The TSQL Benchmark*

Christian S. Jensen (editor) James Clifford Shashi K. Gadia Fabio Grandi Patrick P. Kalua Nick Kline Angelo Montanari Sunil S. Nair Elisa Peressi Barbara Pernici Edward L. Robertson John F. Roddick Nandlal L. Sarda Maria Rita Scalas Arie Segev Richard T. Snodgrass Abdullah Tansel Paolo Tiberio Alexander Tuzhilin Gene T. J. Wuu

Abstract

This document presents the temporal database community with an extensive, consensus benchmark for temporal query languages. The benchmark is semantic in nature. It is intended to be helpful when evaluating the user-friendliness of temporal query languages, including proposals for the consensus temporal SQL that is currently being developed.

The benchmark consists of a database schema, an instance for the schema, and a set of queries on the this database. The queries are classified according to a taxonomy, which is also part of the benchmark.

*Correspondence may be directed to the TSQL electronic mail distribution, tsql@cs.arizona.edu, or to the editor at Aalborg University, Datalogi, Fr. Bajers Vej 7E, DK–9220 Aalborg Ø, Denmark, csj@iesd.auc.dk. Affiliations and e-mail addresses of the authors follow. J. Clifford, Information Systems Dept., New York University, jcliffor@is-4.stern.nyu.edu; S. K. Gadia, Computer Science Dept., Iowa State University, gadia@cs.iastate.edu; F. Grandi, Dip. di Elettronica Informatica e Sistemistica, Università di Bologna, Italy, fabio@deis64.cineca.it; P. P. Kalua, Computer Science Department, Indiana University, kalua@cs.indiana.edu; N. Kline, Computer Science Dept., University of Arizona, kline@cs.arizona.edu; A. Montanari, Dip. di Matematica e Informatica, Università di Udine, montanari@uduniv.cineca.it; S. S. Nair, Computer Science Department, Iowa State University, snair@cs.iastate.edu; E. Peressi, Dip. di Matematica e Informatica, Università di Udine, peressi@udmi5400.cineca.it; B. Pernici, Dip. di Matematica e Informatica, Università di Udine, pernici@uduniv.cineca.it; E. L. Robertson, Computer Science Department, Indiana University, edrbtsn@cs.indiana.edu; J. F. Roddick, School of Computer and Information Science, University of South Australia roddick@unisa.edu.au; N. L. Sarda, Computer Science and Eng. Dept., Indian Institute of Technology, Bombay, India, nls@cse.iitb.ernet.in; M. R. Scalas, Dip. di Elettronica Informatica e Sistemistica, Università di Bologna, Italy, rita@deis64.cineca.it; A. Segev, School of Business Adm. and Computer Science Research Dept., University of California, segev@csr.lbl.gov; R. T. Snodgrass, Computer Science Dept., University of Arizona, rts@cs.arizona.edu; A. Tansel, Bernard M. Baruch College, City University of New York, UZTBB@CUNYVM.CUNY.EDU; P. Tiberio, Dip. di Elettronica Informatica e Sistemistica, Università di Bologna, Italy, tiberio@deis64.cineca.it. A. Tuzhilin, Information Systems Dept., New York University, tuzhilin@square1.stern.nyu.edu; G. T. J Wuu, Bell Communications Research, wuu@ctt.bellcore.com.

1 Introduction

The central goal of this document is to provide the temporal database community with a *comprehensive consensus benchmark* for temporal query languages that is *independent* of any existing language proposal.

This is not a performance benchmark, but is rather a *semantic* benchmark intended to be an aid in evaluating the user-friendliness of proposals for temporal query languages. Thus, temporal query languages should ideally be able to express the benchmark queries both conveniently and naturally.

To obtain a consensus benchmark, researchers in temporal databases were invited to participate in this initiative, and each researcher that contributed significantly is a coauthor. The electronic mail distribution tsql@cs.arizona.edu was used as the medium for discussing the benchmark and related issues.

The benchmark consists of a database schema, an instance for the schema, and a set of queries on the this database. The queries are classified according to a taxonomy, which is also part of the benchmark. As a consequence of the central goal above, no existing temporal data models are used or mentioned. The relation schemas of the benchmark are expressed as sets of attributes, including one attribute illustrating user-defined time. However, the underlying temporal aspects are implicit (of course, specific temporal data models might add explicit temporal attributes). The contents of the relations are described in natural language. The benchmark queries are also given only in natural language. The taxonomy is independent of any particular temporal query language.

The benchmark is *not* intended to constitute a metric for query language completeness, and as such it is not a substitute for a rigorous *theoretical* study of expressive powers of various temporal query languages.

This paper appeared in Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, June 14-16, 1993, pp. QQ-1-QQ-28.

Comprehensiveness of the benchmark is desirable only to ensure that a wide range of query language design aspects are covered.

It it emphasized that using the benchmark as an advanced, quantitative scoring system for comparing languages makes little sense. Thus, one language is not necessarily superior to another just because one is capable of expressing more benchmark queries than the other. Rather, the focus is on user-friendliness.

The presentation is structured as follows. Below, the intended scope of the benchmark is defined. Sections 3, 4, and 5 first state criteria for what should be required from a suitable database schema and instance and classification scheme, respectively. Second, an actual schema, instance, and classification schema is presented. The main body of the paper is Section 6, which presents, using the classification scheme, approximately 170 benchmark queries. Comments related to this benchmark, by identified contributors, are included as appendices at the end.

2 Scope

Within certain boundaries, discussed next, the benchmark is intended to contain all queries that appear reasonable and that are consistent with the schema and data of the benchmark.

First, the benchmark is of a semantic nature—in its current form, it is not aimed at performance comparisons. The intention is to provide a foundation for comparing the descriptive and operational characteristics and capabilities of temporal data models and query languages, not their performance characteristics. Properly extended with additional relation schemas and a variety of large instances, the benchmark can also be used for performance comparisons.

Second, a number of restrictions are imposed on which types of queries are admissible in this version of the benchmark, including the following.

- Queries are restricted to valid time only. Transaction-time related queries are not explored.
- Schema evolution and versioning are not considered.
- Incompleteness is not considered.
- Recursive queries are not included.
- General temporal reasoning is beyond the scope of this version of the benchmark.
- Queries involving aggregation facilities are not considered.
- Only queries are included—updates are not considered.
- Continuous attributes such as time are not included.
- The querying of data valid in the future is not explored.

These advanced aspects are excluded solely for pragmatic reasons, and the exclusion is not meant to imply in any way that the aspects are not important. The restrictions simply represent an attempt to reduce the size of the initial benchmark to manageable proportions.

It is emphasized that this benchmark is merely the first in a sequence of ever-more comprehensive benchmarks. Later benchmarks will relax the above restrictions on the scope of comprehensiveness imposed on this benchmark. Specifically, the next version of the benchmark is intended to include queries that involve aggregation.

3 The Benchmark Database Schema

3.1 Criteria

A suitable database schema for a semantic benchmark satifies four criteria.

- The schema should be natural. That is, it should correspond to a reasonable, though possibly greatly simplified, segment of the real world. This both reduces the need to explain the model and enhances the ability to recognize verball pitfalls in the path to the query instances.
- The schema should be simple. This will aid in making the benchmark easy to understand. This criterion restricts the number of relation schemas and the number of attributes of the individual schemas. Additionally, the names of the relations and of the attributes should be short, as they will be referenced repeatedly.

  When an expansion is proposed, the benefits should be carefully compared with the added complexity.
- The schema should allow for comprehensiveness within the chosen scope. Using the schema, it should be possible formulate queries of all the types that appear reasonable.

  This indicates a need for at least two related relation schemas (for natural-join queries).

- A schema that has already been used frequently is preferred over a new schema. This guarantees that many existing queries can be adapted easily to the benchmark.
- For clarity, schema and attribute names must start with capital letters.

3.2 The Schema

The database schema consists of three valid-time relation schemas, `Emp`, `Skills`, and `Dept`. They are defined as follows.

Relation `Emp` uses the attributes `Name`, `Salary`, and `Dept` for recording the salaries of employees and the departments where they work. In addition, it contains attributes `Gender` and `D-birth` which indicate the gender and date of birth of an employee. While the name, salary, and department of an employee varies over time, both `Gender` and `D-birth` are time-invariant.

Relation `Skills` records the association of employees with skills via the two attributes `Name` and `Skill`. The skills of an employee may vary over time. For example, employees are considered to have the skill "driving" only during those interval(s) when they hold valid licenses.

Relation `Dept` records the association of employees, as managers, with departments, and it contains three attributes, `Department`, recording a department name, `Manager`, recording the manager of the department, and `Budget`, recording the budget of the department.

Attributes `Name`, `Dept`, `Department`, `Skill`, and `Manager` are of type `textString`; attribute `Gender` is one of `F` (female) and `M` (male); `Salary` and `Budget` are of type `integer`; and `D-birth` is a user-defined time value which may be compared with valid times.

The relation schemas obey the following *snapshot* functional and multivalued dependencies:

For `Emp`:
- $\texttt{Name} \rightarrow \texttt{Salary}$
- $\texttt{Name} \rightarrow \texttt{Dept}$
- $\texttt{Name} \rightarrow \texttt{Gender}$
- $\texttt{Name} \rightarrow \texttt{D-birth}$

For `Skills`:
- $\texttt{Name} \twoheadrightarrow \texttt{Skill}$ (and $\texttt{Name} \not\rightarrow \texttt{Skills}$)

For `Dept`:
- $\texttt{Department} \rightarrow \texttt{Manager}$
- $\texttt{Manager} \rightarrow \texttt{Department}$
- $\texttt{Department} \rightarrow \texttt{Budget}$

Note that `Name` is the primary key of `Emp` (it is the only candidate key). For `Skills`, there is no non-trivial key. For `Dept`, each of `Department` and `Manager` is a candidate key, and `Department` is selected as the primark key.

Each of the relation schemas are in snapshot Boyce-Codd normal form.

It is emphasized that the notion of key does not capture correspondence between attribute values and the real-world objects they represent. As one consequence, it is possible in this schema, e.g., for a person to change `Name` attribute value over time.

The attribute `Manager` of `Dept` is a foreign key for the attribute `Name` of `Emp`. Thus, a tuple is allowed to exist in the `Dept` relation only if, for each non-empty snapshots of this tuple, the `Manager` attribute value exists as a `Name` value of some tuple in the simultaneous snapshot of the `Emp` relation.

4 The Benchmark Data

4.1 Criteria

- For clarity, the database instance should ideally accord with *all and only* those constraints which are explicitly stated in the definition of the database schema.
- For simplicity and ease of typing, attribute values should be short and salary values should be multiples of $10,000.
- Transitions (i.e., timestamp values) occur only at the beginning of the month, and all dates should be in the time interval from 1/1/81 to 12/31/88 (because the digits 8 and 9 are relatively hard to distinguish). Time intervals are all specified by the inclusive starting and ending events. Also for clarity, relation instance names should start with lowercase letters.
- The data should include a "hole in the history" of some entity. For example, the database may be designed to contain a whole in the employment of some employee.
- The data should include asynchronous behavior of attributes. For example, the department and salary of employees may change independently.

4.2 The Data

Three instances, `emp`, `skills`, and `dept`, are defined over the `Emp`, `Skills`, and `Dept` relation schemas, respectively. The contents of these instances is described below.

There are two employees, identified by *ED* and *DI* in the following.

ED worked in the Toy department from 2/1/82 to 1/31/87, and in the Book department from 4/1/87 to the present. His name was Ed from 2/1/82 to 12/31/87, and Edward from 1/1/88 to the present. His salary was $20K from 2/1/82 to 5/31/82, then $30K from 6/1/82 to 1/31/85, then $40K from 2/1/85 to 1/31/87 and 4/1/87 to the present. *ED* is male and was born on 7/1/55. Several skills are recorded for *ED*. He has been qualified for typing since 4/1/82 and qualified for filing since 1/1/85. He was qualified for driving from 1/1/82 to 5/1/82 and from 6/1/84 to 5/31/88.

DI worked in and managed the Toy department from 1/1/82 to the present. Her name is Di throughout her employment. The budget of the Toy department was $150K from 1/1/82 to 7/31/84, $200K from 8/1/84 to 12/31/86, and $100K from 1/1/87 to the present. Her salary was $30K from 1/1/82 to 7/31/84, $40K from 8/1/84 to 8/31/86, then $50K from 9/1/86 to the present. *DI* is female and was born on 10/1/60. *DI* has been qualified for directing from 1/1/82 to the present.

