Using Oracle Extensibility Framework for Supporting Temporal and Spatio-Temporal Applications

Ravi Kothuri, Robert Hanckel, Aravind Yalamanchi
Oracle USA
{Ravi.Kothuri, Robert.Hanckel, Aravind.Yalamanchi}@oracle.com

Abstract

Databases such as Oracle, IBM, SQLServer are increasing their support for temporal and spatio-temporal data with every release. In this paper, we describe how Oracle users can extend the existing functionality in Oracle and ground their research by using Oracle's extensibility. To illustrate its applicability to temporal data, we present how a Map21[7] based index can be implemented using this framework.. Oracle uses the same extensibility framework to implement R-trees and Rules manager. We describe enhancements to Oracle' R-trees for spatio-temporal data, and real-time event processing using Oracle's Rules manager.

1. Time and Space in Commercial Databases

Time and space dimensions are inherent to a majority of data. Extensive research [14,15] has been performed to effectively model and capture the time dimension. The temporal and bitemporal research are examples of such efforts. Most commercial databases including Oracle, IBM, SQL Server provide efficient support for storing time as timestamps, and dates. Users can create indexes on these types of attributes to enable fast searching on such attributes. Beyond the basic support, different databases vary in their support for temporal data.

Oracle provides an interval type to store time intervals. However, currently no specific interval indexing is provided. A variety of temporal indexes have been proposed in literature. One can easily incorporate these indexes into Oracle using the extensibility framework. In Section 2, we briefly describe this framework. In Section 3 we describe our experiences with implementing a prototype interval indexing using this framework.

To support the space dimension, Oracle provides several data types and indexes for this purpose. Spatio-temporal support, however, is still limited. In Section 4 we discuss enhancements in Oracle's spatial indexes to support spatio-temporal data. In Section 5 we present the rule manager framework for processing time-sensitive events in Oracle.

2. Extensibility Framework in Oracle

Oracle's extensibility framework [10, 11] allows users to integrate their own domain-specific type with Oracle Server. Figure 1 shows the various components in the extensibility framework. It also shows example domains such as Spatial, Image, Text data that already utilize this framework.

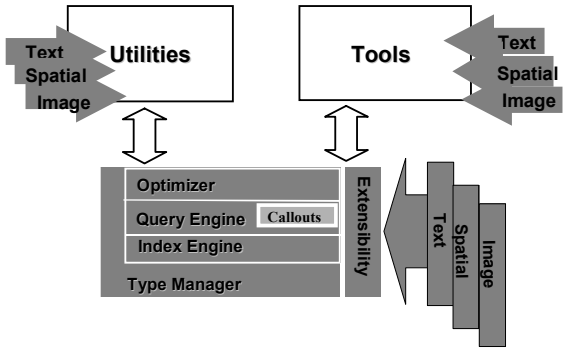


Figure 1: Oracle's Extensibility Framework

As shown in Figure 1, the extensibility framework allows users to:

- create user-defined data types (e.g., one for spatial, image, text, timeinterval),
- define an *indextype* and a corresponding *implementation type* to implement index-creation and maintenance operations on the user-defined datatype
- create *operators* on the data type that can be invoked in the WHERE clause of SQL statements. These operators identify whether

1530-1311/08 $25.00 © 2008 IEEE
DOI 10.1109/TIME.2008.29

or not queries on table columns of the defined data type satisfy a given criterion (e.g., whether spatial locations stored in a table column intersect a specified region of interest). The operators themselves are evaluated using the associated indextype in a sequence of *callbacks* referred to as the *start-fetch-close* interface.

- An associated statistics type to gather statistics [12] about the new indextype and utilize them for estimating the cost and the selectivity for an evaluation of an operator using the indextype.

3. Prototyping an Interval Index using Oracle Extensibility

In this section, we describe how to build a prototype for an interval index using Oracle's extensibility framework. First, we create a data type, say *timeinterval*, to store temporal intervals (you can also use the native interval type provided by Oracle). This type can have two timestamps, a *start_time* and an *end_time*[1]. Once this type is created, users can create tables with columns of this type.

Next, we define an indextype, called *timeindex*, to index *timeinterval* type column data. The implementation in this prototype is mainly based on the Map21 [7] indexing strategy. This approach is to fetch the interval data from the table column, divide them into groups as in Figure 2 based on interval length, and create B+ trees as described below:

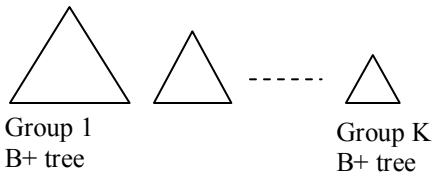


Figure 2: Partition Intervals on length and index each group using a B+ tree

- The Intervals are divided into groups where the i^{th} group stores intervals of length $B^{(i-1)}$ to B^i (where B is a parameter to index creation by default set to 2).
- The last group, K, will contain all intervals of length greater than $B^{(K-1)}$.
- The number of groups, K, can be either based on the precision of the timestamps, or specified as a user-defined parameter.
- For each group, a concatenated B+tree index on the start_time, end_time of intervals is created.

