Geoinformatica
DOI 10.1007/s10707-009-0086-3

Tightly-coupled spatial database features in the Odysseus/OpenGIS DBMS for high-performance

Kyu-Young Whang · Jae-Gil Lee · Min-Soo Kim · Min-Jae Lee · Ki-Hoon Lee · Wook-Shin Han · Jun-Sung Kim

Received: 13 December 2007 / Revised: 30 March 2009 /
Accepted: 15 April 2009

Abstract Conventional object-relational database management system (ORDBMS) vendors provide extension mechanisms for adding user-defined types and functions to their own DBMSs. Here, the extension mechanisms are implemented using a high-level (typically, SQL-level) interface. We call this mechanism *loose-coupling*. The advantage of loose-coupling is that it is easy to implement. However, it is not preferable for implementing new data types and operations in large databases when high performance is required. We have earlier proposed the tight-coupling architecture (Whang et al. 2002, 2005) to satisfy this requirement. In *tight-coupling*, new data types and operations are integrated into the core of the DBMS engine in the *extensible*

K.-Y. Whang (✉) · J.-G. Lee · M.-S. Kim · M.-J. Lee · K.-H. Lee · J.-S. Kim
Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, South Korea
e-mail: kywhang@mozart.kaist.ac.kr

J.-G. Lee
e-mail: jglee@mozart.kaist.ac.kr

M.-S. Kim
e-mail: mskim@mozart.kaist.ac.kr

M.-J. Lee
e-mail: mjlee@mozart.kaist.ac.kr

K.-H. Lee
e-mail: khlee@mozart.kaist.ac.kr

J.-S. Kim
e-mail: jskim@mozart.kaist.ac.kr

W.-S. Han
Department of Computer Engineering, Kyungpook National University, Daegu, South Korea
e-mail: wshan@knu.ac.kr

Published online: 07 May 2009

type layer. Thus, they are supported in a consistent manner with high performance. This tight-coupling architecture is being used to incorporate information retrieval features and spatial database features into the Odysseus ORDBMS that has been under development at KAIST/AITrc for 19 years. In this paper, we introduce the tightly-coupled spatial database features of Odysseus/OpenGIS. By taking advantage of tight-coupling, Odysseus/OpenGIS provides excellent performance in processing spatial queries as well as flexible concurrency control and recovery on spatial data. We show the performance through extensive experiments. Finally, we present sample applications of a geographical information system (GIS) implemented using Odysseus/OpenGIS.

Keywords Tight-coupling · Object-relational DBMSs · Spatial DBMSs · Geographical information system (GIS)

1 Introduction

The amount of data on the internet has been growing at a rate of 10 times for every 3∼4 years, and many applications handling new data types have been emerging [2]. They include information retrieval (IR), spatial databases, data mining, and data streaming. Accordingly, DBMSs have been evolving to support these new applications. DBMS vendors provide extension mechanisms for adding new data types and operations to their own DBMSs. Examples are Cartridge [13] for Oracle and Extender [1] for IBM DB2. In these products, new data types are added by using user-defined types [11], their operations by using user-defined functions [11], and their indexes by using extensible indexing [20]. Here, user-defined types, functions, and extensible indexing are implemented through the high-level (typically, SQL-level) interface provided by the DBMS [20]. We call this mechanism *loose-coupling*.

In the loose-coupling architecture, the high-level interface causes the following problems. First, inter-process communication or dynamic linking overhead is incurred because operations on new data types are performed outside the core DBMS engine. Second, concurrency control and recovery in fine granularity are hard to perform because low-level functions of the DBMS engine cannot be fully utilized for new data types through the high-level interface [20].

We have earlier proposed the tight-coupling architecture [24, 25] to solve these problems. In the *tight-coupling* architecture, new data types and operations are implemented directly into the core of the DBMS engine (i.e., the storage system). Hence, the problems above do not occur in the tight-coupling architecture. This tight-coupling architecture is being used to incorporate IR and spatial database features into the Odysseus ORDBMS [25][1] that has been under development at KAIST/AITrc for 19 years. Whang et al. [25] have introduced the tightly-coupled IR features of Odysseus.[2]

[1]The Odysseus ORDBMS consists of approximately 450,000 lines of C and C++ precision codes.

[2]This work received the **Best Demonstration Award** from the IEEE ICDE 2005.

In this paper, we present the tightly-coupled spatial database features of Odysseus/OpenGIS. By taking advantage of tight-coupling, Odysseus/OpenGIS provides excellent performance in processing spatial queries as well as flexible concurrency control and recovery on spatial data. Through extensive experiments, we show excellence of Odysseus/OpenGIS in performance. Finally, we present sample applications of a geographical information system (GIS) implemented using Odysseus/OpenGIS.[3]

2 Tight-coupling architecture

In this section, we present the characteristics of the tight-coupling architecture [25] that is being used in the Odysseus DBMS.

2.1 The architecture of the Odysseus/OpenGIS DBMS

Figure 1 shows the architecture of the Odysseus/OpenGIS DBMS. Odysseus/OpenGIS consists of a storage system (*Odysseus/COSMOS*) and a query processor (*Odysseus/ OOSQL*). Odysseus/COSMOS is a sub-system that stores and manages objects in the database. It consists of Disk Manager, Small Object Manager, Large Object Manager, Index Manager, Cursor Manager, Recovery Manager, and Transaction Manager. Odysseus/COSMOS uses the Multi-Level Grid File (MLGF) [22, 23] for spatial indexing and supports fine or coarse granularity concurrency control and recovery on spatial data. We note that Odysseus/COSMOS contains the *extensible type layer* for tight-coupling. Odysseus/OOSQL is a sub-system that processes SQL queries. It consists of Query Analyzer, Query Plan Generator and Optimizer, and Query Plan Executor.