The present time (i.e., the value of `now`) is 1/1/90.

5 Classification of Benchmark Queries

A classification of benchmark queries will be based on a comprehensive taxonomy of queries. First, critria for such a taxonomy are outlined. Next, the taxonomy itself is presented. As the taxonomy is too fine-grained, categories are then merged into an adequate number of groups which can subsequently be used for classification.

5.1 Criteria

Three criteria for an appropriate taxonomy of benchmark queries are suggested.

- The taxonomy should be schema and instance independent. This criterion helps ensure that the taxonomy will persist when the benchmark database schema evolves as new versions appear. Ideally, this will allow for an incremental mode of work, where only new queries need to be categorized and existing queries do not need re-categorization.

- The taxonomy should provide comprehensive coverage of benchmark queries. Comprehensiveness is desirable to avoid holes and point to many categories of queries.

- The taxonomy should be useful when structuring the presentation of benchmark queries. Most importantly, it should provide sufficient structure. Thus, taxonomies that have only few categories and that map many queries to single categories are problematic. If the number of categories is excessive for presentation purposes, classes of categories may be identified with individual sections.

5.2 The Taxonomy

The taxonomy is characterized as having a projection (output) and a selection component, much like SQL. Then each component is covered in turn. Finally, the full taxonomy is summarized and a notation for naming individual categories is defined.

5.2.1 Top-level Taxonomy

At the top level, the taxonomy is divided into two orthogonal parts, namely a part where queries are categorized according to their *output component* and a part where the categorization is based on the *selection component*. Thus, a category is described by two components, as illustrated in Figure 1.

$$\{<\text{output component}>\} \times \{<\text{selection component}>\}$$

Figure 1: Top-level Description of Benchmark Taxonomy

This top-level design reflects the SQL template (i.e., `SELECT` ...`FROM` ...`WHERE` ...). The first component categorizes the contents of the `SELECT` clause, and the second component categorizes the contents of the `WHERE` clause. No component is needed to reflect the `FROM` clause where tuple variables are defined. The two components are orthogonal only in the same sense that the `WHERE` and `SELECT` clauses of a particular query are orthogonal.

5.2.2 Output-based Taxonomy

The output-based taxonomy is intended to reflect the part of queries where the format of the resulting tuples is specified. The taxonomy is described in Figure 2 and is explained in the following.

The idea is to distinguish between queries based on the format of the result tuples. A tuple may include

$$\left\{\begin{array}{c}\underline{\text{explicit-attribute component}}\\ \text{none}\\ \text{projected}\\ \text{complete}\end{array}\right\} \times \left\{\begin{array}{c}\underline{\text{valid-time component}}\\ \text{none}\\ \left\{\begin{array}{c}\underline{\text{type}}\\ \text{event}\\ \text{interval}\\ \text{element}\end{array}\right\} \times \left\{\begin{array}{c}\underline{\text{value}}\\ \text{derived}\\ \text{imposed}\end{array}\right\}\end{array}\right\}$$

Figure 2: Output-based Taxonomy

an explicit-attribute component and a valid-time component, each of which are considered next.

If present, the explicit-attribute component, may contain all attributes in the argument relation (multiple relations are discussed below) or it may contain a subset of the attributes in the argument relation. In the first case, the explicit attribute component is "complete," and in the second, it is "projected."

To exemplify, consider a tuple telling that Ed is in the Book department from 1/1/82 to 12/31/84. Here "Ed" and "Book" constitute the explicit-attribute component, and "1/1/82" and "12/31/84" is the valid-time component. If the argument relation contained an attribute "Salary" in addition to the Name and Department attributes, this result is projected.

If several relations are used in a query, the argument relation is the Cartesian product of these, i.e., the schema is the concatenation of the schemas of the relations used in the query.

The valid-time component of a tuple may be of three types. First, it may be an event, i.e., a single time value (e.g., 3/1/83). Second, it may be an interval, i.e., a sequence of consecutive time values (e.g., as above). Third, it may be an element, i.e., a set of time values which may be described by a set of intervals (e.g., 1/1/82 to 12/31/84, 2/1/85 to 3/31/85, and 5/1/86 to 5/31/86).

Orthogonally, the value of a valid-time component may be derived or imposed. A derived value is computed solely in terms of the valid-time components of the tuples in the argument relation. An imposed value is computed by explicit assignment in the query.

Note that at least one of the two components must be present in the result of a query. This part of the taxonomy results in 20 mutually exclusive categories.

The distinctions above are based on the schema of result relations. It is possible also to distinguish between the cardinalities of result relations, e.g., between set-valued and single-tuple valued results.

5.2.3 Selection-based Taxonomy

The selection component is divided into two parts, one for valid-time selection and one for selection not involving valid time. See Figure 3.

$$\{<\text{valid-time selection}>\}^{*} \times \{<\text{non-temporal selection}>\}^{*}$$

Figure 3: Top-level Selection-based Taxonomy

Both parts are based on the same observation. In general, a selection predicate is built from atomic selection predicates using logical operators (e.g., `and`, `or`, and `implies`) and parenthesis. Both parts categorize queries based on the atomic predicates used in the queries. As several types of atomic predicates may be used in the same query, queries generally fall into multiple categories (as indicated in Figure 3 by the Kleene star, "*"). We examine each part of the selection-based taxonomy in turn.

Atomic valid-time selection predicates are assumed to be of the form

$$arg_1 \ \texttt{op} \ arg_2 \ ,$$

where `op` is a some comparison operator (e.g., `precedes`, and `contains`). It is assumed that arg_1 is the valid time of the data, and restrictions are imposed based on the type of the comparison operator, on the origin of arg_2, and on the type of arg_2. Figure 4 outlines the categories.

Three types of comparison operators are identified. First, a comparison operator may be duration-based. For example the operator `spanExceeds` returns true if the duration of the first argument is equal to or larger than the duration of the second argument. Second, comparison operators may be based on ordering. Operators in this category include `precedes` and `meets`. The first applies to all timestamps and evalutes to true if the largest time in the first argument is smaller

$$\left\{\begin{array}{c}\underline{\text{type of comparison operator}}\\ \text{duration-based}\\ \text{ordering-based}\\ \text{containment-based}\end{array}\right\} \times \left\{\begin{array}{c}\underline{\text{type of } arg_2}\\ \text{event}\\ \text{interval}\\ \text{element}\end{array}\right\} \times \left\{\begin{array}{c}\underline{\text{origin of } arg_2}\\ \text{explicitly supplied in query}\\ \text{user-defined attribute value}\\ \text{computed from other valid times}\end{array}\right\}$$

Figure 4: Valid-time Selection-based Taxonomy

than the smallest times in the second argument. Operator `meets` appears to be useful only for events and intervals. Two timestamps meet if they are not separated by any event (i.e., may be coalesced). Operators based on containment include `=` (identity), `overlaps`, and `contains`.

The second argument (arg_2) may be an event, an interval, or an element. Also, it may come from three sources. First, it may be supplied directly in the query, as a constant. Second, it may be the value of a user-defined time attribute in an argument tuple. Note that this is only possible for events if first normal form is required. Third, like the first argument, the second argument may be computed from valid times in the argument tuples.

If the three types of categories are completely orthogonal, this part of the taxonomy will contribute with a total of 27 categories. However, it may be debated whether intervals and elements may be used as values of user-defined attributes (resulting in non-1NF relations).

The final part of the selection-based taxonomy categorizes queries based solely on the part of the selection component that involves only ordinary, non-temporal selection.

Many possibilities for categorization exist. Below, in Figure 5, we distinguish between four significant types of atomic selection predicates. First, an attribute may be compared with a constant, supplied by the user. Second, attribute values, both in the same relation, may be compared. Third, a primary key value may be compared with a matching foreign key value. Fourth, arbitrary attributes of possibly distinct relations may be compared. In the figure, $\theta ::= < \mid > \mid \leq \mid \geq \mid =$, i.e., a combination of equality and/or the one of the two inequality operators. If we distinguish between situations where only equality is involved and situations where inequality is involved, this give 8 categories.

5.3 Overview and Naming of Categories

Each query has a single output component, zero or more valid-time selection components (one per such operator), and zero or more non-temporal selection-based components (one per such operator). The taxonomy is summarized in Figure 6. There, the names introduced in the taxonomy are used along with punctuation in order to name a category.

To exemplify the use of Figure 6 for naming categories, consider the query "When was Ed Manager of the Toy Department." This query is in the category shown next (with no valid-time selection).

(None, Element, Derived) // (=, Constant) (=, Constant)

It may be observed that the taxonomy gives rise to a large number of categories. For example, assuming a single non-temporal operator and no valid-time operators, there are $20 \times 8 = 160$ categories. Adding a single valid-time operator while assuming orthogonality yields an additional 4320 categories!

As a result, it becomes necessary to create classes of categories which then may be used for clasifying the benchmark queries.

One approach would be to name a *class* of categories of queries, by simply replacing one or more of the entries with the Kleene star ("*"), e.g.,

(None, Element, Derived) / (*,*,*) / (=, Constant)

The above query category would be in this class. In the next section, we define the classes to be used in the benchmark.

5.4 Forming Classes from Categories

The idea is to remove distinctions from the comprehensive taxonomy until a suitable number of classes is obtained. Figure 7 is thus a reduced version of Figure 6.

The second and third lines concern output. Only the prescence or absence of explicit attributes and

$$\left\{\begin{array}{c}\underline{\text{non-temporal attribute value selection}}\\ att\ \theta\ Constant \\ att_1\ \theta\ att_2 \\ att_k\ \theta\ att_{fk} \\ att(rel_1)\ \theta\ att(rel_2)\end{array}\right\} \times \left\{\begin{array}{c}\underline{\text{comparison operator, }\theta}\\ \text{only equality }(=)\\ \text{inequality }(<>)\end{array}\right\}$$

Figure 5: Non-temporal Selection-based Taxonomy

```
<category>         ::= <output> '/' {<v-t selection> }* '/' {<non-t selection> }*
<output>           ::= '(' {None | Projected | Complete } ','                 /* explicit-attribute component
                       {None |                                                /* no valid-time attribute
                        {Event | Interval | Element } ','                     /* type of valid-time attribute
                        {Derived | Imposed } } ')'                            /* value of valid-time attribute
<v-t selection>    ::= '(' {Duration | Ordering | Containment } ','           /* operator type
                       {Event | Interval | Element } ','                      /* argument type
                       {Explicit | User-defined | Computed } ')'              /* argument origin
<non-t selection>  ::= '(' {'=' | '<>' } ','                                  /* operator type
                       {Constant | Single | Foreign | Arbitrary } ')'         /* argument types
```

Figure 6: Overview of the Taxonomy used for Naming Categories

```
<class-name>              ::= <reduced output> '/' {<reduced v-t selection> }*
<reduced output>          ::= '(' {None | Proj/Comp } ','            /* explicit-attribute component
                                  {None | Not empty } ')'           /* valid-time attribute component
<reduced v-t selection>   ::= '(' {Duration | Other } ','           /* comparison operator type
                                  {Event | Interval | Element } ',' /* argument type
                                  {Computed | Other } ')'           /* argument origin
```

Figure 7: Overview of the Classification of Queries

timestamps are distinguished, leading to three categories. The last three lines concern valid-time selection (non-temporal selection is disregarded). Comparison operators may be duration-based or not; arguments be of either event, interval, or element type; and the arguments may or may not derive from valid times of tuples.

6 The Benchmark Queries

6.1 Overview

The structure of this section is based on Figure 7. There are three sections, each of which contains ten sections. The three top-level sections classify queries according to the output. Thus, the output from a query may have either only an explicit-attribute value component (O1), only a valid-time component (O2), or it may have both (O3).

Each top-level section contains the same ten sections. These divide queries based on a single, distinguished valid-time selection predicate[1]. The predicate is of the format arg_1 `op` arg_2 where `op` is a comparison operator which may (or may not) be duration-based. One argument is the valid time of tuples in the argument relation. The other argument may be of event, interval, or element type; orthogonally, it may (or may not) be computed from existing valid times in the argument relation. This results in a total of ten classes (the combination of duration-based predicates and event arguments is omitted).

