Temporal Databases

Week 8

(Chapter 22 & Notes)

Introduction

- Temporal database contains historical, time-varying as well as current data.
  - Note: ‘historical record’ is a misleading term - a temporal database may contain reference to future.
- Extreme case: data is only inserted, never deleted from a temporal database.
- So far, we have studied the other extreme - i.e. ‘snapshot’ database.
- Distinguishing feature: the element of time.
Introduction ... continued

- Temporal data: encoded representation of timestamped facts.
  - Each tuple must include at least one timestamp.
    - Problem: What about queries that produce results that are not temporal? i.e. result of query is outside the domain of (temporal) database.
      - e.g. Get names of all people who have supplied something in the past.

- Redefine temporal database: database that includes, but is not limited to, temporal data.

Motivation

- Queries on time-varying data are difficult to express in SQL.
- Temporal databases provide build-in support for recording and querying such information.
- It is possible to use SQL to evaluate these queries, but performance is poor.
Examples of Temporal Queries

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Title</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>60000</td>
<td>Assistant Provost</td>
<td>1/1/93</td>
<td>5/31/93</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Assistant Provost</td>
<td>6/1/93</td>
<td>9/30/93</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Provost</td>
<td>10/1/93</td>
<td>1/31/94</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Full Professor</td>
<td>2/1/94</td>
<td>1/1/95</td>
</tr>
</tbody>
</table>

- Find the current salary for Bob
- Find the salary history for all the employees
- How long Bob has been a Full Professor?
- Find the names of all the Full Professors in the year of 1994.

Motivation

- Most applications manage temporal data.
- If a temporal database is used for such data:
  - Schemas, including integrity constraints are simpler.
  - Queries are simpler
- Application code is less complex
  - easier to understand
  - easier to produce
  - easier to maintain
Application

- Academic: Transcripts
- Accounting:
- Data Warehousing: Historical trends for analysis
- Financial: stock market analysis, Audit analysis
- Legal records: commercial laws change frequently
- Medical records: patient records, lab tests
- Process monitoring: chemical process
- Resource management and planning

Intervals

- An interval \([s,t]\) is a set of times from time \(s\) to time \(e\).
- Does interval \([s,e]\) represent an infinite set?
- Assumption: Timeline is a finite sequence of discrete, indivisible time quanta.
- Time Quanta: smallest unit of time system can represent.
- Timepoints/point: time unit considered indivisible for our purpose.
- An interval is treated as a single type, not as pair of separate values.
- Interval can be open/ closed w.r.t. start point/ end point.
  - eg. \([d04,d10],[d04,d11),(d03,d10),(d03,d11)\)
  - all represent the sequence of days from day 4 to day 10 inclusive.
Operators on Intervals

- Temporal predicate operators
  - $i_1$ BEFORE $i_2$ \( (e_1 < s_2) \)
  - $i_1$ MEETS $i_2$ \( (s_2 = e_1) \)
  - $i_1$ EQUALS $i_2$ \( (s_1 = s_2 \text{ AND } e_1 = e_2) \)
  - $i_1$ OVERLAPS $i_2$ \( (s_2 < s_1 < e_2 \text{ AND } e_2 < e_1) \)

- $i_1$ DURING $i_2$ \( (s_2 < s_1 \text{ AND } e_2 > e_1) \)
- $i_1$ STARTS $i_2$ \( (s_1 = s_2 \text{ AND } e_1 < e_2) \)
- $i_1$ FINISHES $i_2$ \( (e_1 = e_2 \text{ AND } s_1 > s_2) \)

- Additional operators
  - $i_1$ MERGES $i_2$ (\( i_1 \) MEETS \( i_2 \) OR \( i_1 \) OVERLAPS \( i_2 \))
  - $i_1$ CONTAINS $i_2$ (\( i_2 \) DURING \( i_1 \))
Scalar and Relational Operators on Intervals

- DURATION(i) - returns the number of time points in i
  - eg. DURATION ([d03,d07]) returns 5
- i1 UNION i2
  - returns [MIN(s1,s2),MAX(e1,e2)]
    if (i1 MERGES i2)
    otherwise undefined
- i1 INTERSECT i2
  - returns [MAX(s1,s2),MIN(e1,e2)]
    if (i1 OVERLAPS i2)
    otherwise undefined

Aggregate Operators

- UNFOLD(X):
  Where X is a set. The output is also a set.
  Used to generate time quantum intervals.
  - The unfolded form of X is the set of all intervals of the form
    [p,p] where p is a time point in some interval in X.
  e.g.
  X1 = {[d01,d01],[d03,d05],[d04,d06]}
  X2 = {[d01,d01],[d03,d04],[d05,d05],[d05,d06]}
  X3 = {[d01,d01],[d03,d03],[d04,d04],[d05,d05],[d06,d06]}
  X3 = UNFOLD (X1)
  = UNFOLD (X2)
Aggregate Operators