2.2 The extensible type layer

We propose to employ the notion of the extensible type layer [25] to facilitate tight-coupling. We define the *extensible type layer* as the layer that provides new data types, their operators, and their indexes at the level of the storage system. More specifically, the pseudo built-in types (Section 2.2.1) and type-management APIs (Section 2.2.2) are implemented in the extensible type layer.

We first define *loose-coupling* and *tight-coupling* depending on the location of the extensible type layer in Definition 1.

Definition 1 Loose-coupling is a mechanism of supporting new data types by locating the extensible type layer *on top of the query processor*; *tight-coupling* by locating the extensible type layer *inside the storage system*.

Figure 2 compares loose-coupling with tight-coupling. Loose-coupling in Fig. 2a is adopted by commercial DBMSs, and tight-coupling in Fig. 2b by the Odysseus/OpenGIS DBMS. In loose-coupling, Cartridge or Extender correspond to the extensible type layer.

[3] An extended abstract was presented as a demonstration paper in the IEEE International Conference on Data Engineering, Istanbul, Turkey, Apr. 2007 [26].

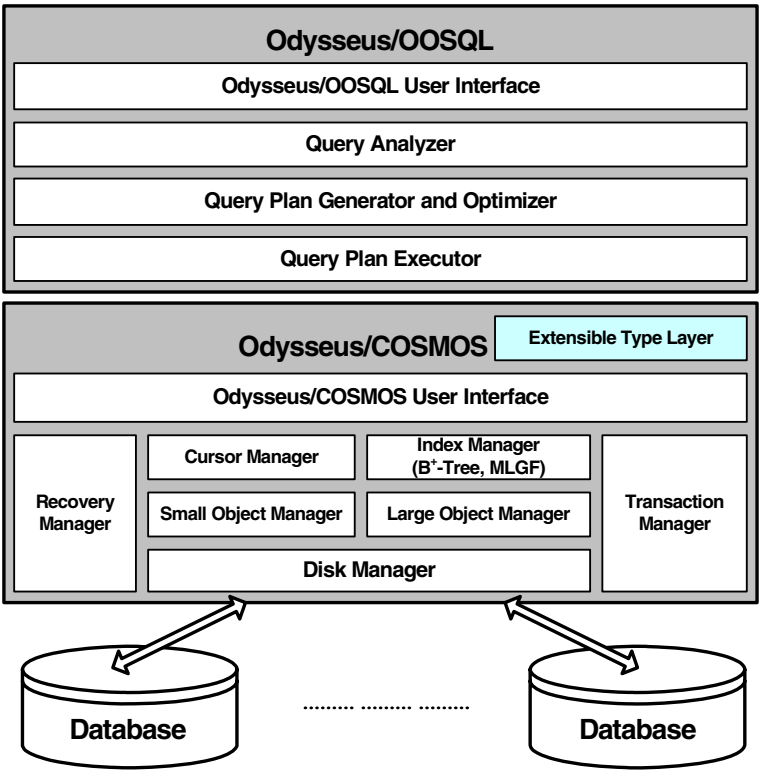


Fig. 1 The architecture of the Odysseus/OpenGIS DBMS [25]

2.2.1 The pseudo (built-in) type

We call a data type defined in the extensible type layer as the *pseudo built-in type* (simply, the *pseudo type*) [25]. The reason for including the term "pseudo" is as follows: although it is not a *real* built-in type, the pseudo built-in type has the performance benefit equivalent to that of a built-in type. In general, built-in types

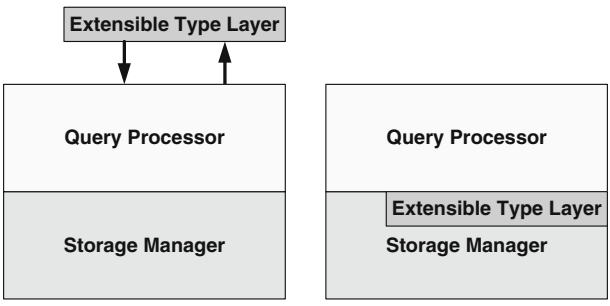


Fig. 2 Comparison between loose-coupling and tight-coupling (**a**, **b**)

(a) Loose-coupling. (b)Tight-coupling.

mean those specified in the SQL3 [11] standard, e.g., *int* and *varchar*. Pseudo built-in types are supported not from the *original* storage system, but from the *extended* storage system. In contrast, real built-in types are those supported from the original storage system without any extension.

We define that a data type has the performance benefit *equivalent* to that of a built-in type if using the data type does not require accessing the database catalog. Using pseudo types does not require accessing the database catalog because the information of pseudo types is hard-coded in the extensible type layer. In contrast, user-defined types [11] employed in loose-coupling do not have the performance benefit equivalent to those of built-in types according to this definition. The basic reason for this difference is that pseudo types are added into the storage system, while user-defined types are not.

2.2.2 Type-management API

The extensible type layer provides a programming interface. We call this interface the *type-management API*. The type-management API is implemented using the storage system API and is classified into the following categories.

- *Column Definition API*: to declare the type of a column to be a pseudo type
- *Column Manipulation API*: to insert a value into, delete a value from, and update a value in a column of a pseudo type, or to maintain an index if one is available
- *Index Definition API*: to create or delete an index on a column of a pseudo type
- *Index Scan API*: to support index scan over a column of a pseudo type
- *Sequential Scan API*: to support sequential scan over a column of a pseudo type
- *Operator API*: to support execution of the operations defined for a pseudo type
- *Statistics API*: to support statistics for query processing and optimization

Tight-coupling using the extensible type layer is implemented in the following two phases: (1) implementing the type-management API using the storage system API and (2) modifying the query processor so as to call the type-management API.

2.3 Advantages of tight-coupling

The tight-coupling architecture has three major advantages over the loose-coupling architecture in terms of performance, concurrency control, and flexible implementation.