Next, we define a new operator called *intersect*. This operator takes a table column as the first argument, and an arbitrary query interval as second argument, and returns 'TRUE' if both intersect each other (see [7] for details). If the timeinterval column is indexed by the timeindex, then the intersect operator is evaluated using the start-fetch-close interface defined in the indextype.

- In the start call, the relevant groups for the given query interval are identified by comparing the query interval length with the maximum interval length allowed in each group. For each group in the relevant group, a cursor is setup to retrieve the intervals in the group that intersect the query interval.
- In the fetch call, each of the cursors that is setup is evaluated and fetched rows are returned to the server.
- In the close call, the cursors and other associated memory are released.

This approach not only provides an elegant indexing mechanism by leveraging the scalability and concurrency features of the B+ tree, but it is also amenable to parallel processing if the table is partitioned (and the index is created as a local index). As described in [7], performance comparisons indicate substantial improvements when compared to a single B+ tree. Table 1 shows intersect query results on 10M and 100M time intervals in a simulated warehouse inventory tracking system. Each interval represents the duration for which an item stays in the RFID warehouse during a 10-year period.

Table 1: Performance of the Map21 indexing approach v/s B+ trees in Oracle.

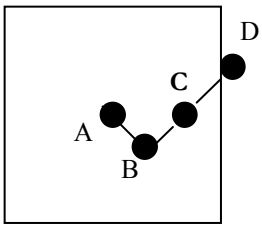
	10M	100M
B+ tree	4.77s	34.5s
Interval indexing (MAP21) prototype	0.12s	0.10s

[1] End_time being "current" can be treated as a special case. See [7] for details.

4. Enhancements for Spatio-temporal Data

For spatial data, Oracle supports the SDO_GEOMETRY type and allows users to create Quadtree or R-tree indexes on column data of that type (see [5] for a full list of details). The indexes are built in the extensibility framework described in Section 2. The storage, query and analysis functionality provided by Oracle Spatial is widely used in a variety of applications including cadastral management, utility and transportation networks, and insurance industries.

To cater to changes in spatial data, databases should be able to efficiently track the locations of objects (see [2] for details). Frequent updates to the data tables will invoke updates to the spatial indexes [5,6] such as R-trees. Since updates to indexes are costly, Oracle 11g allows a user to specify an 'expansion' parameter at index creation time. Using this 'expansion' parameter, Oracle creates a 'safe'MBR (minimum bounding rectangle) around each indexed location. Instead of indexing the actual locations of the objects, only the 'safe'MBRs are recorded as shown in Figure 3. As long as an object remains in its safe-MBR (e.g., locations A,B, and C), the location is not updated in the index. Once it moves out of the safe-MBR, the index is updated with the safeMBR of the current location.



Location	Index
A	safeMBR(A)
B	safeMBR(A)
C	safeMBR(A)
D	safeMBR(D)

Figure 3: Reducing Index Updates during Location Updates using safe (i.e., expanded) MBRs in Index.

As shown in Figure 3, this approach can avoid applying substantial number of location updates to the spatial index. However, the footprints stored in the index are larger than the exact location which could result in a slight degradation of query as the expansion parameter is increased. Note that this approach has been introduced in Oracle to support mainly modifications to otherwise static spatial data such as roads (as part of road construction/modification etc.). It can also be utilized for tracking moving objects in the database.

A more suitable approach for moving objects would be to update the index with an object-specific safe-extent every time an object moves out of the current one. The new safe-extent could be based on the object's direction (or the trajectory/road segment it is on [1,2]), its velocity and the time between location updates. To minimize the impact on the index, the extent may be restricted to axis-aligned-rectangles (but taking into account the underlying network or the direction). This approach works with existing spatial indexes with minimal modification. A more refined approach is to modify the Oracle R-tree algorithm to behave like the time-parameterized R-tree [3,9] for mobile data. In future work, the performance of these different proposed approaches may be compared using the COST framework [4] to the TPR-tree and its variants (which store the velocity parameters also in the internal nodes of the tree hierarchy).