- **COALESCE**
  - The coalesced form of X is the set Y of intervals of the same type such that
    (a) X & Y have the same unfolded form.
    (b) no two distinct members i1 and i2 of Y are such that (i1 MERGES i2) is true.
  - e.g.
    X1 = { [d01,d01], [d03,d05], [d04,d06] }
    X2 = { [d01,d01], [d03,d04], [d05,d05], [d05,d06] }
    X3 = { [d01,d01], [d03,d06] }
    X3 = COALESCE (X1)
    = COALESCE (X2)

Relation Operators Involving Intervals

- **R COALESCE A/ P**
  - (Temporal Projection)
  - Get set of intervals which satisfy predicate P.
    Where R is a relational expression and A is an attribute (of some interval type) of the relation.
    The output is a relation with the desired attributes.
Example of COALESCE

- Get S#-DURING pairs for suppliers who have supplied parts P1 or P2 at some time.

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>During</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>[d04,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P7</td>
<td>[d05,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
<td>[d09,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P5</td>
<td>[d06,d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>[d02,d04]</td>
</tr>
<tr>
<td>S2</td>
<td>P9</td>
<td>[d03,d03]</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>[d08,d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P5</td>
<td>[d09,d10]</td>
</tr>
<tr>
<td>S3</td>
<td>P1</td>
<td>[d08,d10]</td>
</tr>
<tr>
<td>S4</td>
<td>P2</td>
<td>[d06,d09]</td>
</tr>
<tr>
<td>S4</td>
<td>P5</td>
<td>[d04,d08]</td>
</tr>
<tr>
<td>S4</td>
<td>P7</td>
<td>[d05,d10]</td>
</tr>
</tbody>
</table>

SP \{S#\_DURING\} COALESCE DURING/ P1 OR P2

Relation Operators Involving Intervals

- R UNFOLD A
  - Used to generate set of all datum (time samples) from a given relation R.
  - Where R is a relational expression and A is an attribute (of some interval type) of the relation.
  - The output is another relation with the same headers.
### Example

<table>
<thead>
<tr>
<th>S#</th>
<th>During</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>[d04,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>[d05,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>[d09,d10]</td>
</tr>
<tr>
<td>S1</td>
<td>[d06,d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d02,d04]</td>
</tr>
<tr>
<td>S2</td>
<td>[d03,d03]</td>
</tr>
<tr>
<td>S2</td>
<td>[d08,d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d09,d10]</td>
</tr>
<tr>
<td>S3</td>
<td>[d08,d10]</td>
</tr>
<tr>
<td>S4</td>
<td>[d06,d09]</td>
</tr>
<tr>
<td>S4</td>
<td>[d04,d08]</td>
</tr>
<tr>
<td>S4</td>
<td>[d05,d10]</td>
</tr>
</tbody>
</table>

### SP UNFOLD DURING

<table>
<thead>
<tr>
<th>S#</th>
<th>During</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>[d04,d04]</td>
</tr>
<tr>
<td>S1</td>
<td>[d05,d05]</td>
</tr>
<tr>
<td>S1</td>
<td>[d06,d06]</td>
</tr>
<tr>
<td>S1</td>
<td>[d07,d07]</td>
</tr>
<tr>
<td>S1</td>
<td>[d08,d08]</td>
</tr>
<tr>
<td>S1</td>
<td>[d09,d09]</td>
</tr>
<tr>
<td>S1</td>
<td>[d10,d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d02,d02]</td>
</tr>
<tr>
<td>S2</td>
<td>[d03,d03]</td>
</tr>
<tr>
<td>S2</td>
<td>[d04,d04]</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

---

### Temporal Difference Operator

- Used to answer negated queries.
- `R1 \ MINUS \ R2 \ ON \ A`
  - where R1 and R2 are relations of the same type and A is an attribute of some interval type common to these relations.
  - The output is a relation with the same heading as R1 or R2.
  - Equivalent to:
    ```
    ( (R1 UNFOLD DURING) \ MINUS \
      (R2 UNFOLD DURING) ) \ COALESCE \ DURING
    ```
Temporal Difference Operator

Example: Get S#-DURING pairs for suppliers who have been unable to supply any part at all sometime, where DURING designates the maximal continuous period during which S# was unable to supply a part.