1. Performance of query processing is superior. In loose-coupling, the overhead of executing an SQL query, such as those of parsing an SQL query and generating a query plan, is incurred because the add-on packages perform operations using SQL queries. In contrast, in tight-coupling, such overhead is not incurred because the extensible type layer performs operations using the storage system API. Some loose-coupling systems (e.g., PostGIS) exploit user APIs instead of SQL queries, and those systems are relieved from this problem.
2. Flexible concurrency control is possible on pseudo types. We are able to implement arbitrary protocols because acquiring or releasing a lock on a page can be done using the locking API of the storage system API. For example, it is possible to implement the link-based concurrency control protocol [12] on a spatial index such as the R-tree in tight-coupling, but it is hard in loose-coupling.

3. Implementable data types and operations are more flexible since the extensible type layer uses the storage system API. The storage system API offers more capabilities of the DBMS engine than the high-level interface does because the former is a lower-level interface compared with the latter. For example, the storage system API allows us accessing tuples at the byte (or page) granularity, while the high-level interface accessing only at the column granularity. This advantage also allows us to speed up query processing since we can implement tailored query processing algorithms and perform sufficient optimizations. Please see Section 4 for experimental results.

In addition, we are able to reduce the implementation overhead for tight-coupling by virtue of the extensible type layer. The reason for easy implementation is that implementations for new data types and operations are concentrated in the extensible type layer, and thus, modifications of existing source codes in other layers of the DBMS engine are minimized. Our experiences indicate that incorporating a new data type in the extensible type layer of the DBMS engine requires modification of approximately 20,000 lines of existing source codes in the query processor layer, which is manageable, being a very small proportion of the total source code. We estimate that the development effort is two person months, i.e., one expert programmer can complete this modification within 2 months.

3 Tightly-coupled spatial database features

In this section, we present the tightly-coupled spatial database features of the Odysseus/ OpenGIS DBMS. Section 3.1 explains the spatial types and operators supported by Odysseus/OpenGIS. Section 3.2 explains the structure of the MLGF spatial index. Section 3.3 presents the algorithms for spatial query processing.

In Odysseus/OpenGIS, users can specify the database schema using the spatial types and the MLGF index just in the same way as using nonspatial types and their indexes. Figure 3 shows the physical structure of a data record abiding by a schema involving the Point type, an MLGF index, the integer type, and a B^+-tree index. The Point type is specified just in the same way as the integer type is specified. Likewise, an MLGF index is specified in the same way as a B^+-tree index is.

3.1 Spatial types and operators

In this section, we summarize the spatial types and operators supported by Odysseus/ OpenGIS. These spatial types and operators conform to the OpenGIS [15] standard.

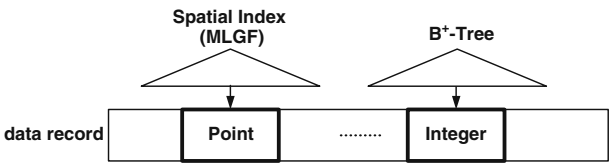


Fig. 3 The structure of a data record involving the Point type and an MLGF index

Table 1 Spatial types supported by Odysseus/OpenGIS

Types (classes)	Description
Geometry	A generic type for geometric objects
Point	A 0-dimensional geometric object representing a single location in coordinate space
Curve	A one-dimensional geometric object usually stored as a sequence of points
LineString	A Curve with linear interpolation between points
Surface	A two-dimensional geometric object
Polygon	A planar Surface defined by 1 exterior boundary and 0 or more interior boundaries
GeometryCollection	A collection of Geometry objects
MultiPoint	A collection of Point objects
MultiCurve	A collection of Curve objects
MultiLineString	A collection of LineString objects
MultiSurface	A collection of Surface objects
MultuPolygon	A collection of Polygon objects

Table 1[4] shows the spatial types, which are implemented as pseudo types in the extensible type layer.

The spatial operators are classified into three categories: relational operators, geometric operators, and miscellaneous operators [15]. *Relational operators* return true or false depending on whether a specified topological spatial relationship exists between two spatial objects. *Geometric operators* return a geometric measure of a spatial object or between two spatial objects. Miscellaneous operators include Buffer, ConvexHull, Intersection, Union, Difference, and SymDifference. Tables[4] 2, 3, and 4 show these three categories of operators, respectively.

3.2 Spatial index (MLGF)

The spatial index MLGF is implemented inside the storage system (Odysseus/COSMOS) as shown in Fig. 1. Odysseus/COSMOS supports the B^+-tree as a built-in index for nonspatial attributes and the MLGF [22, 23] as that for spatial attributes.

The MLGF is a balanced tree consisting of a multilevel directory and data pages [22]. Figure 4 shows the structure of the MLGF. A directory entry consists of a region vector and a pointer to a data page or a lower-level directory page. A region vector in an n-dimensional file consists of n hash values that uniquely identify the region. The MLGF uses an order-preserving hashing function to map attribute values to four-byte signed integers. The i-th hash value of the region vector is the common prefix of the hash values for the i-th attribute of all the records that belong to the region. For example, the region vector $<10, 0>$ in Fig. 4a represents the regions `E`, `F`, and `G` in Fig. 4b; the hash value '10' is the common prefix of the hash values for the first attribute of all the records in these regions, and the hash value '0' is the common prefix of the hash values for the second attribute.

The distinct characteristic of the MLGF is that it uses the *local splitting strategy* [23], which splits only the region where splitting is required rather than across the

[4]Most descriptions in these tables are borrowed verbatim from the OpenGIS specification [15].