(S1)	(Duration, Interval, Computed)
(S2)	(Duration, Interval, Other)
(S3)	(Duration, Element, Computed)
(S4)	(Duration, Element, Other)
(S5)	(Other, Event, Computed)
(S6)	(Other, Event, Other)
(S7)	(Other, Interval, Computed)
(S8)	(Other, Interval, Other)
(S9)	(Other, Element, Computed)
(S10)	(Other, Element, Other)

6.2 Explicit-attribute Output

This section involves queries which return only explicit-attribute values—no valid-time values are present in the result.

[1]A query may contain several selection predicates, but it is classified according to a single predicate.

6.2.1 Class O1.S1 (Duration, Interval, Computed)

Query Q 2.1.1: Which departments had managers who served for the shortest continuous period?
Answer: "Toy"
Category: (Projected, None) / (Duration, Interval, Computed) /

Here is a case where zero spans should be ignored.

Query Q 2.1.2: Who worked continuously in the Book department for as long as Di did?
Answer: "Ed" and "Edward"
Category: (Projected, None) / (Duration, Interval, Computed) / (=, Constant) (=, Constant)

Di never worked in the Book department, so this should return everyone that ever worked in the Book department. Here is a case where zero spans are significant.

Query Q 2.1.3: Who worked continuously in the Toy department for as long as Di did?
Answer: "None"
Category: (Projected, None) / (Duration, Interval, Computed) / (=, Constant) (=, Constant)

Query Q 2.1.4: Who worked continuously in a department longer than their current manager worked in that department?
Answer: "None"
Category: (Projected, None) / (Duration, Interval, Computed) (Containment, Event, Explicit) /

Query Q 2.1.5: Who had the same salary for the longest continuous time period?
Answer: "Di"
Category: (Projected, None) / (Duration, Interval, Computed) /

Query Q 2.1.6: Who worked for a manager in a department for a period as long as that manager managed that department?
Answer: "Di"
Category: (Projected, None) / (Duration, Interval, Computed) /

Query Q 2.1.7: Which managers served continuously longer than some other manager?
Answer: "Di"
Category: (Projected, None) / (Duration, Interval, Computed) /

6.2.2 Class O1.S2 (Duration, Interval, Other)

Query Q 2.2.1: Which employees had the same salary for a single period of at least three years?
Answer: "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) /

Query Q 2.2.2: Who worked for the same manager for at least five years continuously?
Answer: "Ed" and "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) /

Query Q 2.2.3: Which employees have stayed in the same department for the past 5 years?
Answer: "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) (Containment, Event, Explicit) /

Could also be classified as (Containment, Interval, Explicit.)

Query Q 2.2.4: For all departments which have not changed managers AND their budgets for the last 18 months, list their names, the managers and the budgets.
Answer: "(Toy, Di, 100K)"
Category: (Complete, None) / (Duration, Interval, Explicit) (Containment, Event, Explicit) /

Query Q 2.2.5: Who worked in the Toy department and earned at least 40K for the last two years?
Answer: "Edward" and "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) / (=, Constant) (<>, Constant)

Query Q 2.2.6: Who had at least three raises in a continuous five-year period?
Answer: "Ed" and "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) /

Query Q 2.2.7: Who had the most raises in a continuous five-year period?
Answer: "Ed" and "Di"
Category: (Projected, None) / (Duration, Interval, Explicit) /

6.2.3 Class O1.S3 (Duration, Element, Computed)

Query Q 2.3.1: Who worked in the Book department for as long as Di did?
Answer: "Ed" and "Edward"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Di never worked in the Book department, so this should return everyone that ever worked in the Book department.

Query Q 2.3.2: Who worked in the Toy department for as long as Di did?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 2.3.3: Who worked in a department longer than their current manager worked in that department?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Computed) (Containment, Event, Explicit) /

Query Q 2.3.4: Which managers managed which departments, longer than Di managed the Toy department?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 2.3.5: Who had the same salary for the longest total time?
Answer: "Di"
Category: (Projected, None) / (Duration, Element, Computed) /

Query Q 2.3.6: Who worked for a manager in a department for as long as that manager managed that department?
Answer: "Di"
Category: (Projected, None) / (Duration, Element, Computed) /

Query Q 2.3.7: Which departments had managers who served for the shortest total time?
Answer: "Toy"
Category: (Projected, None) / (Duration, Element, Computed) /

Query Q 2.3.8: List all employees in the Book department who received salaries of over 40K longer than Edward did.
Answer: "None"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (<>, Constant) (=, Constant)

Query Q 2.3.9: Who worked in the Toy department for at least as long as the total time that the Toy department was NOT managed by Ed?
Answer: "Di"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (<>, Constant)

Query Q 2.3.10: Find the names of employees that have been in a department named Toy for a shorter period than has DI.
Answer: "Ed" and "Edward."
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

The employee ED has been in a department named Toy for a period which is shorter than that of DI. The categorization is with respect to Figure 6. "(Projected, None)" indicates that only part of the attributes of the argument relation are present in the result and that there is no valid-time component. Next, "(Duration, Element, Computed)" indicates that a duration based predicates is used on element-valued arguments which are both derived from the valid-times of stored facts. Finally, "(=, Constant) (=, Constant)" indicates that there are two non-temporal selection predicates that test for equality of an attribute value with a constant (i.e., the person must be DI and the department must have name Toy).

Query Q 2.3.11: Find the current name and department name for the persons which made $40K for a longer period than DI did.
Answer: "(Edward, Book)."
Category: (Projected, None) / (Duration, Element, Computed) (Containment, Event, Explicit) / (=, Constant)

In this query, there are two valid-time selection based predicates. The one used for categorization compares the duration of time when a person makes $40K with the period of time that DI makes $40K. The other selects the name and department that overlap with the current time of qualifying persons.

6.2.4 Class O1.S4 (Duration, Element, Other)

Query Q 2.4.1: Who managed the Book department for at least two years?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Explicit) / (=, Constant)

Query Q 2.4.2: Which employees had the same salary for at least three years?
Answer: "Di"
Category: (Projected, None) / (Duration, Element, Explicit) /

Query Q 2.4.3: Who worked for the same manager for at least five years?
Answer: "Ed" and "Di"
Category: (Projected, None) / (Duration, Element, Explicit) /

Query Q 2.4.4: Who worked in a department for less than 6 months total?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Explicit) /

Query Q 2.4.5: Who had a position in the Toy department and a salary of 50K for at least two years?
Answer: "Di"
Category: (Projected, None) / (Duration, Element, Explicit) / (=, Constant) (=, Constant)

This query and the next illustrate the dependence on complete understanding in formulating a query. The syntax is almost the same, but the interpretation of "and" is different. In the above, "and" becomes temporal intersection while in the next query, it is merely logical conjunction.

Query Q 2.4.6: Who had a position in the Toy department and a position in the Book department for at least two years?
Answer: "None"
Category: (Projected, None) / (Duration, Element, Explicit) / (=, Constant) (=, Constant)

Query Q 2.4.7: Who worked for the same manager for total time of at least five years?
Answer: "Ed" and "Di"
Category: (Projected, None) / (Duration, Element, Explicit) /

6.2.5 Class O1.S5 (Other, Event, Computed)

Query Q 2.5.1: Find Ed's skills when he joined the Book department
Answer: "typing," "filing" and "driving."
Category: (Projected, None) / (Containment, Event, Computed) / (=, Constant)

The employee with name Ed joined the Book department on 4/1/87, when he was qualified for typing, filing and driving.

The category "(Containment, Event, Computed)" indicates that a containment based predicate is used to select skill data containing an event-valued argument (i.e. 4/1/87) which is derived from the valid-times of stored facts (begin of the valid-times Ed was at the Book department). Finally, "(=, Constant)" indicates that there are only non-temporal selection predicates that test for equality of an attribute value with a constant (i.e., the name of the person is Ed both in employee and in skills data, and the department of the person in employee data is Book).

Query Q 2.5.2: Find the name and the budget of Ed's departments when he joined them
Answer: "(Toy, $150K)" and "(Book, —)."
Category: (Projected, None) / (Containment, Event, Computed) / (=, Constant) (=, Foreign)

Ed joined the Toy department on 2/1/82, when the budget was $150K, and the Book department on 4/1/87. No information about the Book budget is available.

The valid-time selection is categorized as "(Containment, Event, Computed)" since it checks whether the valid-times of selected budget data contain an event-valued argument computed from employee data (the date Ed joined a department). A non-temporal selection predicate — in the category "(=, Constant)" — is used to select the department names and the periods in which Ed was in a department. A non-temporal predicate — in the category "(=, Foreign)" — is used to select the budget of that department, via the join condition `dept.Department=employee.Dept`*.*

Query Q 2.5.3: Find all the data and the skills of the employees of the Toy department when it opened
Answer: "(Di, $30K, Toy, F, 10/1/60, directing)."
Category: (Complete, None) / (Containment, Event, Computed) / (=, Constant) (=, Foreign)

The Toy department opened on 1/1/82. On that date, only Di worked in it. She was qualified for directing only.

The result is composed of all the non-temporal attributes retrieved from the relations employee and skills according to the output category "(Complete, None)". In both relations, data are selected if their valid-times contain the event "opening of the Toy department", according to two selection clauses in the category "(Containment, Event, Computed)". Additional non-temporal conditions are imposed to the selected data in either relations: "(=, Constant)" indicates the selection of the employees working in the Toy department; "(=, Foreign)" indicates the selection of the skills of such employees. Another predicate of the class "(=, Constant)" is used to determine the event "opening of the Toy department" as the beginning of the temporal element in which Toy data are stored in relation `dept`*.*

Query Q 2.5.4: Find the names of the employees who had been working in the Toy department before the budget was decreased
Answer: "Ed."
Category: (Projected, None) / (Ordering, Event, Computed) (Ordering, Interval, Computed) / (=, Constant) (<>, Constant)

The budget of the Toy department was decreased on 1/1/87. Ed is the only employee who had been working in it before that date.

The temporal selection predicate used for categorization is "(Ordering, Event, Computed)". It indicates that an ordering based predicate is used to select employee data with valid-times preceding an event-valued argument (i.e. 1/1/87) which is derived from stored facts. A predicate in the category "(=, Constant)" is used to select an employee working in the Toy department. The other temporal predicate — categorized as "(Ordering, Interval, Computed)" — selects two adjacent versions of department data, testing if their valid-time intervals meet. Furthermore, non-temporal predicates are used to determine such versions. "(=, Constant)" also indicates that the two versions must have the same department name: "Toy." "(<>, Constant)" indicates that a test must be effected on the decreasing value of the budget data between the two versions.

Query Q 2.5.5: Find Ed's skills when his salary increased from $30K to $40K
Answer: "typing," "filing" and "driving."
Category: (Projected, None) / (Containment, Event, Computed) (Ordering, Interval, Computed) / (=, Constant)

Ed had typing, filing and driving skills when his salary was increased from $30K to $40K on 2/1/86.

The query is categorized as "(Containment, Event, Computed)" since it selects skills data having valid times containing the event 2/1/86 which is computed from other data. Such an event is determined considering the two consecutive versions of data for the employee "Ed", and extracting the beginning of the second versions. The versions can be determined as follows: a first predicate in category "(=, Constant)" selects one Ed's version (with salary $30K); two predicates in the categories "(=, Constant)" and "(Ordering, Interval, Computed)" select the Ed's version (with salary $40K) that follows the first one. A third non-temporal predicate "(=, Constant)" selects Ed's data in relation `skills`*.*

6.2.6 Class O1.S6 (Other, Event, Other)

Query Q 2.6.1: Find the current Toy department data
Answer: "(Toy, Di, $100K)."

Category: (Complete, None) / (Containment, Event, Explicit) / (=, Constant)

The current manager and budget of the Toy department are Di and $100K, respectively.

The temporal selection predicate "(Containment, Event, Explicit)" selects the department data whose valid-time includes the explicit event "now." A non-temporal predicate in the category "(=, Constant)" selects Toy data.

Query Q 2.6.2: Find the skills for which Ed was qualified after 1/1/83
Answer: "filing" and "driving."
Category: (Projected, None) / (Ordering, Event, Explicit) / (=, Constant)

Ed was qualified for filing on 1/1/85 and for driving (for the second time in his career) on 6/1/84. The valid-time selection predicate "(Ordering, Event, Explicit)" compares the valid-time of Ed's skills with the explicitly supplied event 1/1/83. The non temporal predicate selects Ed's data in the skills relation.

