1974.
3. Damerau F. An experiment in automatic indexing. Am. Doc., 16:283–289, 1965.
4. Edmundson H.P. and Wyllys R.E. Automated abstracting and indexing–survey and recommendations. Commun. ACM, 4(5):226–234, May 1961. Reprinted in Readings in Information Retrieval, pp. 390-412. H. Sharp (ed.). New York, NY: Scarecrow; 1964.
5. Harter S.P. A probabilistic approach to automatic keyword indexing. PhD thesis, Graduate Library, The University of Chicago, Thesis No. T25146, 1974.
6. Harter S.P. A probabilistic approach to automatic keyword indexing. part I: On the distribution of specialty words in a technical literature. J. American Soc. for Inf. Sci., 26:197–216, 1975.
7. Harter S.P. A probabilistic approach to automatic keyword indexing. part II: An algorithm for probabilistic indexing. J. American Soc. for Inf. Sci., 26:280–289, 1975.
8. Luhn H.P. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1:309–317, 1957.
9. Maron M.E. Automatic indexing: an experimental inquiry. J. ACM, 8:404–417, 1961.
10. Puri P.S. and Goldie C.M. Poisson mixtures and quasi-infinite divisibility of distributions. J. Appl. Probab., 16(1):138–153, 1979.
11. Stone D. and Rubinoff B. Statistical generation of a technical vocabulary. Am. Doc., 19(4):411–412, 1968.

Two-Sorted First-Order Logic

▶ Temporal Relational Calculus

Type Theory

▶ Data Types in Scientific Data Management

Type-based Publish/Subscribe

HANS-ARNO JACOBSEN
University of Toronto, Toronto, ON, Canada

Definition
Type-based publish/subscribe is an instance of the publish/subscribe concept, where publications are instances of application-defined types, subscriptions express interest in receiving publications of a specified type or sub-type, and publish/subscribe matching amounts to type conformance checking.

Key Points
The characterization of the type-based publish/subscribe class originated in the programming languages context with the objective of bridging the impedance mismatch in the integration of publish/subscribe abstractions into programming languages. For example, the representation of subscriptions as strings that are parsed, checked, and processed at runtime by the underlying publish/subscribe implementation, are frequent sources of errors that materialize too late due to the inability to properly type check the subscription string at the language level. To enable static type checking, type-based publish/subscribe includes subscriptions and publications as first class citizens into the language.

Publications become instances of language types. Arbitrary composite types, subject to the type definition capabilities of the host language, can be used to model application-defined events. Subscribers express interest in receiving publications of a specified type. Matching of publications against subscriptions amounts to type conformance checking. That is a publication matches a subscription, if the type of interest specified by a subscription conforms to the type of a publication, for some definition of type conformance. Matching is based on type determination, sub-type checking, and is-instance-of type checks. This is similar to topic-based publish/subscribe matching, except that a type is more general than a topic. Moreover, types may support operations that can model content-based predicates as tests on the instances of types. In this sense, type-based publish/subscribe can model content-based processing.

Several standards, such as the OMG Notification Service [2] and the OMG Data Dissemination Service [3] exhibit elements of type-based publish/subscribe, but do not directly follow the prescription of the type-based publish/subscribe model. The characterization of type-based publish/subscribe can be found in [1].

Cross-references
▶ Publish/Subscribe
▶ Topic-Based Publish/Subscribe

Recommended Reading
1. Eugster P. Type-based publish/subscribe: concepts and experiences. ACM Trans. Program. Lang. Syst., 29(1):6, 2007.
2. MG. Notification Service Specification, version 1.1, formal/04–10–11 edition, October 2004.
3. MG. Data Distribution Service for Real-time Systems, version 1.2, formal/07–01–01 edition, January 2007.