Table 2 Relational operators supported by Odysseus/OpenGIS

Operators	Description
Equals	Equals(g_1, g_2) returns true if g_1 and g_2 are equal
Disjoint	Disjoint(g_1, g_2) returns true if the intersection of g_1 and g_2 is empty
Intersects	Intersects(g_1, g_2) returns true if the intersection of g_1 and g_2 is non-empty
Crosses	Crosses(g_1, g_2) returns true if the intersection of g_1 and g_2 results in a value whose dimension is less than the maximum dimension of g_1 and g_2, and g_1 and g_2 are not equal
Overlaps	Overlaps(g_1, g_2) returns true if the intersection of g_1 and g_2 results in a value of the same dimension as those of g_1 and g_2, and g_1 and g_2 are not equal
Touches	Touches(g_1, g_2) returns true if the points in common between g_1 and g_2 lie only in the union of the boundaries of g_1 and g_2
Within	Within(g_1, g_2) returns true if g_1 is completely contained in g_2
Contains	Contains(g_1, g_2) returns true if g_2 is completely contained in g_1

entire hyperplane. As a result, the directory growth is linearly dependent on the growth of the inserted records regardless of data distributions or data skew. Thus, the MLGF gracefully adapts to highly skewed data.

Since the MLGF is a point access method (PAM), the objects with extents are represented as points using corner transformation. Spatial join algorithms based on the MLGF and corner transformation were proposed in our earlier work and will be briefly mentioned in Section 3.3.2. Performance enhancement of these algorithms over the ones using the R-tree was shown in the references [14, 19].

3.3 Spatial query processing

3.3.1 Region query

A *region query* finds all objects satisfying a given spatial relationship (e.g., Intersects and Within) with a query region. Odysseus/OpenGIS processes region queries by using the MLGF as follows. Until a leaf page is reached, the algorithm follows the pointer to a lower-level directory page as long as the intersection of the region specified by the region vector and the query region is non-empty. The intersection of the two regions can be easily checked by comparing the minimum and maximum hash values of those regions for each dimension.

3.3.2 Transform-based spatial join

Spatial join finds all pairs of objects satisfying a given spatial relationship (e.g., Intersects and Within) between two sets of spatial objects [18]. Odysseus/OpenGIS uses the transform-based spatial join (TBSJ) algorithm proposed by Song et al. [19].

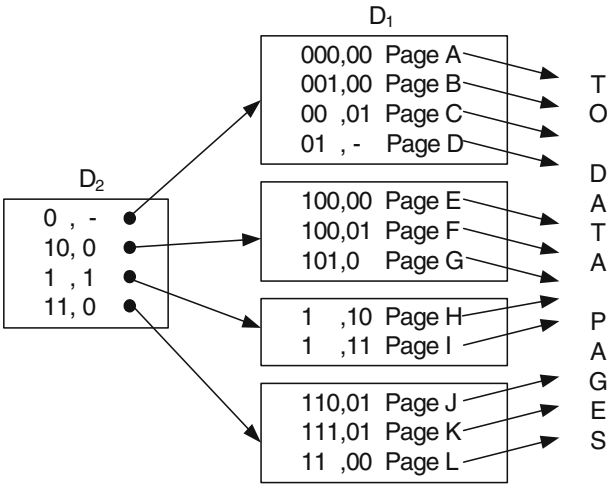
Table 3 Geometric operators supported by Odysseus/OpenGIS

Operators	Description
Area	Area(g_1) returns the area of g_1
Length	Length(g_1) returns the length of g_1
Distance	Distance(g_1, g_2) returns the shortest distance between g_1 and g_2

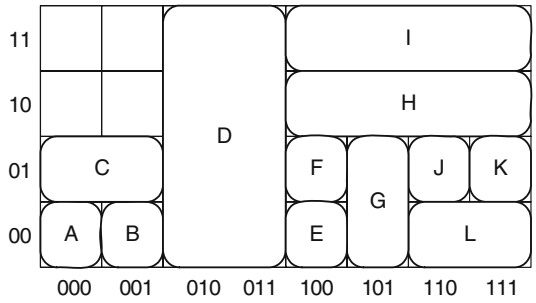
Table 4 Miscellaneous operators supported by Odysseus/OpenGIS

Operators	Description
Buffer	Buffer(g_1, d) returns a geometry defined by buffering a distance d around g_1
ConvexHull	ConvexHull(g_1) returns a geometry that is the convex hull of g_1
Intersection	Intersection(g_1, g_2) returns $g_1 \cap g_2$
Union	Union(g_1, g_2) returns $g_1 \cup g_2$
Difference	Difference(g_1, g_2) returns $g_1 - g_2$
SymDifference	SymDifference(g_1, g_2) returns $(g_1 - g_2) \cup (g_2 - g_1)$

The TBSJ algorithm transforms spatial objects with extents into points without extents using corner transformation [14, 19], and then, performs spatial join. Since this algorithm deals only with points but no extents, global optimization is relatively simple compared with existing algorithms [14]. *Corner transformation* transforms a spatial object in the n-dimensional original space (o-space) into a point in the



(a) The structure of a two-level MLGF directory.



(b) The regions represented by directory entries in D_1.

Fig. 4 A two-level MLGF directory and its domain space partition (**a**, **b**)

$2n$-dimensional transform space (t-space). In corner transformation, the coordinates of a point in the $2n$-dimensional t-space are determined by the minimum and maximum values of the MBR on each of the n axes in the o-space. For example, a one-dimensional object whose minimum and maximum values on the x-axis are lx and rx, respectively, is transformed into the point (lx, rx) in the two-dimensional t-space.

An operation that finds spatial objects satisfying a given spatial relationship in the o-space is transformed into an operation that finds points contained in a certain region in the t-space [14, 19]. We refer to this region as the *spatial join window* (*SJW*). The TBSJ algorithm optimizes the sequence of accessing disk pages in the SJW to minimize the amount of disk I/O's while performing spatial join.