Query Q 2.6.3: Find Di's salary on her 25th birthday
Answer: "$40K."
Category: (Projected, None) / (Containment, Event, User-defined) / (=, Constant)

The date of Di's 25th birthday is 10/1/85. At that time her salary was $40K.

The valid-time selection embedded in the query is categorized as "(Containment, Event, User-defined)," since it selects employee data whose valid-time contains an event computed from the user-defined time attribute `D-birth`*. Two non-temporal selections in the category "(=, Constant)" are used to select employees whose name is "Di" (once to select her* `D-birth` *and once to select her* `Salary`*).*

Query Q 2.6.4: Find the departments Ed worked in before and not after 1/1/88
Answer: "Toy."
Category: (Projected, None) / (Ordering, Event, Explicit) / (=, Single)(=, Constant)

Before 1/1/88 Ed worked only in the Toy department (from 2/1/82 to 1/31/87).

The query can be answered in two steps. At first, the names of all the departments Ed worked in are selected by means of a "(=, Const)" non-temporal predicate. In the second step, each name is used to find out the temporal element in which Ed worked in such a department by means of a "(=, Single)" non-temporal predicate. Finally, the department name is retrieved only if the temporal element precedes the event-valued constant 1/1/88 according to an "(Ordering, Event, Explicit)" temporal selection predicate.

Query Q 2.6.5: Find the name and date of birth of the women who were working in the Toy department on 1/1/83
Answer: "(Di, 10/1/60)."
Category: (Projected, None) / (Containment, Event, Explicit) / (=, Constant)

The only woman working in Toy department on 1/1/83 was Di. Di was born on 10/1/60.

The categorization of the query as "(Containment, Event, Explicit)" indicates that employee data are selected if their valid-times contain the event 1/1/83, explicitly supplied in the query. Furthermore, the non-temporal clause selects only female employees working in the Toy department.

Query Q 2.6.6: Who worked in their current department longer than their current manager?
Answer: "None"
Category: (Projected, None) / (Containment, Event, Explicit) /

6.2.7 Class O1.S7 (Other, Interval, Computed)

Query Q 2.7.1: Find the names of all employees that changed departments while DI was working in a department called Toy.
Answer: "Ed" and "Edward"
Category: (Projected, None) / (Containment, Interval, Computed) (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 2.7.2: Find the skills ED did not acquire while he was working in the BOOK department.
Answer: "Driving", "Directing", "Filing" and "Typing"
Category: (Projected, None) / (Containment, Interval, Computed) / (<>, Constant)

Query Q 2.7.3: Of the skills at some time possessed by ED, list those he did not acquire while he was working in the BOOK department.
Answer: "Driving", "Filing" and "Typing"
Category: (Projected, None) / (Containment, Interval, Explicit) (Containment, Interval, Explicit) / (<>, Constant)

Query Q 2.7.4: Find the manager and department of anyone who let a skill lapse for more than a month while their salary was less than $30K.

Answer: "(Di, Toy)."
Category: (Projected, None) / (Containment, Interval, Computer) (Duration, Interval, Explicit) / (=, Constant) (=, Constant)

Query Q 2.7.5: Find the name of anyone who reacquired a skill.
Answer: "Ed."
Category: (Projected, None) / (Ordering, Interval, Computed) (Ordering, Interval, Computed) / (<>, Constant)

Query Q 2.7.6: Find the name and sex of all employees that started work anywhere before ED acquired the skill driving for the 2nd time.
Answer: "(Ed, M) and (Di, F)."
Category: (Projected, None) / (Ordering, Interval, Computed) (Ordering, Event, Computed) / (Sequence, Constant)

Query Q 2.7.7: Who worked in a department for a given manager for at least the period when that manager managed that department?
Answer: "Di"
Category: (Projected, None) / (Containment, Interval, Computed) /

Query Q 2.7.8: What was the highest salary earned by Ed before changing his name to Edward?
Answer: "40K"
Category: (Projected, None) / (Ordering, Interval, Computed) / (=, Constant) (=, Constant)

6.2.8 Class O1.S8 (Other, Interval, Other)

Query Q 2.8.1: Find the name and skills of all people who worked for the BOOK or TOY department last year.
Answer: "(Edward, Typing), (Edward, Filing), (Edward, Driving), (Ed, Typing), (Ed, Filing), (Ed, Driving) and (Di, Directing)."
Category: (Complete, None) / (Containment, Interval, Explicit) / (=, Constant) (=, Constant)

Query Q 2.8.2: Find the name and current skills of all people who worked for the BOOK or TOY department last year.
Answer: "(Edward, Typing), (Edward, Filing) and (Di, Directing)."
Category: (Complete, None) / (Containment, Interval, Computed) (Containment, Interval, Explicit) / (=, Constant) (=, Constant)

Query Q 2.8.3: Find the name of all people who reported to DI before last year.
Answer: "Ed"
Category: (Complete, None) / (Ordering, Interval, Explicit) / (<, Constant)

Query Q 2.8.4: Find the manager of anyone who acquired a skill between 1983 and 1987 inclusive.
Answer: "Di."
Category: (Projected, None) / (Containment, Interval, Computed) / (=, Constant)

Query Q 2.8.5: Find the name of anyone who lost a skill in the last four years.
Answer: "()."
Category: (Projected, None) / (Containment, Interval, Explicit) / (=, Constant)

Query Q 2.8.6: Find the name and department of anyone who changed their name or salary between July 1987 and June 1988 inclusive.
Answer: "(Ed, Book)."
Category: (Projected, None) / (Containment, Interval, Computed) / (=, Constant)

Query Q 2.8.7: Which employees stayed at their first salary for less than one year?
Answer: "Ed"
Category: (Projected, None) / (Containment, Interval, Explicit) /

Query Q 2.8.8: List the names, managers and budgets of all departments with budgets of less than 200K during any period between Jan 1, 1985 and Dec 31, 1989?
Answer: "(Toy, Di, 100K)"
Category: (Complete, None) / (Containment, Interval, Explicit) / (<>, Constant)

Query Q 2.8.9: Who worked in the Toy department and earned at least 40K in the last two years?
Answer: "Edward" and "Di"
Category: (Projected, None) / (Containment, Element, Explicit) / (=, Constant) (<>, Constant)

6.2.9 Class O1.S9 (Other, Element, Computed)

Query Q 2.9.1: Find the names of departments that had a budget greater than $90K during the period when Di managed them.
Answer: "Toy".
Category: (Projected, None) / (Containment, Element, Computed) / (=, Constant) (=, Constant) (=, Foreign)

The employee with name Di has managed the Toy department since 1/1/1982 and during that period the *budget was always greater than $90K. The category "(Containment, Element, Computed)" indicates that a containment based predicates is used on element-valued arguments which are both derived from the valid-times of stored facts. The non-temporal selection predicates test for equality between the employee name and "Di", the equality between the same attribute and the foreign key of the Dept relation and the comparison between the department budget and the amount of $120K.*

Query Q 2.9.2: Find Ed's salary during the periods he worked in the same department as Di's.
Answer: $20K, $30K, $40K.
Category: (Projected, None) / (Containment, Element, Computed) / (=, Constant) (=, Single) (=, Constant)

The second non-temporal selection predicate belonging to the class "(=, Single)" test the equality between the department of Ed and Di.

Query Q 2.9.3: Find the names of the departments which Ed worked in earning $40K.
Answer: "Toy","Book".
Category: (Projected, None) / (Containment, Element, Computed) / (=, Constant) (=, Constant)

The predicates belonging to the class "(=, Constant)" indicate the selection of the employee Ed and test the equality between his salary and the amount of $40K.

Query Q 2.9.4: Find Ed's name after he left the Toy department.
Answer: "Edward".
Category: (Projected, None) / (Ordering, Element, Computed) / (=, Constant) (=, Constant)

An ordering based predicate is used to select the name of employeee Ed with valid-time following the period he worked for the Toy department.

Query Q 2.9.5: Find Ed's skills when he worked in the Toy department.
Answer: "driving","filling","typing".
Category: (Projected, None) / (Containment, Element, Computed) / (=, Constant) (=, Foreign) (=, Constant)

The non-temporal predicate of the class "(=, Constant)" indicates the selection of the employee with name Ed; "(=, Foreign)" indicates the selection of the skills of such employee and "(=, Constant)" indicates the selection of the Toy department.

Query Q 2.9.6: Who ever worked in a department longer than their manager while in that department?
Answer: "None"
Category: (Projected, None) / (Containment, Element, Computed) /

Query Q 2.9.7: What new skills did Ed hold after he had changed his name to Edward?
Answer: "None"
Category: (Projected, None) / (Ordering, Element, Computed) / (=, Constant) (=, Constant)

Query Q 2.9.8: What were Toy's departmental budgets when it was managed by Di?
Answer: "150K", "200K" and "100K"
Category: (Projected, None) / (Containment, Element, Computed) / (=, Constant) (=, Constant)

6.2.10 Class O1.S10 (Other, Element, Other)

Query Q 2.10.1: Which managers managed which departments between Jan 1, 1982 and Dec 31, 1989?
Answer: "(Di, Toy)"
Category: (Projected, None) / (Containment, Element, Explicit) /

6.3 Valid-time Output

This section involves queries which return only valid-time values—no explicit-attribute values are present in the result.

6.3.1 Class O2.S1 (Duration, Interval, Computed)

Query Q 3.1.1: Find the times when persons with a shorter employment in the Toy department than DI were employed in the Book department.
Answer: "{ [4/1/87 - `now`]}"
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant)

Query Q 3.1.2: Find the employment periods of persons that made 40K for a longer time than when DI made 40K.
Answer: "{ [2/1/82 - 1/31/87], [4/1/87 - `now`]}"
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant)

Query Q 3.1.3: Find the starting times in the Book department of persons which possessed the Filing skill for a longer time than DI.
Answer: "{ 4/1/87}"

Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant)

Query Q 3.1.4: Return the times when persons employed a shorter time than DI acquired a skill.
Answer: "{ 1/1/82, 4/1/82, 6/1/84, 1/1/85, }"
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant)

Query Q 3.1.5: Find the employment periods of persons employed shorter time than DI.
Answer: "{ [2/1/82 - 1/31/87], [4/1/87 - `now`]}"
Category: (None, Element, Derived) / (Duration, Element, Computed) /(=, Constant)

Query Q 3.1.6: When did someone get a raise more quickly than Di's first raise?
Answer: "06/01/82" and "09/01/86"
Category: (None, Event, Derived) / (Duration, Interval, Computed) (Ordering, Interval, Computed) / (= ,Constant)

Query Q 3.1.7: When was the longest period when no one was hired or left employment?
Answer: "02/01/82–01/01/90"
Category: (None, Interval, Derived) / (Duration, Interval, Computed) /

Query Q 3.1.8: When was the longest period when no one received a raise?
Answer: "06/02/82–07/31/84"
Category: (None, Interval, Derived) / (Duration, Interval, Computed) /

Query Q 3.1.9: When was the longest period when a department was without a manager?
Answer: "04/01/87–01/01/90"
Category: (None, Interval, Derived) / (Duration, Interval, Computed) /

6.3.2 Class O2.S2 (Duration, Element, Other)

Query Q 3.2.1: Find employment periods in the Toy department for persons that have worked there for at least 8 years.
Answer: "{ [1/1/82 - `now`]}"
Category: (None, Element, Derived) / (Duration, Element, Explicit) /(=, Constant)

Query Q 3.2.2: Find the starting times of managers which managed a department for at least 5 years.
Answer: "{ 1/1/82}"
Category: (None, Element, Derived) / (Duration, Element, Explicit) /(=, Constant)

Query Q 3.2.3: Find the rehiring dates of employees with a gap in employment that exceeds 1 month.
Answer: "{ 4/1/87}"
Category: (None, Element, Derived) / (Duration, Element, Explicit) /(=, Constant)

Query Q 3.2.4: Find the times when persons possessed skills that they lost and regained more than 1 year later.
Answer: "{ [1/1/82 - 5/1/82], [6/1/84 - 5/31/88]}"
Category: (None, Element, Derived) / (Duration, Element, Explicit) /(=, Constant)

Query Q 3.2.5: Find budget periods that exceed 2 years.
Answer: "{ [1/1/87 - `now`]}"
Category: (None, Element, Derived) / (Duration, Element, Explicit) /(=, Constant)

Query Q 3.2.6: When did no one's salary change for at least six months?
Answer: "06/01/82–07/31/84", "08/01/84–01/31/85", "02/01/85–08/31/86"
Category: (None, Interval, Derived) / (Duration, Interval, Explicit) (temporal aggregate) /

6.3.3 Class O2.S3 (Duration, Element, Computed)

Class remark : Most sensible queries are of similar to "when did X have X.A longer than Y had Y.B?"

