Temporal Data in SQL Server

Dejan Sarka

Thank you!
Agenda

- Temporal databases
- Optimization problem
- SQL Server temporal tables
  - Creating
  - Querying
  - Altering and dropping
- System-versioned tables problems

Temporal Databases

- Many relational databases show current state only
  - Constraints are valid in present, past and future time
  - E.g., supplier A is under a contract
- Temporal databases hold *time-stamped* propositions
  - *Since* predicate means ever since and not immediately before
  - E.g., supplier A is under a contract since a specific time point A
  - *During (or from ... to)* predicate means throughout and not immediately before or immediately after
  - E.g., supplier A is under a contract from time point A to time point B
- *Timeline* consists of discrete, indivisible time quanta
  - *Time quantum* is the smallest represented unit
  - Can represent time quanta with integers
Semi-Temporal Problems

- Example: suppliers, supplied parts

<table>
<thead>
<tr>
<th>S#</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>d04</td>
</tr>
<tr>
<td>S2</td>
<td>d07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>d04</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>d05</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>d08</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>d09</td>
</tr>
</tbody>
</table>

- No problem with PKs and FKs
- Need additional constraint: no supplier can supply a part before the supplier is under contract

Full-Temporal Problems (1)

<table>
<thead>
<tr>
<th>S#</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>d04</td>
<td>d10</td>
</tr>
<tr>
<td>S2</td>
<td>d02</td>
<td>d04</td>
</tr>
<tr>
<td>S2</td>
<td>d07</td>
<td>d10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>d04</td>
<td>d10</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>d05</td>
<td>d09</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>d08</td>
<td>d10</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>d09</td>
<td>d10</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>d03</td>
<td>d03</td>
</tr>
</tbody>
</table>

- Two candidate keys in each table (BCNF)
- \( \text{To} \) should not be less than \( \text{From} \)
- Cannot end contract on one day and begin on the next day
  - No abutting
Full-Temporal Problems (2)

- No supplier can be under two distinct contracts on the same day
- No overlapping
- No supplies if not under contract
  - $SP(S\#, From) \not\subseteq S(S\#, From)$
- Inclusion dependency:

\[ SP(S\#, From, To) \subseteq S(S\#, From, To) \]

- Joins should also consider inclusion

Valid and Transaction Times

- Bi-temporal databases support valid and transaction times
  - Valid times are human times
  - Transaction times are database times
  - Valid times are updatable
  - Transaction times are not updatable
- In a logical model, we should not need transaction times
  - Should be implemented in a RDBMS
  - Should be possible to query the state in any point in time
  - Support for valid times would also be appreciated
Overlaps: Classical T-SQL Solution

- Intervals represented with two columns – begin and end [b, e]
- Representation with integers
- Find all intervals [b, e] that overlap with a given interval [@b, @e]
- Classic predicate: WHERE b <= @e AND @b <= e

\[(i1 \xleftarrow{\text{overlaps}} i2) \equiv (b_1 \leq e_2 \land b_2 \leq e_1)\]

Optimizing Problem

- Optimal indexes exist
  - IDX1 – key b, included e
  - IDX2 – key e, included b
- Problem: two range predicates (WHERE b <= @e AND @b <= e)
  - Only one index is used
  - E.g., use IDX1, seek for b, and then use e as the residual predicate to filter while scanning the rows after the seek
  - Such a seek is very efficient if @e is high
  - Otherwise, IDX2 could be used
  - Indexes are efficient only if they eliminate most of the rows before the scan of the residual rows – if the [@b, @e] is close to the beginning or to the end of the timeline
  - Inefficient around the middle of the timeline
  - Must prevent parameter sniffing
Overlaps: Enhanced T-SQL Solution (1)

- A simple idea: try to use a single index to eliminate non-matching intervals from both sides
- Eliminating using the [lower] index
  - Eliminating intervals from the right side: just eliminate all intervals where the beginning is at least one unit bigger (more to the right) of the end of the given interval
  - In order to use the same index for eliminating from the left, you need to use the beginning of the intervals in the table in the `WHERE` clause of the query
  - Go to the left side away from the beginning of the given interval at least for the length of the longest interval in the table
  - The intervals that begin before the beginning of the given interval minus the length of the longest interval cannot overlap with the given interval

Overlaps: Enhanced T-SQL Solution (2)
Overlaps: Enhanced T-SQL Solution (3)

• Can get extremely efficient queries
• The solution is simple
• However, the performance drops substantially as soon as you have one very long interval in your table
• Therefore, the solution efficiency depends on the interval length distribution
  • The more uniform distribution, the more efficient solution