3.3.3 k-Nearest neighbor search

k-Nearest neighbor search finds the first k spatial objects nearest to a given query point [10]. The k-nearest neighbor search algorithm implemented in Odysseus/ OpenGIS is simple, consisting of two steps. In the first step, the algorithm visits the data page containing a query point, and then, selects a *candidate object* whose z-order [17] value is closest to that of the query point. In the second step, the algorithm executes a region query over the region where spatial objects nearer to the query point than the candidate object may exist. This region is a circle whose center is the query point, and whose radius is the distance between the query point and the candidate object. If more than k spatial objects are found in this region, the k-nearest objects among them are selected as the k-nearest neighbor. Otherwise, the algorithm doubles the radius of the region and repeats the second step.

3.3.4 Advantages of tightly-coupled spatial algorithms

The tightly-coupled spatial algorithms in Odysseus/OpenGIS have advantages over loosely-coupled ones for the following reason. In tight-coupling, we can fully implement tailored spatial query processing algorithms and perform sufficient optimization since the spatial index structure can be accessed directly using the storage system API. However, in loose coupling, we are not able to integrate spatial query processing algorithms if they require a specific tree traversal. For example, in order to implement transformation-based join, sophisticated tree traversal must be controlled by the join algorithm to allow global optimization. The loosely coupled algorithms cannot implement such tailored spatial join due to inflexibility of the extensible interface of the loosely-coupled architecture. Therefore, the performance improvement of the tightly-coupled spatial algorithms becomes more prominent when optimization techniques implemented have an effect on spatial queries.

4 Performance evaluation

In this section, we evaluate the performance of the tightly-coupled spatial database features of Odysseus/ OpenGIS. Section 4.1 describes the experimental data and environment. Section 4.2 the results for spatial database features.

Table 5 The relations used in the experiments

Relation name	Description	The number of tuples
SUBWAY	Subway lines	315
DISTRICT	Districts	849
APARTMENT	Apartments	37,821
STRUCTURE	Geometric objects that do not belong to any categories	78,260

4.1 Experimental setting

We compare the performance of the Odysseus/OpenGIS DBMS with that of PostGIS (http://postgis.refractions.net), which is a spatial database extension for the PostreSQL DBMS.

We construct a geographical information system (GIS) to evaluate the performance of spatial database features. We use the map data of Seoul, the capital of Korea. The data contain approximately 850,000 spatial objects; among them, 250,000 objects represent buildings, and 600,000 objects the center lines of roads. The size of the source data is approximately 40 MBytes. Table 5 shows the relations used in the experiments, and Tables 6, 7, 8 and 9 show the schemas of the relations. We create an MLGF index on the `geometry` attribute of each relation in Odysseus/OpenGIS, and an R-Tree index in PostGIS. The relations are not clustered. Using this GIS, we compare the performances of three kinds of queries: (i) region queries, (ii) spatial join queries, and (iii) *k*-nearest neighbor queries.

We measure cold start time and warm start time. *Cold start* time is defined as the wall clock time for executing a query when no data residing in the DBMS buffer. *Warm start* time is defined as the wall clock time for executing a query when all the relevant data residing in the DBMS buffer.

To make the comparison as fair as possible, we use the same parameter values for Odysseus/OpenGIS and PostGIS. For both Odysseus/OpenGIS and PostGIS, we set the page size for data and indexes to be 4 KBytes and the buffer size to be 32 MBytes.

We conduct all the experiments on a Sun Blade 2000 workstation with 900 MHz CPU and 2 GBytes of main memory. Sun Blade 2000 is running on the operating system Solaris 8. Disks are installed in a disk array Sun StorEdge T3+. The workstation and the disk array are connected through an optical channel whose bandwidth is 200 MBytes/s. The controller of the disk array has 1 GBytes of cache, but we disable the cache to avoid the effect of the disk array cache. The transfer rate of the disks installed is 59∼118 MBytes/s (59.5 MBytes/s on the average).

Table 6 The schema of the SUBWAY relation

Attribute name	Attribute type	Description
name	Varchar	The name of a subway line
geometry	LineString	The line segments composing a section of a subway line

Table 7 The schema of the DISTRICT relation

Attribute name	Attribute type	Description
`name`	Varchar	The name of a district
`geometry`	Polygon	The polygon representing a district

4.2 Spatial query performance

Region queries Figure 5 shows the wall clock time for processing region queries. We execute queries that find buildings larger than a specific size within a given rectangular query region. We vary the ratio of the size of the query region to that of the entire domain space as 0.1%, 0.2%, 0.4%, 0.8%, 1.6%, 3.2%, and 6.4%. Figure 5 shows that Odysseus/OpenGIS improves the performance by 1.0∼1.9 times at cold start and by 2.0∼2.5 times at warm start compared with PostGIS. The primary reason for this advantage is that, in Odysseus/OpenGIS, the spatial types, operators, and index (MLGF) are implemented directly into the core of the DBMS engine. In contrast, in PostGIS, only the spatial index (R-Tree) is implemented directly into the core DBMS engine, and it is hard to implement sophisticated spatial algorithms (e.g., index-level filtering) since the spatial operators reside outside the core DBMS engine.

Spatial join queries Figure 6 shows the wall clock time for processing spatial join queries. We execute queries that find districts overlapping with the Seoul subway lines 1∼8, respectively. There are eight subway lines and 849 districts in the data set. A subway line consists of multiple line segments: 39.4 tuples (sections) per subway line on average, and each tuple consists of 9.6 line segments on average. A district is represented by a polygon, and each polygon consists of 79.3 points on average. Thus, spatial join is done between 3023 line segments and 849 polygons. Figure 6 shows that Odysseus/OpenGIS improves the performance by 1.1∼2.8 times at cold start and by 1.3∼3.2 times at warm start compared with PostGIS.

We also execute a large spatial query that finds geometric objects in the STRUCTURE relation (having 78,260 tuples) containing apartment objects in the APARTMENT relation (having 37,821 tuples). In fact, the two relations are disjoint, resulting in an empty result. For this query, Odysseus/OpenGIS improves the performance by 9.5 times at cold start and by 10.2 times at warm start compared with PostGIS. The reason why Odysseus/OpenGIS outperforms PostGIS is that Odysseus/ OpenGIS uses the sophisticated transform-based spatial join algorithm [19] implemented in the core DBMS engine, but PostGIS uses a naive index nested-loop join algorithm which invokes user-defined geometry functions outside the core DBMS engine. The transform-based spatial join allows global optimization and cannot be implemented in PostGIS through the extensible interface.