Query Q 3.3.1: When did somebody have the same salary for the longest total time?
Answer: "09/01/86–01/01/90"
Category: (None, Element, Derived) / (Duration, Element, Computed) (temporal aggregate) /

Query Q 3.3.2: When did anybody work for a manager in a department for as long as that manager managed that department?
Answer: "01/01/82–01/01/90"
Category: (None, Element, Derived) / (Duration, Element, Computed) /

Query Q 3.3.3: When did one person hold a skill for a longer period than another but earn a lower salary?
Answer: "No time."
Category: (None, Element, Derived) / (Duration, Element, Computed) /

Query Q 3.3.4: When did someone manage the Toy department for longer than Di did?
Answer: "No time."

Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 3.3.5: When did anyone have a skill longer than Ed had driving?
Answer: "No time."
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 3.3.6: When did a department have at least two employees longer than the Toy department did?
Answer: "No time."
Category: (None, Element, Derived) / (Duration, Element, Computed) (Containment, Element, Computed) / (=, Constant)

Query Q 3.3.7: Who worked in one department for at least two years continuously and what was the period of employment in that department?
Answer: "(Ed, 02/01/82–01/31/87), (Edward, 01/01/88–01/01/90), (Di, 01/01/82–01/01/90)"
Category: (Projected, Interval, Derived) / (Duration, Interval, Explicit) /

6.3.4 Class O2.S4 (Duration, Element, Other)

6.3.5 Class O2.S5 (Other, Event, Computed)

6.3.6 Class O2.S6 (Other, Event, Other)

Query Q 3.6.1: When did anybody have at least the skills that Di currently has?
Answer: "No time."
Category: (None, Element, Derived) / (Containment, Event, Explicit) / (=, Constant)

6.3.7 Class O2.S7 (Other, Interval, Computed)

Query Q 3.7.1: Find when the Toy budget decreased
Answer: "1/1/87."
Category: (None, Not Empty) / (Ordering, Interval, Computed) / (=, Constant) (<>, Single)

The budget of the Toy department decreased on 1/1/87.

The required date is extracted when two consecutive periods are found (i.e. selected via the predicate "(Ordering, Interval, Computed)") in the first of which the budget of the Toy department was higher than in the second one (as tested by "(<>, Single)"). The category "(=, Constant)" indicates that Toy data are selected in relation `dept` *via a non-temporal predicate testing the equality of the key with a constant. The required date is extracted as the beginning of the valid-time of the second period.*

Query Q 3.7.2: Find when the name of an employee was presumably changed
Answer: "1/1/88."
Category: (None, Not Empty) / (Ordering, Interval, Computed) / (=, Single) (<>, Single)

Only ED's name changed from Ed to Edward on 1/1/88.

We can guess that employees with different names but with the same gender and date of birth are the same person (also the salary and the department before and after the name change can be tested for equality, but their values could have been changed together with the name). The retrieved event is the validity beginning of the second of two consecutive employee versions. Such versions are detected by means of the following conditions: their valid-time intervals meet (cf. "(Ordering, Interval, Computed)"), their time-invariant attributes but the name are equal (cf. "(=, Single)" used twice), their names are different (cf. "(<>, Single)").

Query Q 3.7.3: Find when the salary of a manager increased
Answer: "8/1/84" and "9/1/86."
Category: (None, Not Empty) / (Ordering, Interval, Computed) (Containment, Element, Computed) / (=, Foreign) (=, Const) (<>, Single)

The only manager in our data is Di. Di's salary was increased on 8/1/84 and on 9/1/86.

First, all the manager names (together with the valid-times they were manager) are retrieved from the relation `dept` *by means of a simple projection. Then, the data of each manager are selected in the relation* `emp` *by means of a non-temporal predicate in the class "(=, Foreign)" and a valid-time predicate in the class "(Containment, Element, Computed)". In order to find the desired events, two consecutive versions of the data of a manager with increasing salary must be detected. The desired dates are extracted as the begin of the valid-time interval of the second version. A version of the manager data is selected via the "(=, Foreign)" predicate as a candidate to be the first desired version. The second version is selected in relation* `emp` *if its valid-time interval is met by the valid-time interval of the first version (temporal selection in* *category "(Ordering, Interval, Computed)"), has the same* `Name` *and has a greater* `Salary` *than the first version (non-temporal selection with categories "(=, Constant)" and "(<>, Single)"). Notice that, if the name equality for the second version is tested with the names retrieved from* `dept` *rather than with the name in the first selected version, two predicates of the class "(=, Foreign)" are used and "(=, Constant)" is no longer present in the categorization.*

Query Q 3.7.4: Find the periods in which Di earned $40K while she was manager of the Toy department
Answer: "8/1/84 – 8/31/86."
Category: (None, Not Empty) / (Containment, Interval, Computed) / (=, Constant)

DI has been manager of the Toy department since 1982. In this period, she earned $40K from 8/1/84 to 8/31/86.

The query selects the valid-time intervals — of the employee data with name "Di" and salary $40K, according to "(=, Constant)" non-temporal selection — completely contained in another interval computed from other data, according to a "(Containment, Interval, Computed)" predicate. Such an interval is the valid time she was manager of the Toy department and is selected from the relation `dept` *via a non-temporal predicate in the category "(=, Const)" concerning the* `Manager` *attribute value.*

Query Q 3.7.5: Find the acquisition dates of the skills Ed had during the year he joined the Toy department
Answer: "1/1/82" and "4/1/82."
Category: (None, Not Empty) / (Containment, Interval, Computed) / (=, Constant)

Ed joined the Toy department on 2/1/82. During 1982, he acquired a driving skill on the 1st of January, and a typing skill on the 1st of April.

The year 1982 is computed from the beginning of the temporal element Ed worked in the Toy department, which is selected by means of non-temporal predicates in the category "(=, Constant)." Hence, the Ed's skills (selected by another "(=, Constant)" predicate) whose valid-times overlap the computed year are retrieved according to a "(Containment, Interval, Computed)" temporal selection predicate.

6.3.8 Class O2.S8 (Other, Interval, Other)

Query Q 3.8.1: Find the beginning of the period which includes the year 1989 and in which Edward had a constant salary.
Answer: "1/1/88."
Category: (None, Not Empty) / (Containment, Interval, Explicit) / (=, Constant)

Edward has got a constant salary since 1/1/88. As this period includes 1989, 1/1/88 is the answer.

Assuming that valid-time intervals of employee data with a constant salary can be extracted from the relation `employee` *(it can be noticed that, if the relation is stored by means of temporally homeogeneous tuples, the employee histories must be* **coalesced** *after projection on* `salary` *in order to obtain maximal intervals in which the salary has not been changed), those concerning Edward can be selected via a "(=, Constant)" predicate. The query retrieves the beginning of the computed period which contain the explicit interval 1/1/89 – 12/31/89, according to a "(Containment, Interval, Explicit)" temporal predicate.*

Query Q 3.8.2: Find the dates Ed acquired a skill before or after years 1984–1985.
Answer: "1/1/82" and "4/1/82."
Category: (None, Not Empty) / (Ordering, Interval, Explicit) / (=, Constant)

Before or after years 1984–1985, Ed acquired a skill on 1/1/82 (driving) and on 4/1/82 (typing).

The beginning of the valid-times of Ed's skills are selected if they precede or follow the explicit interval "1/1/84 – 12/31/85," according to the category "(Ordering, Interval, Explicit)." A non-temporal predicate in the category "(=, Constant)" is used to select Ed's data in relation `skills`*.*

Query Q 3.8.3: Find ED's unemployment periods when he was not 30 years old
Answer: "1/31/87 – 1/4/87."
Category: (None, Not Empty) / (Ordering, Interval, User-defined) (Containment, Interval, User-defined) / (=, Constant)

ED has been 30 years old from 7/1/85 to 6/30/86. Before 7/1/85 or after 6/30/86, he was unemployed from 1/31/87 to 1/4/87.

The query is categorized as "(Ordering, Interval, User-defined)" because valid-times of data are selected if they precede or follow an interval which is computed from user-defined times stored in other data, that is 7/1/85 – 6/30/86. The other temporal predicate, in the category "(Containment, Interval, User-defined)," is used because valid-times qualify for the query also when they overlap the interval 7/1/85 – 6/30/86 without being completely contained in it. In this case, only the portion of the qualifying intervals outside

7/1/85 – 6/30/86 must be retrieved. Non-temporal predicates in the category "(=, Constant)" are used to restrict `employee` *to ED's data (*`Name="Ed"` *or* `Name="Edward"`*), and to find out the periods in which he was unemployed (e.g. testing for null values of his department and salary; for the problem of coalescing intervals, see the discussion of query Q 3.8.1). The restriction to ED's data also allows the determination of his date of birth which is used for the temporal selection.*

Query Q 3.8.4: Find the period in which Di worked in the department in which she has been working during the whole 1987
Answer: "1/1/82 – now."
Category: (None, Not Empty) / (Containment, Interval, Explicit) / (=, Constant)

Di worked in Toy department during 1987. She has worked in that department from 1/1/82.

The category "(Containment, Interval, Explicit)" indicates that a containment-based predicate is used to select employee data whose valid-times contain the user-supplied interval "1/1/87 – 12/31/87." As in Query Q 3.8.1, we assume that periods in which an employee worked in the same department can be determined. Di's data are selected via a "(=, Constant)" non-temporal predicate. If the containment test succeds, the entire element-valued valid-time argument of the predicate is retrieved.

Query Q 3.8.5: Find all the dates from 1/1/83 to 12/31/85 at which the Toy department budget changed
Answer: "8/1/84."
Category: (None, Not Empty) / (Containment, Interval, Explicit) (Ordering, Interval, Computed) / (=, Single) (<>, Single)

From 1/1/83 to 12/31/85, the Toy budget changed only on 8/1/84.

In this query, there are two valid-time selection predicates. The one used for categorization is "(Containment, Interval, Explicit)" and selects the dates of interest by means of their containment in an interval explicitly supplied in the query. The other one, "(Ordering, Interval, Computed)," is used to determine two consecutive versions of department data, testing if their valid-time intervals meet. The non-temporal predicates are also used for their determination: the two versions must have the same department name — according to "(=, Single)" — and different salary values — according to "(<>, Single)".

6.3.9 Class O2.S9 (Other, Element, Computed)

Query Q 3.9.1: When did anybody have at least the skills that Di had at the same time?
Answer: "None"
Category: (None, Element, Derived) / (Containment, Element, Computed) / (=, Constant)

Query Q 3.9.2: When was the budget for Toy department more than 100K?
Answer: "01/01/82–12/31/86"
Category: (None, Element, Derived) / (Containment, Element, Computed) / (=, Constant) (<>, Constant)

Query Q 3.9.3: When was the last period when Edward had driving, filing and typing skills simulataneously?
Answer: "01/01/88–05/31/88"
Category: (None, Event, Derived) / (Containment, Event, Computed) (temporal aggregate) / (=, Constant) (=, Constant)

Query Q 3.9.4: When did anybody have at least the skills that Di had?
Answer: "No time."
Category: (None, Element, Derived) / (Containment, Element, Computed) / (=, Constant)

Query Q 3.9.5: When did Di earn less than Ed?
Answer: "No time."
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 3.9.6: When did Ed work in Toy department while the department was managed by Di?
Answer: "02/01/82–12/31/87"
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant) / (=, Constant) (=, Constant)

Query Q 3.9.7: When did Edward have driving skills?
Answer: "01/01/82–05/01/82" and "06/01/84–05/31/88"
Category: (None, Element, Derived) / (Containment, Element, Computed) / (=, Constant) (=, Constant)

6.3.10 Class O2.S10 (Other, Element, Other)

6.4 Explicit-attribute and Valid-time Output

The output from queries in this section contains explicit attribute values with associated valid times.