5. Real-Time Event Processing

Most moving object applications are often time-sensitive and may need to react to events in real-time. For instance, as the current location of each moving object is updated (*event*) in the database, there may be a need to identify traffic patterns such as road congestions (*conditions*) and perform remedial measures (*actions*) such as re-routing to alleviate traffic. These events often translate to DML activity on relevant database tables. To facilitate such event-condition-action rule [13] processing in the database, Oracle provides the Rule Manager framework. This framework allows for correlation of multiple events in a user-specifiable temporal window to identify interesting patterns. The following code snippet describes one such rule to process 'update' events that modify the location (to currLoc on road roadID) of a vehicle (identified by vId). In the condition clause, this rule specifies the sdo_distance spatial function to restrict the processing to vehicles within 10-miles of Boston center and then group such vehicles by the roadId in the past 1 hour (specified for the *windowlen* attribute). If the number of vehicles in this time window is greater than 100 (as specified in the *having* attribute), it executes the specified action in the *then* clause.

```
ON
 VehicleTrack(vId, roadId, currLoc) vehicle
IF
<condition>
     <collection name = "vehicle"
        windowlen = "1" -- time window length
        groupby="roadId"
        having = "count(*) > 100">
       sdo_distance(cuuLoc,
        :BostonCenter, 'units=MILE') <10
     </collection>
   </condition>
```

```
THEN
  AvoidRoute(roadId) -- action
```

The prior code illustrates the use of spatial restrictions in a temporal event window. The framework also allows detecting scenarios involving non-occurrence of events within a time window (such as item not being shipped within 48 hours of placing an order). This functionality can also be utilized for a wider set of applications involving complex event processing.

6. Conclusions

In this paper, we described how to utilize the extensibility framework of Oracle to support interval indexing using the Map21 technique. This approach enables tight integration with the server and provides SQL-level query operators with little or no additional overheads. Examples of products in Oracle that use this extensibility framework include the R-tree and the Rules Manager. We then presented enhancements to R-trees to support moving objects data. Finally, we discussed how to utilize the Rules manager for real-time processing of events in a temporal window.

7. References

[1] Ralf Hartmut Güting, Victor Teixeira de Almeida, Zhiming Ding: Modeling and querying moving objects in networks. VLDB J. 15(2): 165-190 (2006).

[2] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, Dimitrios Gunopulos: Indexing spatiotemporal archives. VLDB J. 15(2): 143-164 (2006).

[3] Dan Lin, Christian S. Jensen, Beng Chin Ooi, Simonas Saltenis: Efficient indexing of the historical, present, and future positions of moving objects. Mobile Data Management 2005: 59-66.

[4] Christian S. Jensen, Dalia Tiesyte, Nerius Tradisauskas: The COST Benchmark-Comparison and Evaluation of Spatio-temporal Indexes. DASFAA 2006: 125-140.

[5] Ravi Kothuri, Albert Godfrind, Euro Beinat: Pro Oracle Spatial for Oracle 11g, Apress, November 2007.

[6] Ravi Kothuri, Siva Ravada: Spatio-Temporal Indexing in Oracle: Issues and Challenges. IEEE Data Eng. Bull. 25(2): 56-60 (2002).

[7] Mario A. Nascimento, Margaret H. Dunham: Indexing Valid Time Databases via B+-Trees. IEEE Trans. Knowl. Data Eng. 11(6): 929-947 (1999).

[8] Oracle 11g Database Documentation: http://www.oracle.com/technology/documentation/database.html.

[9] Mindaugas Pelanis, Simonas Saltenis, Christian S. Jensen: Indexing the past, present, and anticipated future positions of moving objects. ACM Trans. Database Syst. 31(1): 255-298 (2006).

[10] Jagannathan Srinivasan, Ravi Murthy, Seema Sundara, Nipun Agarwal, Samuel DeFazio: Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i. ICDE 2000: 91-100.

[11] Samuel DeFazio, Ramkumar Krishnan, Jagannathan Srinivasan, Saydean Zeldin: The Importance of Extensible Database Systems for E-Commerce. ICDE 2001: 63-70.

[12] Ravi Murthy, Ying Hu, Seema Sundara, Timothy Chorma, Nipun Agarwal, Jagannathan Srinivasan: Supporting Ancillary Values from User Defined Functions in Oracle. ICDE 2003: 151-162.

[13] Hanson, E. N. "Rule Condition testing and action execution in Ariel", SIGMOD Conference 1992 : 49-58.

[14] Richard T. Snodgrass: Towards a Science of Temporal Databases, TIME2007, 6-7.

[15] Michael D. Soo, Richard T. Snodgrass: Temporal Data Types. The TSQL2 Temporal Query Language 1995: 119-148