Table 8 The schema of the APARTMENT relation

Attribute name	Attribute type	Description
`name`	Varchar	The name of an apartment
`geometry`	Polygon	The polygon representing an apartment

Table 9 The schema of the STRUCTURE relation

Attribute name	Attribute type	Description
`name`	Varchar	The name of a geometric object
`geometry`	Geometry	A geometric object

k-Nearest neighbor queries Figure 7 shows the wall clock time for processing *k*-nearest neighbor queries. We execute queries that find the *k* buildings nearest to a given query point. Since PostGIS does not support *k*-nearest neighbor queries, we have implemented it through the extensible interface of GIS using the same algorithm as our algorithm in Section 3.3.3. We vary the value of *k* as 1, 4, 16, 64, and 256. Figure 7 shows that Odysseus/OpenGIS improves the performance by 1.2∼1.5 times at cold start and by 1.3∼1.5 times at warm start compared with PostGIS. In addition, the wall clock time of Odysseus/OpenGIS increases rapidly when *k* increases from 64 to 256. The reason is that a region query is executed again by doubling the radius (δ) if there are less than *k* objects in the region constructed by the candidate object and the query point. We find out that no doubling occurs when $k \leq 64$, while doubling occurs once when $k = 256$. The performance advantage of Odysseus/OpenGIS is due to the tight-coupling nature of the system.

5 Demonstration

We present sample applications [26] of a geographical information system (GIS) implemented using Odysseus/OpenGIS to show excellence of the tightly-coupled spatial database features. We store approximately 850,000 spatial objects in the database; among them, 250,000 objects represent buildings in Seoul, and 600,000 objects the center lines of roads in Seoul. Our demo system runs on a PC with 2.5 GHz CPU and 1 GBytes of main memory. We provide a graphical user interface using the Tcl/Tk package.

Our sample applications support four kinds of queries as follows.

(1) Finding the *k* buildings nearest to a point (*k-nearest neighbor search*): We execute the *k*-nearest neighbor search algorithm explained in Section 3.3.3.

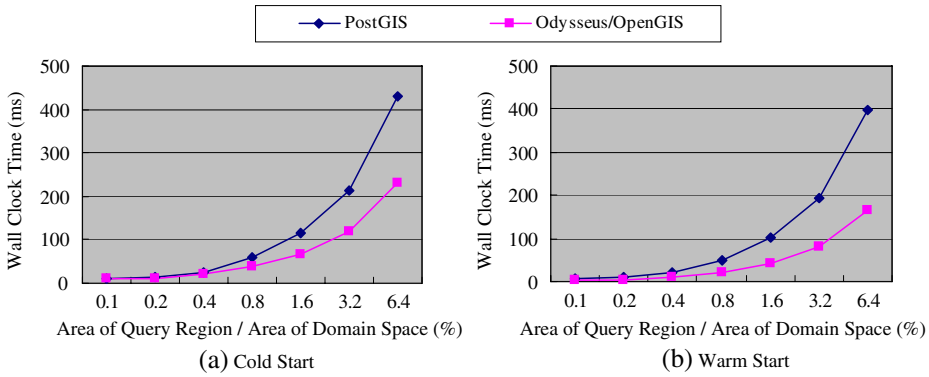


Fig. 5 The wall clock time for processing region queries (**a**, **b**)

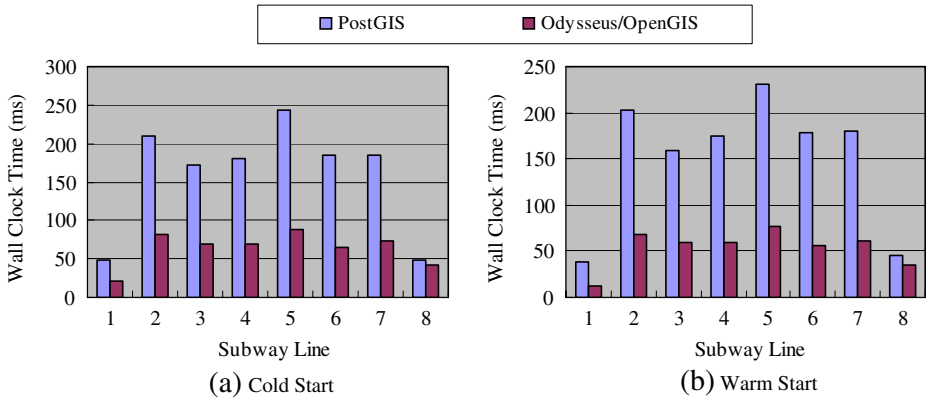


Fig. 6 The wall clock time for processing spatial join queries (**a**, **b**)

(2) Finding the nearest neighbor of every point on a path (*continuous nearest neighbor search*): We find the points where there is a change of neighborhood (called *split points*) by using the scheme proposed by Tao et al. [21], and then, find each nearest neighbor.
(3) Finding the districts overlapping with a subway line (*spatial join*): We execute the spatial join algorithm explained in Section 3.3.2.
(4) Finding the shortest path between two points (*shortest path search*): We find the shortest path incrementally using the *hierarchical graph* proposed by Chan and Zhang [4]. This algorithm significantly reduces memory and computation requirements since the entire graph need not be loaded into main memory at the same time unlike Dijkstra's algorithm.

Figure 8a–d show the results of executing the queries (1)–(4) above, respectively. In Fig. 8a, our demo system highlights the k buildings nearest to the point selected.