6.4.1 Class O3.S1 (Duration, Interval, Computed)

Query Q 4.1.1: Who, and when, were continuously employed in the Toy department shorter than Di was continuously employed in the Toy department?
Answer: "Ed from 01-Feb-1982 to 31-Jan-1987."
Category: (Projected, Interval, Derived) / (Duration, Interval, Computed) / (=, Constant) (=, Constant) (=, Constant)

Query Q 4.1.2: Who, and when, were continuously employed in the Toy department shorter than Di was continuously employed in the Toy department, and what are their gender and date of birth?
Answer: "Ed from 01-Feb-1982 to 31-Jan-1987."
Category: (Complete, Interval, Derived) / (Duration, Interval, Computed) / (=, Constant) (=, Constant) (=, Constant)

Query Q 4.1.3: Who were continuously employed in the Toy department shorter than Di was continuously employed in the Toy department, and when did this employment start?
Answer: "Ed from 01-Feb-1982 to 31-Jan-1987."
Category: (Projected, Event, Derived) / (Duration, Interval, Computed) / (=, Constant) (=, Constant) (=, Constant)

Query Q 4.1.4: Who were continuously employed in the Toy department shorter than Di was continuously employed in the Toy department, throughout 1984?
Answer: "Ed from 01-Jan-1984 to 31-Jan-1984."
Category: (Projected, Interval, Imposed) / (Duration, Interval, Computed) / (=, Constant) (=, Constant) (=, Constant)

Query Q 4.1.5: Who were continuously employed in the Toy department shorter than Di was continuously employed in the Toy department, by the start of 1984?
Answer: "Ed on 01-Jan-1984."
Category: (Projected, Event, Imposed) / (Duration, Interval, Computed) / (=, Constant) (=, Constant) (=, Constant)

6.4.2 Class O3.S2 (Duration, Interval, Other)

Query Q 4.2.1: When was the Toy department's budget constant and greater than $175K for more than one year?
Answer: "$200K from 01-Jan-1984 to 31-Dec-1986."
Category: (Projected, Interval, Derived) / (Duration, Interval, Supplied) / (≠, Constant) (=, Constant)

Query Q 4.2.2: When was the Toy department's budget constant and greater than $175K for more than one year, and who was the manager for that time?
Answer: "$200K from 01-Jan-1984 to 31-Dec-1986, managed by Di."
Category: (Complete, Interval, Derived) / (Duration, Interval, Supplied) / (≠, Constant) (=, Constant)

Query Q 4.2.3: When did the Toy department's budget exceed $175K and then continue unchanged for a year?
Answer: "$200K at 01-Jan-1984."
Category: (Projected, Event, Derived) / (Duration, Interval, Supplied) / (≠, Constant) (=, Constant)

Query Q 4.2.4: Who managed departments (and when) whose budgets exceeded $175K and then held constant for at year?
Answer: "Di, from 01-Aug-1984 to 31-Dec-1986."
Category: (Projected, Interval, Derived) / (Duration, Interval, Supplied) / (=, Constant) (=, Foreign)

Query Q 4.2.5: What departments were in continuous operation (and when) longer than the duration between Ed's and Di's birthdates?
Answer: "Toy, from 01-Jan-1982 to present."
Category: (Projected, Interval, Derived) / (Duration, Interval, User-defined) / (=, Constant) (=, Constant)

Values computed from attribute values of type user-defined time don't quite fit the taxonomy. Also, the books department is not included in the answer to the last query, because it started operation on 01-Apr-87, which is less than five years before now (01-Jan-1990).

6.4.3 Class O3.S3 (Duration, Element, Computed)

Query Q 4.3.1: Who, when and for which department did anybody work for as long as the length of time that department's budget was below 200K?

Answer: "(Di, Toy, 01/01/82–01/01/90)"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (<>, Constant)

Query Q 4.3.2: Who and when did anybody work in a department longer than their manager while in that department?
Answer: "None"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) /

Query Q 4.3.3: Who and when did anybody work in a department longer than their current manager worked in that department?
Answer: "None"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (Containment, Event, Explicit) /

Query Q 4.3.4: For all employees who managed any departments at least as long as Di managed the Toy department, list their names, their gender, their departments and their salary histories during that time.
Answer: "(Di, F, Toy, ((30K, 01/01/82–07/31/84), (40K, 08/01/84–08/31/86), (50K, 09/01/86–01/01/90)))"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Query Q 4.3.5: List the names of departments, the managers and the times when those departments had managers who served for the shortest total time?
Answer: "Toy, Di, 01/01/82–01/01/90"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) /

Query Q 4.3.6: For all departments which had budgets of at least 200K for a longer total time than budgets of less than 200K, list their names, their budgets and the times when the budgets were NOT below 200K.
Answer: "None"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (<>, Constant) (<>, Constant)

Query Q 4.3.7: What skills did Ed hold for as long as the total time that he did NOT have driving skills, and when did he have these skills?
Answer: "Typing, 04/01/82–12/31/87"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (<>, Constant)

6.4.4 Class O3.S4 (Duration, Element, Other)

Query Q 4.4.1: Find who was a driver, and the associated times, for a longer period than Ed, from 1/1/1982 to 12/31/1984.
Answer: "(Di, 1/1/1982 – 12/31/1984)"
Category: (Projected, Not Empty) / (Duration, Element, Explicit) / (<>, Constant)

Query Q 4.4.2: Find who had been making the same salary for the longest time in on Ed's birtday, and the length of that time.
Answer: "(Di, 8/1/1984 – 7/1/1985)"
Category: (Projected, Not Empty) / (Duration, Element, User-defined time) / (=, Constant)

Query Q 4.4.3: Find who was the oldest typist on 12/31/1985, and the time during which they had they been a typist.
Answer: "(Ed, 4/1/1982 – 12/31/1985)"
Category: (Complete, Not Empty) / (Duration, Element, Explicit) / (=, Constant), (=, Foreign)

Query Q 4.4.4: Find the names of employees in the toy department who had worked less than Di in that department as of 1/1/1985, and the amount of the difference in time.
Answer: "(Ed, 1 month less)"
Category: (Projected, Not Empty) / (Duration, Element, Explicit) / (=, Constant), (<>, Single)

Query Q 4.4.5: Find the working employees who worked the least during 1987, and the times during which they worked (as based on the *Employee relation*).
Answer: "(Ed, 1/1/1987 – 1/31/1987 U 4/1/1987 – 12/31/1987)"
Category: (Projected, Not Empty) / (Duration, Element, Explicit) / (Nothing)

Query Q 4.4.6: Who had at least two years of seniority in some department and what were the periods of such seniority?
Answer: "(Di, 01/01/82–01/01/90)"
Category: (Projected, Element, Derived) / (Duration, Element, Explicit) /

6.4.5 Class O3.S5 (Other, Event, Computed)

Query Q 4.5.1: List the names and ages of all employees at the time they received their first salary increment.
Answer: "(Ed, 26 years & 11 months), (Di, 23 years & 10 months)"
Category: (Projected, Interval, Derived) / (Containment, Event, Computed) /

Query Q 4.5.2: List the names and salary histories of all female employees as of their 25th birthday.
Answer: "(Di, (30K, 01/01/82–07/31/84), (40K, 08/01/84–10/01/85))"
Category: (Projected, Element, Derived) / (Containment, Event, Computed) (Duration, Interval, explicit) / (=, Constant)

Query Q 4.5.3: When and who ever changed their names?
Answer: "(Ed, 01/01/88)"
Category: (Projected, Event, Derived) / (Containment, Event, Computed) /

Query Q 4.5.4: How old was Ed and what skills did he have at the time he changed his name to Edward?
Answer: "(32 years & 6 months, {Driving, Filing, Typing})"
Category: (Projected, Interval, Derived) / (Containment, Event, Computed) / (=, Constant) (=, Constant)

Query Q 4.5.5: When did Ed acquire driving skills and what other skills did he have at the time?
Answer: "(01/01/82, None), (06/01/84, Typing)"
Category: (Projected, Event, Derived) / (Containment, Event, Computed) / (=, Constant) (=, Constant)

Query Q 4.5.6: Who and when was the first female manager of the Toy department?
Answer: "(Di, 01/01/82)"
Category: (Projected, Event, Derived) / (Containment, Event, Computed) (temporal aggregate) / (=, Constant) (=, Constant)

6.4.6 Class O3.S6 (Other, Event, Other)

Query Q 4.6.1: Find the name and salary histories of employees whose date-of-birth was after 1/1/56.
Answer: "DI's" entire Name and Salary history
Category: (Projected, Element,Derived) / (Ordering, Event, User-defined) /

Query Q 4.6.2: Find the name and salary histories of employees who were called "Ed" after 1/1/88.
Answer: "Empty Answer"
Category: (Projected, Element, Derived) / (Ordering, Event, User-Defined) /

Query Q 4.6.3: Find the name and salary histories since their latest pay raise of employees whose latest pay raise was in 1985.
Answer: "ED's Name and Salary history from 2/1/85 onwards"
Category: (Projected, Element, Derived) / (Containment, Event, Explicit) /

Query Q 4.6.4: Find the name and salary histories of employees whose latest pay raise occurred after the date-of-birth of every other employee.
Answer: "Both DI's and ED's enitre Name and Salary history
Category: (Projected, Element, Derived) / (Ordering, Event, User-Defined) /

Query Q 4.6.5: Find the name and salary histories of employees whose latest pay raise occurred on the date-of-birth of some other employee.
Answer: "Empty Answer"
Category: (Projected, Element, Derived) / (Containment, Event, User-Defined) /

Query Q 4.6.6: Who and when had at least the skills that Di currently has?
Answer: "None"
Category: (Projected, Element, Derived) / (Containment, Event, Explicit) / (=, Constant)

6.4.7 Class O3.S7 (Other, Interval, Computed)

Query Q 4.7.1: Find the salary paid and the time it was paid, of any toy department employee before Ed worked there.
Answer: "(Di, $40K, 1/1/1982 – 7/31/1984)"
Category: (Projected, Not Empty) / (Ordering, Interval, Computed) / (<>, Constant)

Query Q 4.7.2: Find the greatest salary paid to Ed under $50K and the times during which it was paid.
Answer: "($40K, 2/1/1985 – 1/31/1987)," "($40k, 4/1/1987 – present)"
Category: (Projected, Not Empty) / (Containment, Interval, Computed) / (=, Constant)

Query Q 4.7.3: Find Ed's salary history.
Answer: "(Ed, $20K, 2/1/1982 – 5/31/1982)," "(Ed, $30K, 6/1/1982 – 1/31/1985)," "(Ed, $40K, 2/1/1985 – 1/31/1987)," "(Ed, $40k, 4/1/1987 – present)"
Category: (Projected, Not Empty) / (Containment, Interval, Computed) / (=, Constant)

Query Q 4.7.4: Find the name, salary and the time during which this occured, for drivers who made less than $40K.