SQL Server Temporal Tables

• System-versioned tables store a full history of data changes
  • A validity period is stored in two datetime2 columns – SysStartTime and SysEndTime
• All versioning is automatic, no change in the code needed
  • Can be added to existing tables
• History is stored in an associated table
  • Historical table can be named, or take system name
  • Must have the same schema as the current table
  • The history table is a rowstore table with a clustered index on the period columns
Considerations and Limitations (1)

- The SysStartTime and SysEndTime columns must use the datetime2 data type
- The table must have a primary key; the historical table cannot use constraints
- To name the history table, specify the schema and table names
- History table compression defaults to PAGE
- The history table must reside in the same database as the current table
- Cannot alter or drop the history table
- No triggers on history table

Considerations and Limitations (2)

- System-versioned tables are not compatible with FILETABLE or FILESTREAM
- Columns with a BLOB data type, such as varchar(max) or image, can result in high storage requirements
- INSERT and UPDATE statements cannot reference the SysStartTime or SysEndTime columns
- You cannot directly modify data in the history table
- You cannot truncate a system-versioned table
- Merge replication is not supported
- No CDC or CT on the history table
Querying Temporal Tables

- System-versioned tables can be queried using the new FOR SYSTEM_TIME clause in the FROM part of a query
- Use one of the following four sub-clauses:
  - AS OF <date_time>
  - FROM <start_date_time> TO <end_date_time>
  - BETWEEN <start_date_time> AND <end_date_time>
  - CONTAINED IN (<start_date_time>, <end_date_time>)
- ALL to return everything

Creating a Temporal Table (1)

```sql
CREATE TABLE dbo.Employee
(
    EmployeeID int NOT NULL PRIMARY KEY CLUSTERED,
    ManagerID int NULL,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    SysStartTime datetime2
        GENERATED ALWAYS AS ROW START NOT NULL,
    SysEndTime datetime2
        GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
) WITH (SYSTEM_VERSIONING =
        ON (HISTORY_TABLE = dbo.EmployeeHistory));
```
Creating a Temporal Table (2)

- Can create your own history table
  - Can define your own indexes
    - Including columnstore indexes
  - With the HIDDEN clause, you can hide the new PERIOD columns
    - Avoid impacting on existing applications that are not designed to handle new columns (SELECT *)

Converting Existing Tables

- You can also alter an existing table and add attributes to it to convert it to a system-versioned temporal table
  - Add period columns
  - Define the SYSTEM_TIME period on them
  - Set the temporal attribute
  - You need to provide default constraints for both columns, since they must be non-nullable
  - You can use a date value from the past for the default constraint for the first period column; however, you cannot set values in the future
Altering Temporal Tables

- When you use ALTER TABLE to add a new nullable column, change data type or remove an existing column, system will automatically perform the action against both current and history tables.
- When you add a non-nullable column, you need to provide a default constraint as well.
  - You should take in account that this operation is an offline operation in all editions of SQL Server except in the Enterprise Edition.
  - Adding LOB or BLOB columns will cause a mass update in both the current and the history table in all SQL Server editions!
- You can also use ALTER TABLE statement to add or remove the attribute HIDDEN to period columns or to remove it.

Altering Temporal Tables Limitations

- Adding an IDENTITY or computed column.
- Adding a ROWGUIDCOL column or changing an existing column to it.
- Adding a SPARSE column or changing an existing column to it, when history table is compressed.
- If you need to perform schema changes to a temporal table not supported in the ALTER statement, you have to set its SYSTEM_VERSIONING attribute to false, to convert the tables to non-temporal tables, perform the changes and then convert back to temporal table.
Dropping Temporal Tables

- You cannot drop a system-versioned temporal table
- Both current and history tables are protected until the attribute SYSTEM_VERSIONING of the current table is set to ON
- When you set it to OFF, both tables automatically become non-temporal tables and are fully independent of each other
- Therefore, you can perform all operations against them that are allowed to you according to your permissions
- You can also drop period if you want to convert a temporal table to non-temporal one

Miscellaneous

- SQL Server 2017 adds retention policy
  - Enable on database level
  - Configure on table level
- System-versioned temporal tables are also supported for memory-optimized tables
  - You can assign or let SQL Server create a history table for your memory-optimized table
  - The history table must be a disk table
  - Recent history table is an internal memory-optimized table that handles changes in the current table synchronously and enables DMLs to be executed from natively compiled code
Temporal Tables Join

- Multiple tables scenario
  - Manual queries – write them correctly!
- Can use a view that joins the tables and filter the view for specific system time
  - Works well for a specific time
  - Works ...khm... For system time all
  - All possible combinations, even incorrect ones!

Temporal Tables as DW Dimensions

- System-versioned tables can be out of the box solution for Type 2 SCD problem
  - Can simplify the ETL process
- However, SSAS and Power BI do not support temporal tables yet
  - You are limited to query the data warehouse directly, with T-SQL queries
- System versioning works on row level
  - Might need column-level versioning
Thank you!