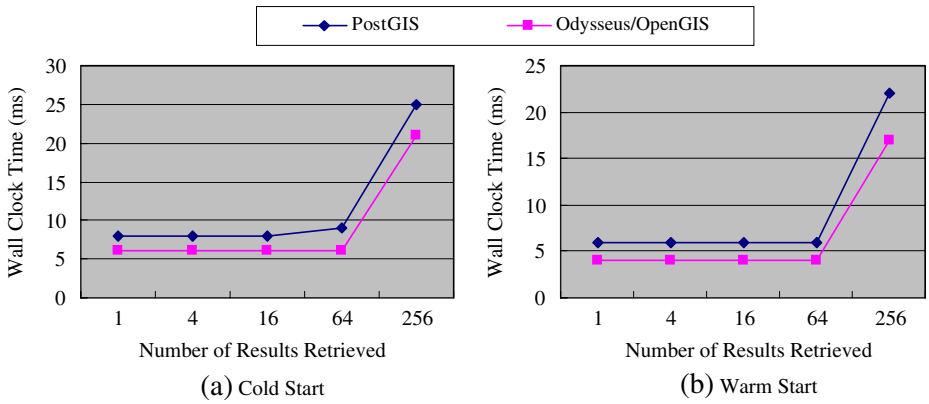
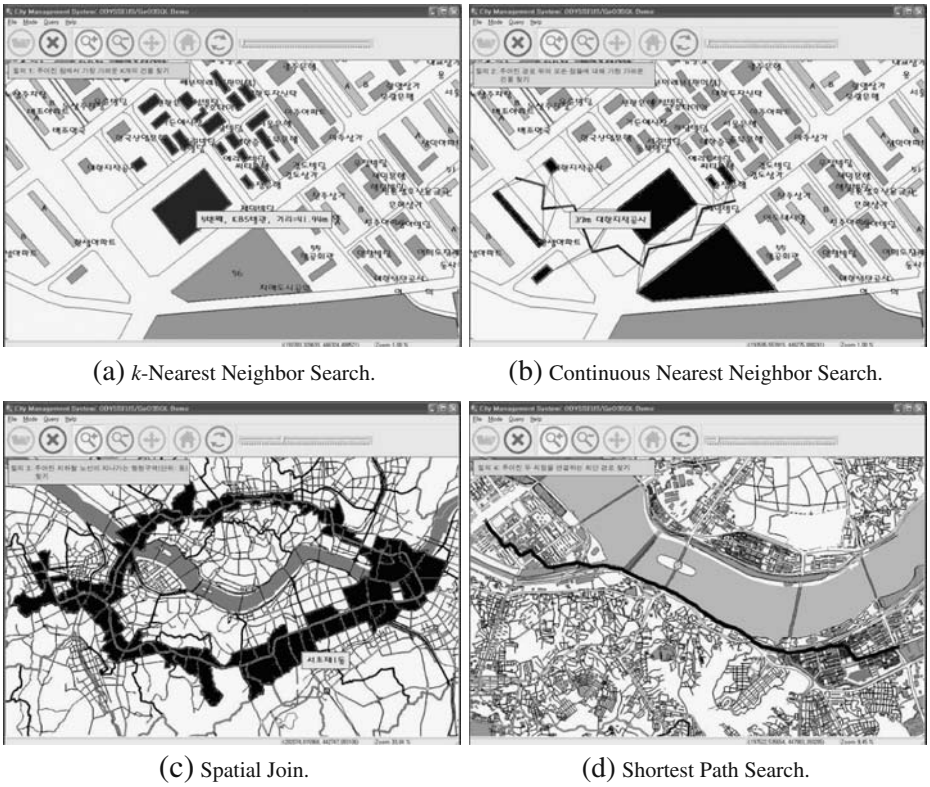


Fig. 7 The wall clock time for processing k-nearest neighbor queries (**a**, **b**)



(a) *k*-Nearest Neighbor Search. (b) Continuous Nearest Neighbor Search.

(c) Spatial Join. (d) Shortest Path Search.

Fig. 8 A demo system implemented using the Odysseus/OpenGIS DBMS

In Fig. 8b, it highlights the pairs of an interval on the path selected and the building nearest to the interval. In Fig. 8c, it highlights the districts overlapping with the subway line selected. In Fig. 8d, it highlights the shortest path between two points selected. Our demo system displays results very fast (in a fraction of a second) for all queries above. These results demonstrate excellence of the tightly-coupled spatial database features of Odysseus/OpenGIS.

6 Related work

6.1 Commercial DBMSs

We describe extension mechanisms provided by commercial DBMS vendors. In Cartridge and Extender, new data types are added by using user-defined types, and their operations by using user-defined functions [2, 7]. We do not explain user-defined types and functions in detail since they are defined in the SQL3 [11] standard and are already well known. However, Cartridge and Extender have differences in *extensible indexing* schemes that are employed for adding indexing schemes on new data types. Thus, we focus on extensible indexing schemes in this section.

6.1.1 Oracle cartridge

Oracle Cartridge is a package for adding application-specific features to the Oracle DBMS. Various kinds of Cartridge have been developed: Text Cartridge, Spatial Cartridge, interMedia Cartridge, etc.

Extensible indexing in Oracle is called *cooperative indexing* because the Cartridge module and the DBMS server cooperate to provide an indexing scheme [2, 6, 20]. Here, the Cartridge module is responsible for defining an index structure, maintaining the contents of the index, and searching the index. On the other hand, the DBMS server is responsible for storing the index. The Cartridge module contains the methods—implemented as external procedures—for index creation, insertion, update, and fetch operations. The DBMS server stores the index in a table. This newly added index is called the *domain index*.

Cartridge stores the domain index in a table [20]. Here, each tuple of a table stores one entry of the domain index. The reason for using a table structure is that the Oracle DBMS permits only the SQL interface for programming external procedures. Thus, the domain index is stored in a table so that it can be accessed only using the SQL interface.