- Answer: 
  \[
  \left( ( S \_DURING \{S#,DURING\} \text{UNFOLD DURING}) \right.
  \]
  \[
  \text{MINUS}
  \left( S \_DURING \{S#,DURING\} \text{UNFOLD DURING} \right)
  \]
  \[
  \text{COALESCE DURING}
  \]
  \[
  \text{OR:}
  \]
  \[
  S \_DURING \{S#,DURING\} \text{I-MINUS} S \_DURING \{S#,DURING\}
  \]
  \[
  \text{ON DURING}
  \]

Temporal Join*

• TJOIN (Temporal Join)
  - \[ R1 \text{JOIN} R2 = \langle x_1(i), \ldots, x_n(i), y_1(j), \ldots, y_m(j), P_{ij} \rangle \]
  - Where \( P_1i \cap P_2j \neq \emptyset \), \(<x_1(i), \ldots, x_n(i), P_1i>\) belongs to \( R1 \), \(<y_1(j), \ldots, y_m(j), P_2j>\) belongs to \( R2 \), \( P_{ij} = \left[ \max(a,c), \min(b,d) \right] \)
  - \( P_1i = [a,b], P_2j = [c,d] \)

• TNJOIN (Temporal Natural Join)
  - In addition to satisfying the conditions of TJOIN, there are an equal number of attributes \( x_p(s) \) of \( R1 \) and attribute \( y_q(s) \) of \( R2 \), such that each pair \( x_p, y_q \) are defined on the same domain, and \( p \in [1,n], q \in [1,m] \)
  - \[ R1 \text{JOIN} R2 = \langle x_1(i), \ldots, x_n(i), y_1(j), \ldots, y_m(j), P_{ij} \rangle \]
  - where \( R1. x_p = R2. Y_q \)

* Book by Tarsel et al, Temporal Databases, Theory, Design and Implementation, Benjmin/ Cummings Publishers
An Example of Temporal Natural Join

Table 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>60000</td>
<td>1/1/93</td>
<td>5/31/93</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>6/1/93</td>
<td>1/1/95</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Assistant Provost</td>
<td>1/1/93</td>
<td>9/30/93</td>
</tr>
<tr>
<td>Bob</td>
<td>Provost</td>
<td>10/1/93</td>
<td>1/31/94</td>
</tr>
<tr>
<td>Bob</td>
<td>Full Professor</td>
<td>2/1/94</td>
<td>1/1/95</td>
</tr>
</tbody>
</table>

Result of temporal natural join

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Title</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>60000</td>
<td>Assistant Provost</td>
<td>1/1/93</td>
<td>5/31/93</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Assistant Provost</td>
<td>6/1/93</td>
<td>9/30/93</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Provost</td>
<td>10/1/93</td>
<td>1/31/94</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Full Professor</td>
<td>2/1/94</td>
<td>1/1/95</td>
</tr>
</tbody>
</table>
Constraints Involving Intervals

- Consider the S_DURING relation.
  - What should be the primary key?
  - Is \{S#, during \} primary key sufficient?
  - Consider:

<table>
<thead>
<tr>
<th>S</th>
<th>Jones</th>
<th>Status</th>
<th>City</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
<td>[d02,d08]</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
<td>[d07,d10]</td>
</tr>
</tbody>
</table>

Primary key fails to prevent redundancy (overlap). (information about day 7 and day 8 is recorded twice)

Constraints Involving Intervals

- An example of a contradiction.
  - The status of S2 was both 10 and 20 on days 7 and 8
  - We need to add another constraint
    - If two distinct S_DURING tuples with the same S# values and other possible attributes have DURING values i1 and i2, and (i1 OVERLAPS i2), then those two tuples must be identical except possibly for their DURING values.
  - This can be enforced by keeping the relation unfolded at all times on attribute DURING.
    - i.e. \{S#,DURING\} as key for S_DURING UNFOLD DURING
    - The concept of ‘temporal key’
Update Operators Involving Intervals

- Use of update operators (INSERT, UPDATE, DELETE) on temporal relations is problematic.

Examples:

INSERT a tuple: (S2, JONES, 10, Paris, [d05, d06]) in S_DURING.
  - Cannot simply insert this tuple without violating constraints.
  - Need to delete/change existing tuples

UPDATE: Suppose that we discover that status of S2 was in fact 20 on day9.
  - Have to split S2’s [d07, d10] tuple into three tuples:
    [d07, d08] with status 10
    [d09, d09] with status 20
    [d10, d10] with status 10

DELETE: Suppose we discover that S3’s contract was terminated on day6, but reinstated on day9.
  - Need to split single S3 tuple in two.
Update Operators Involving Intervals

DELETE: Suppose we discover that S3’s contract was terminated on day6, but reinstated on day9
- Need to split single S3 tuple in two.

- The solutions outlined are unsatisfactory because they are specific to:
  - 1) The current values
  - 2) The updates desired