Answer: "(Ed, \$20K, 1/1/1982 – 5/1/1982)," "(Ed, \$30K, 6/1/1984 – 1/31/1985)," "(Di, \$30K, 1/1/1982 – 7/31/1984)"
Category: (Projected, Not Empty) / (Containment, Interval, Computed) / (<>, Constant)

Query Q 4.7.5: Find the budget of the toy department and the time at which Ed worked there.
Answer: "(2/1/1982 – 1/31/1987, \$150K)"
Category: (Projected, Not Empty) / (Containment, Interval, Computed) / (=, Constant), (=, Foreign

6.4.8 Class O3.S8 (Other, Interval, Other)

Query Q 4.8.1: Find the name, active skills and period of all people who worked for the BOOK or TOY department in the last two years.
Answer: "(Edward, Typing, 1-Apr-1982 to date), (Edward, Filing, 1-Jan-1985 to date), (Edward, Driving, 1-Jan-1982 to 1-May-1982), (Edward, Driving, 1-Jun-1984 to 31-May-1988) and (Di, Directing, 1-Jan-1982 to date)."
Category: (Complete, Element) / (Containment, Interval, Explicit) / (=, Constant) (=, Constant)

Query Q 4.8.2: Find the name, active skills and relevant period of all people who worked for the BOOK or TOY department in the last two years.
Answer: "(Edward, Typing, 1-Apr-1982 to date), (Edward, Filing, 1-Jan-1985 to date), (Edward, Driving, 1-Jun-1984 to 31-May-1988) and (Di, Directing, 1-Jan-1982 to date)."
Category: (Complete, Element) / (Containment, Interval, Computed) (Duration, Interval, Explicit) / (=, Constant) (=, Constant)

Query Q 4.8.3: Find the name and period of all people who reported to DI before last year.
Answer: "(Ed, 1-Feb-1982 to 31-Jan-1987)"
Category: (Complete, None) / (Containment, Interval, Computed) / (<, Constant)

Query Q 4.8.4: Find the manager and date of anyone who acquired a skill between 1983 and 1987 inclusive.
Answer: "(Di, 1-Jun-1984) and (Di, 1-Jan-1985)."
Category: (Projected, None) / (Containment, Interval, Computed) / (=, Constant)

Query Q 4.8.5: Find the name and dates of anyone who had two raises in the period March 1982 to March 1985 inclusive.
Answer: "(Ed, 1-Jun-1982, 1-Feb-1985)."
Category: (Projected, None) / (Containment, Interval, Computed) / (=, Constant)

6.4.9 Class O3.S9 (Other, Element, Computed)

Query Q 4.9.1: Find Ed's salary history when he had a driving skill
Answer: "(\$20K, 2/1/82 – 5/1/82)," "(\$30K, 6/1/84 – 1/31/85)" and "(\$40K, 2/1/85 – 1/31/87 ∪ 4/1/87 – 12/31/87)"
Category: (Projected, Not Empty) / (Containment, Element, Computed) / (=, Constant)

Ed has been qualified for driving from 2/1/82 to 5/1/82 and from 6/1/84 to 12/31/87 (after that, Edward nor Ed had). In those periods, he earned \$20K from 2/1/82 to 5/1/82, \$30K from 6/1/84 to 1/31/85 and \$40K from 2/1/85 to 1/31/87 and from 4/1/87 to 12/31/87.

A non-temporal "(=, Constant)" is used to select Ed's data in relation `emp`. *Ed's salary data are retrieved if their valid-times are contained in an element-valued period computed from stored facts, according to "(Containment, Element, Computed)". Such period is computed by means of other "(=, Constant)" predicates which select data with* `Name="Ed"` *and* `Skill="driving"` *in relation* `skills`.

Query Q 4.9.2: Find the salary history, during the periods in which ED had a driving skill, of the employees who earned less than \$50K in 1989
Answer: "(—, 1/1/82 – 1/31/82)," "(\$20K, 2/1/82 – 5/1/82)," "(\$30K, 6/1/84 – 1/31/85)," "(\$40K, 2/1/85 – 1/31/87)," "(—, 1/31/87 – 3/30/87)" and "(\$40K, 4/1/87 – 5/31/88)."
Category: (Projected, Not Empty) / (Containment, Element, Computed) (Containment, Interval, Explicit) / (=, Constant) (<>, Constant)

ED was qualified for driving from 1/1/82 to 5/1/82 and from 6/1/84 to 5/31/88. The only employee who earned less than \$50K in 1989 is ED again. While he had the driving skill, his salary was \$20K from 2/1/82 to 5/1/82, \$30K from 6/1/84 to 1/31/85, \$40K from 2/1/85 to 1/31/87 and from 4/1/87 to 5/31/88 (unknown from 1/1/82 to 1/31/82 and from 1/31/87 to 3/30/87).

Two valid-time selection predicates are present in the query. One — "(Containment, Interval, Explicit)" — is used to select the objects of interest, whereas the other — "(Containment, Element, Computed)" — is used to select a portion of their data history. Objects qualify if the salary is less that \$50K (cf. "(<>, Constant)") in their employee data selected by the temporal predicate "(Containment, Element, Computed)," extracting the versions with valid-times overlapped by *the user-supplied interval "1/1/89 – 12/31/89". The other non-temporal predicate is used to evaluate the element-valued period in which ED had the driving skill (cf. "(=, Constant)" testing for* `skills.Name` *equal to "Ed" or "Edward"). Such period is used as argument by the temporal predicate used for categorization.*

Query Q 4.9.3: Find the name and salary history of the male employees in the periods they were directed by a woman

Answer: "($20K, 2/1/82 – 5/31/82)," and "($30K, 6/1/82 – 1/31/85)."

Category: (Projected, Not Empty) / (Containment, Element, Computed) / (=, Constant) (=, Foreign)

The only employee qualifying for the query is Ed, since he had been directed by Di when he worked in the Toy department from 2/1/82 to 1/31/87. His salary history in that period is: $20K from 2/1/82 to 5/31/82, $30K from 6/1/82 to 1/31/85 and $40K from 2/1/85 to 1/31/87.

The objects of interest for this query are the male employee which can be identified by a "(=, Constant)" predicate in the relation `emp`*. The query is categorized as "(Containment, Element, Computed)," since it selects the data of the selectd objects having their valid-times contained in an element-valued argument computed from other stored facts. In order to find out such an element, two temporal joins must be effected: one between the relations* `emp` *and* `dept`*, to find out the department data of the selected employee, and the other between the relations* `dept` *and* `emp` *again, to find out the department manager data. The two joins can be categorized as "(=, Foreign)" and "(Containment, Element, Computed)", because the join attribute equality and the overlap of the valid-times must be ensured. The intersection of the valid-times of the histories joined, with the final local restriction* `emp.Gender=F` *on the manager data, gives the desired temporal element.*

Query Q 4.9.4: Find the department name, the manager name and the periods, for which a department manager earned more than one third of the department budget

Answer: "(Toy, Di, 1/1/87 – now)."

Category: (Projected, Not Empty) / (Containment, Element, Computed) / (<>, Arbitrary)

The only stored information about department managers is that Di has directed the Toy department from 1/1/82. She earned more than one third of the department budget from 1/1/87 to the present (cf. $50K > $100K/3).

For each department, the requested data are retrieved by means of a temporal join between relations `dept` *and* `emp`*. The categorizations of the temporal join conditions are the following: "(=, Foreign)" for the equality* `emp.Name=dept.Manager`*, "(Containment, Element, Computed)" for the overlap of the salary valid-times with the valid-times he/she was manager, "(<>, Arbitrary)" for the further comparison between department budget and manager salary.*

Query Q 4.9.5: Find the name and the salary history of the employees in the periods they earned as much as their managers

Answer: "(Ed, $30K, 6/1/83 – 7/31/84)" and "(Ed, $40K, 2/1/86 – 8/31/86)."

Category: (Projected, Not Empty) / (Containment, Element, Computed) (Containment, Interval, Computed) / (=, Foreign) (=, Single) (<>, Single)

The only relationship directed-director between employees which can be retrieved from the data is Ed-Di, which held from 2/1/82 to 1/31/87 at the Toy department. Ed earned as much as Di from 6/1/83 to 7/31/84 ($30K) and from 2/1/86 to 8/31/86 ($40K).

The query retrieves salary histories of the employees who earned as much as the employee who was the manager of the department they worked in. Such employees, and the periods of interest, can be selected via a **cyclic** *temporal join between* `emp` *and* `dept`*. As a matter of fact, "(=, Foreign)" non-temporal predicates allow us to express the join conditions* `dept.Department=emp.Dept` *and* `emp.Name=dept.Manager` *as in Query Q 4.9.3, a "(=, Single)" non-temporal predicate allows us to express the furter join condition on salary equality between employee and manager, whereas "(Containment, Element, Computed)" temporal selection predicates allow us to join data versions synchronized along valid-time. An additional "(<>, Single)" condition is assumed in the last join to ensure selected employees do not have the same name as their managers.*

Query Q 4.9.6: When did one person earn a lower salary than another younger person, and who were those persons?

Answer: "Ed and Di, 02/01/82–05/31/82, 08/01/84–01/31/85, 09/01/86–01/31/87, 04/01/87–12/31/87", "Edward and Di, 01/01/88–01/01/90"

Category: (Projected, Element, Derived) / (Containment, Element, Computed) /

Query Q 4.9.7: When and who had the same salary for the longest time?

Answer: "09/01/86–01/01/90, Di"
Category: (Projected, Element, Derived) / (Containment, Element, Computed) /

Query Q 4.9.8: List Di's skills and salary histories during the time she was a manager.
Answer: "(Directing, 01/01/82–01/01/90)" and "(30K, 01/01/82–07/31/84), (40K, 08/01/84–08/31/86), (50K, 09/01/86–01/01/90)"
Category: (Projected, Element, Derived) / (Containment, Element, Computed) / (=, Constant)

Query Q 4.9.9: List the names and salary histories of all managers when they earned at least 36K.
Answer: "Di, ((08/01/84–08/31/86, 40K), (09/01/86–01/01/90, 50K))"
Category: (Projected, Element, Derived) / (Containment, Element, Computed) / (<>, Constant)

6.4.10 Class O3.S10 (Other, Element, Other)

Query Q 4.10.1: Find the budget history in the period from 1/1/82 to 12/31/84 and from 1/1/87 till now of the department Ed ever worked in
Answer: "(Toy, $150K, 2/1/82 – 7/31/84)," "(Toy, $200K, 8/1/84 – 12/31/84)," "(Toy, $100K, 1/1/87 – now)," "(Book, —, 1/1/82 – 12/31/84)" and "(Book, —, 1/1/87 – now)."
Category: (Projected, Not Empty) / (Containment, Element, Explicit) / (=, Foreign) (=, Constant)

Ed worked in the Toy and in the Book department. The budget history of the Toy department in the required period is: $150K from 2/1/82 to 7/31/84, $200K from 8/1/84 to 12/31/84 and $100K from 1/1/87 till now. The budget history of the Book department in the same period is not available from the stored data (we assume "nulls" to be retrieved).

The departments in which Ed ever worked can be found by means of a "(=, Constant)" predicate on the employee data. The collected values are then used to select budget histories in the relation `dept` *via a "(=, Foreign)" predicate. Eventually, portions of such histories are selected by means of a valid-time predicate categorized as "(Containment, Element, Explicit)" since it exctracts data with valid-times contained in the user-supplied temporal element "1/1/82 – 12/31/84 ∪ 1/1/87 — now."*

Query Q 4.10.2: Find the name and the budget history in 1984 and 1987 of the department being directed by Di
Answer: "(Toy, $150K, 1/1/84 – 7/31/84)," "(Toy, $200K, 8/1/84 – 12/31/84)" and "(Toy, $100K, 1/1/87 – 12/31/87)."
Category: (Projected, Not Empty) / (Containment, Element, Explicit) / (=, Constant)

In the periods from 1/1/84 to 12/31/84 and from 1/1/87 to 12/31/87, Di always directed the Toy department. The Toy budget was $150K from 1/1/84 to 7/31/84, $200K from 8/1/84 to 12/31/84 and $100K from 1/1/87 to 12/31/87.

The query is categorized as "(Containment, Element, Explicit)" since it selects data contained in an element-valued time constant supplied by the user (1/1/84 – 12/31/84 ∪ 1/1/87 – 12/31/87). "(=, Constant)" indicates the non-temporal selection of the department data where the manager is Di.

Query Q 4.10.3: Find the names of the department where Ed was working at the beginning of the years 1986 and 1987, and the periods Ed worked there.
Answer: "(Toy, 2/1/82 – 1/31/87)."
Category: (Projected, Not Empty) / (Containment, Element, Explicit) / (=, Constant)

On 1/1/86 and 1/1/87 Ed was working at the Toy department. He has been working in that department from 2/1/82 to 1/31/87.

The query can be answered in two steps. First, department names are selected if stored as data concerning Ed by means of a "(=, Constant)" predicate testing for the employee name. Second, the department names are retrieved together with their valid times if these contain the explicit element "1/1/86 ∪ 1/1/87", by using a "(Containment, Element, Explicit)" valid-time selection predicate.

Query Q 4.10.4: Find the name of the manager Ed had on 1984's Christmas and on his 27^{th} birthday, and the dates he/she began to be manager of their department,
Answer: "(Di, 1/1/82)."
Category: (Projected, Not Empty) / (Containment, Element, User-defined) / (=, Constant) (=, Single) (=, Foreign)

Ed's 27^{th} birthday was 7/1/82. On 7/1/82 and on 12/25/1984, he worked in the Toy department, directed by Di. Di started to direct this department on 1/1/82.

The query contains a "(Containment, Element, User-defined)" valid-time predicate selecting data whose valid-times contain an element-value argument computed from a user-defined time attribute. As a matter of fact, the element "7/1/82 ∪ 12/25/1984" is computed by means of a date explicitly supplied in the query and from the Ed's date of birth, selected via a

"(=, Constant)" predicate in relation `employee`*. The first argument used by the valid-time selection predicate is the element-valued period in which Ed worked in a department. This period can be evaluated from Ed's data by means of a non-temporal selection predicate "(=, Single)", which tests employee data for a department name, once the department names have been selected. Finally, if a department name qualifies, it is used — via a "(=, Foreign)" predicate and together with the valid-time selection predicate — to select the manager data (name and valid-time) in relation* `dept`*.*

Query Q 4.10.5: Find the department name, the then manager, the modification dates and the new values of the budget for every budget change occurred in 1984, 1986 and 1988.
Answer: "(Toy, Di, \$200K, 8/1/84)."
Category: (Complete, Not Empty) / (Containment, Element, Explicit) (Ordering, Interval, Computed) / (=, Single) (<>, Single)

No data are available about the Book department budget. The Toy department budget changed on 8/1/84 and 1/1/87. Only the first date qualifies for the query. The manager at this time was Di and the new budget value was \$200K.

The query is classified as "(Containment, Element, Explicit)" because it uses a valid-time predicate to select data whose valid-times are contained in an element-valued argument supplied in the query (i.e. "1/1/84 – 12/31/84 ∪ 1/1/86 – 12/31/86 ∪ 1/1/88 – 12/31/88"). Candidate data are snaphots of the department data taken on budget changes. Budget changes are detected by means of the comparison of two consecutive department data versions. The first one can be freely selected, whereas the second one is the version of the same department (equality of the key ensured by means of a "(=, Single)" predicate) that follows the first one (as ensured by a temporal "(Ordering, Interval, Computed)" predicate). The two versions qualify — and the validity beginning of the second one is selected as a candidate date — if their budget values differ, according to a "(<>, Single)" predicate.

A On the Benchmark Taxonomy

Angelo Montanari Elisa Peressi Barbara Pernici

In this appendix, we discuss some aspects about the problem of identifying actual entities in the real world within the proposed database instance.

First of all, it was said that our two employees are identified by ED and DI, respectively. Are those a sort of surrogates or not?
If they are, how can we use them in formulating queries and in results of queries? (They do not appear in the proposed taxonomy).
For instance, in the given examples, employee names are allowed to change over time and it is assumed that the system is responsible to mantain the links between the tuples referring to the same real-world entity. However there is no mean to distinguish in the query result between a person who has changed name from two different persons. If we consider the query Q 2.1.1 *"Find the names of employees that have been in a department named Toy for a shorter period than has DI"*, how can we establish that Ed and Edward are two different names of the same person, rather than being the names of two different persons?
A similar problem occurs if the user wants to know something about the whole story of an employee formulating a query knowing only his name at some point in time. It would be reasonable that the system will retrieve all the tuples with the specified employee name and the tuples with the previous/successive names of the same employee. But in this way we do not allow the system to retrieve the tuples which refer to the name the user has explicitly specified, *only.*

We think that it would be useful to support both kinds of queries:

- those where one refers to all the tuples referring the same entity even if he specifies only an attribute value valid at some point in time (e.g. *Find the salary of the employee ED*);
- those where one refers *only* to the tuples that precisely satisfy the selection predicate (e.g *Find the salary of the employee whose* **name** *is (or was) "Ed"*).

The problem is that we need a way to distinguish between the identifier ED (whatever it is) and the string "Ed" representing the name of an employee at a given time, both in query formulation and query results.

B Comments on the Benchmark

Shashi K. Gadia Sunil S. Nair

<u>The time format</u>

We suggest the use of instants 0, 1, ..., NOW instead of dates. This avoids unnecessary distaraction.

Identifiers

For better readability We suggest relation names start with lower case characters and attribute names start with upper case characters.

Database instance

The database instance needs improvement. It should also be listed in a tabular format.

Taxonomy

We do not endorse the taxonomy proposed in the main body of this document. We feel it trivializes the issues and is a major distraction. This does not mean that we ourselves have a taxonomy which would be acceptable to all. In our experience in temporal databases, we have learnt some rules of thumb, and that is all. Similarly, others may have their rules of thumb. Giving these rules of thumb a coherent form is desirable, but will be a challanging task. The basis of such a taxonomy should be in structure of English (a natural language). Instead, the proposed taxonomy already assumes the temporal query languages to have a specific form.

Important questions

1. Are *and*, *or* and *not* of English handled symmetrically?
2. How is $[\![r]\!]$ of our model incorporated in other models?

Queries

We would like the following queries to be included in the benchmark. The queries are natural queries that might be asked against a temporal database. Some of the queries bring out the features mentioned above under "Imporatant Questions".

1. Give history of employees who have worked in the Toy or Book department.
2. Give history of employees who worked in the Toy and Book department.
3. Give Name and Salary history of employees during the time that DI was working.
4. Give Name and Salary history of employees during the time that DI was *not* working.
5. Give salary history of employees who were working at least during 1/1/87 to 31/12/89.
6. Give managers of John.
7. What were the departments that had managers at least during 1/1/87 to 31/12/87?
8. Who were the managers at least during 1/1/87 to 31/12/87 and what departments did they manage?
9. Give information about employees during the time there was someone managing any department.
10. Give information from the `Emp` relation if there were employees at least during 1/1/86 to 1/1/87.

C Comments on the Benchmark

Richard Snodgrass

I note a few inconsistencies that should be easy to fix.

- Did Ed manage the Book department from 01-Apr-1987 to present? As the benchmark data stands, this department didn't ever have a manager, or a budget.

- In the valid-time selection-based taxonomy, there is not a place for computed values from user-defined time (an example of which appears as the last query in class O3.S2).

- The output-based taxonomy distinction between derived and imposed doesn't seem to identify all the important possibilities. A more refined substitute might be intersection (intersection of underlying valid-time components), union (analogous), derived (computed in some other way from the valid-time components), and imposed (computed from values other than the underlying valid-time components.)

D Proposal: Calendar Date Standard for Temporal Benchmarks

John Roddick

In use at the present time are a multitude of methods for specifying the date. They can be categorised as:

- Arithmetically convenient but intuitively difficult (e.g., Julian Dates, VMSdates, etc.).

- Ambiguous (e.g., dd/mm/yy versus mm/dd/yy, yymmdd versus ddmmyy, and dates with two digit years).

- Arithmetically inconvenient, but intuitively easy (e.g., "dd-mmm-yyyy," "ddmmmyyyy," and "mmmmmmmmm dd, yyyy").

For the purpose of a benchmark formats in the latter category must be used as a benchmark must be, above all, unambiguous. Formats in this latter set are characterised by the

- easy recognition of the day, month and year component,
- unambiguous representation of the data in each component, and
- easy identification as a date as opposed to some other molecular datatype.

The following standard is suggested:

dd-mmm-yyyy

where dd = day of the month,
mmm = alpha month from the set "Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec," and
yyyy = year with leading century.

The set of alpha months is chosen as the first three letters of the long form of the month name in English, thus "Jul" is used in preference to "Jly." Four digit years are adopted given the proximity of the turn of the century. The hyphen separator is arbitrary but is used to distinguish between the "/" common used in dd/mm/yy family formats.

E Comments on the Benchmark

Christian S. Jensen

This appendix is structured as a list of observations related to the development of the current version of the TSQL Benchmark. It is hoped that these observations may help produce an improved next version of the benchmark.

It should be stated clearly in the scope of a benchmark what types of data models, it is relevant to. Object-oriented data models and relational data models have partially incompatible characteristics. In particular, object-oriented models include the notion of *identity* while relational models do not and are based completely on the notion of *value*. Due to differences like this, it appears difficult for one benchmark to address all temporal data models.

Another important distinction among temporal data models is where time enters a model. For example, times are associated with tuples in some models and with attribute values in other models.

The scope of the current benchmark is only restricted explicitly in terms of types of queries and data allowed.

Consensus benchmarks are useful for avoiding issues related to conflicts of interest. Like performance benchmarks, semantic benchmarks possess potentials for conflicts of interest. Ignoring this aspect may result in benchmarks that are not accepted by the community.

Benchmarks made by users and reflecting specific applications complement generic benchmarks made by researchers. User-developed benchmarks are good alternatives to consensus benchmarks developed by members of the research community.

The specificity of the benchmark database schema must be chosen carefully. Several alternatives exist for the level of detail when defining a database schema, and each has associated advantages and disadvantages.

At one extreme, the schema may be indicated indirectly by a description of what data is stored in the database. With this approach, the number of relation schemas and even the number of attributes may be left to the interpretation of the user. When this type of benchmark is applied to a query language, an initial task is to design an appropriate database schema which may contain the data indicated in the benchmark. It appears to be beneficial to leave schema design to the individual models when applying the benchmark to diverse models for which no common, best database schema exists. On the other hand, it may be more difficult to propose queries for the benchmark if the schema is unclear. Further, it becomes difficult to compare data models based on the benchmark when queries are expressed in terms of different schemas.

Next, the database schema may be specified in terms of a design model, such as the E-R model (or one of its extensions). Here, the user must first map the design model (E-R model) schema into a schema in the temporal data model at hand. The exact transformation is left unspecified and allows many models to use the benchmark.

Another option is to define the non-temporal aspects of the benchmark database schema. This was the approach chosen in the current benchmark. Thus, the number and names of relation schemas are specified, and specific attributes are given for each relation schema.

The desired level of detail of the schema and the intended scope of the benchmark are clearly interdependent. Specifically, it may be argued that a mismatch exists in the current benchmark where the database instance (designed after the schema was frozen) indicates the need for a data model with object-oriented features such as identity ("DI" and "ED" may be perceived as object identifiers). In contrast, the schema design perhaps appears to assume a purely value-based data model.

A final alternative is to completely design the database schema of the benchmark. This alternative appears to be problematic because the benchmark is intended to be used for comparing different models that do not use the same type of schema. This alternative restricts the scope to a narrow range of models.

A taxonomy of queries should help structure the presentation of benchmark queries and should help ensure a complete coverage of possibilities by suggesting a wide range of categories of queries. With a large benchmark, some kind of structuring of the queries is desirable—a good taxonomy must be useful in this respect. The taxonomy is also useful when dividing the task of proposing queries among multiple contributors.

There is a mismatch in precision between a taxonomy based on a formal query language and queries formulated in natural language. The current taxonomy has a "formal" approach in that is is inspired by formal query language constructs. Problems arise when the taxonomy is used for classifying queries formulated in natural language. Specifically, it has been observed that, when classifying natural language queries, it is necessary to (mentally) map these to some (imaginative) formal query language. Further, it is very often the case that no single, most obvious formal query language version exists of a natural language query.

A detailed, covering, easy-to-use, semantic taxonomy is desirable. While the existing benchmark is certainly detailed and extensive, it is neither easy to use nor semantic in nature.

F Comments and Queries

Patrick P. Kalua Edward L. Robertson

First, a general comment on the formulation of queries is presented. Then, several queries that did not fit the classification schema are listed.

General Comment : English easily allows the specialization from elements to intervals through the word *"continuous"*, but it unfortunately has no simple, natural way of expressing the reverse generalization. Hence we must include elements – discontinuous blocks of time – unless continuity is explicitly required.

Class 01.S11 (None)

Query: What skills did Ed and Edward have in common?
Answer: "Driving", "Filing" and "Typing"
Category: (Projected, None) / (Duration, Element, Computed) / (=, Constant) (=, Constant)

Class 02.S11 (None)

Query: How long was Ed's salary at least 40K?
Answer: "4 years 9 months"
Category: (None, Element, Derived) / (Duration, Element, Computed) / (=, Constant) (<>, Constant)

Class 03.S11 (None)

Query: When, how much and by whom was the highest salary by a female employee earned?
Answer: "09/01/86–01/01/90, 50K, Di"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (=, Constant)

Query: When and for which department was the budget at least 200K?
Answer: "(08/01/84–01/01/90, Toy)"
Category: (Projected, Element, Derived) / (Duration, Element, Computed) / (=, Constant)