Concurrency control on a domain index is performed using record-level locking just like on ordinary tables. However, record-level locking may not be suitable for concurrency control on a domain index [20]. We give an example using the R-tree. Cartridge stores an R-tree node in each tuple of a table [16]. When updating a node, we have to acquire exclusive locks on the nodes in the path between the root and the node to be updated. Then, the whole R-tree is locked because an exclusive lock is held on the tuple storing the root node. Hence, concurrency on the R-tree is decreased drastically.

6.1.2 IBM DB2 extender

IBM DB2 Extender is a package for adding application-specific features to the IBM DB2 DBMS. Various kinds of Extender have been developed: Text Extender, Spatial Extender, XML Extender, AIV Extender, etc.

Extensible indexing in IBM DB2 is analogous to that in Oracle. That is, the Extender module is responsible for providing operations related to the index, and the DBMS server is responsible for storing the index. However, Cartridge and Extender use different mechanisms for storing the index. Extender utilizes existing indexes such as the B-tree rather than a relational table in Oracle.

Extender employs the notion of *key transform* for extensible indexing [5]. Given the value of an index column of a user-defined type, one or more index key values are generated through key transform. Then, these generated index key values are stored in the B-tree index. Examples of key transform include transformation of a spatial object into a z-value or integer values representing the MBR. The main advantage of key transform is to allow us to use existing indexes such as the B-tree for indexing values of a user-defined type.

6.2 PostGIS

PostGIS (http://postgis.refractions.net) is a package for adding spatial features to the PostgreSQL DBMS like Spatial Cartridge of Oracle and Spatial Extender of IBM

DB2. It is an open-source software program that has been developed by Refractions Research and is released under the GNU General Public License. In PostGIS, new spatial data types, functions, and operators are implemented in C code and registered using SQL statements [8]. The PostgreSQL server then incorporates the C code (compiled into shared libraries) into itself through dynamic loading [8]. In effect, PostGIS spatially enables the PostgreSQL server. PostGIS uses the R-Tree index for spatial indexing, and the R-Tree index in PostGIS is implemented using the GiST (Generalized Search Tree) index [9] in PostgreSQL.

PostGIS can be viewed as a slight variation of loose-coupling since its spatial query processing algorithms are implemented using the user-level APIs but it supports the R-tree at the storage system level. Thus, as explained in Section 3.3.4, not every spatial query processing algorithm can be integrated into the DBMS—e.g., the depth-first R*-tree join algorithm [3]. PostGIS supports only tuple-based nested loop join algorithms due to inherent inflexibility of the loose-coupling architecture. As opposed to PostGIS, our system employs a true tight-coupling architecture in the sense that spatial object types are treated as first-class citizens both in the storage and query processing levels, thereby fully supporting tailored spatial query processing algorithms.

7 Conclusions

In this paper, we have presented the tightly-coupled spatial database features of Odysseus/ OpenGIS. Odysseus/OpenGIS provides excellent performance in processing spatial queries as well as flexible concurrency control and recovery on spatial data. In addition, Odysseus/OpenGIS is an ORDBMS and, at the same time, a spatial DBMS since it is tightly-coupled with the spatial database features.

We have explained the tightly-coupled spatial database features: the spatial types and operators conforming to the OpenGIS standard, the structure of the spatial index MLGF, and the query processing algorithms for region queries, spatial join queries, and *k*-nearest neighbor queries.

We have then performed extensive experiments using Odysseus/OpenGIS and PostGIS. The results for spatial queries show that Odysseus/OpenGIS outperforms PostGIS by 1.0∼2.5 times for region queries, by 1.1∼10.2 times for spatial join queries, and by 1.2∼1.5 times for *k*-nearest neighbor queries. These results demonstrate the superiority of the tight-coupling architecture of Odysseus/OpenGIS.

In summary, Odysseus/OpenGIS provides excellent performance in processing spatial queries by taking advantage of tight-coupling and has a capability for supporting various GIS applications with high performance.

Acknowledgements This work was primarily supported by the Korea Science and Engineering Foundation (KOSEF) through the National Research Lab Program funded by the Korea government (MEST) (No. R0A-2007-000-20101-0) and was partially supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation(KOSEF), grant number R11-2008-007-02004-0. Ki-Hoon Lee was partially supported by Brain Korea 21 Project, the School of Information Technology, KAIST in 2009.

References

1. Adler DW (2001) DB2 spatial extender—spatial data within the RDBMS. In: Proc 27th int'l conf on very large data bases, Rome, September, pp 687–690
2. Banerjee S, Krishnamurthy V, Murthy R (1999) All your data: the oracle extensibility architecture. Oracle White Paper, Oracle Corp, Redwood Shores, February
3. Brinkhoff T, Kriegel H, Seeger B (1993) Efficient processing of spatial joins using R-trees. In: Proc int'l conf on management of data. ACM SIGMOD, May, pp 237–246
4. Chan EPF, Zhang N (2001) Finding shortest paths in large network systems. In: Proc 9th ACM int'l symp on advances in geographic information systems, Atlanta, November, pp 160–166
5. Chen W, Chow J, Fuh Y, Grandbois J, Jou M, Mattos NM, Tran BT, Wang Y (1999) High level indexing of user-defined types. In: Proc 25th int'l conf on very large data bases, Edinburgh, September, pp 554–564
6. DeFazio S, Daoud AM, Smith LA, Srinivasan J, Croft WB, Callan JP (1995) Integrating IR and RDBMS using cooperative indexing. In: Proc 1995 ACM SIGIR int'l conf on information retrieval, Seattle, July, pp 84–92
7. Fuh Y, Deßloch S, Chen W, Mattos N, Tran B, Lindsay B, DeMichel L, Rielau S, Mannhaupt D (1999) Implementation of SQL3 structured types with inheritance and value substitutability. In: Proc 25th int'l conf on very large data bases, Edinburgh, September, pp 565–574
8. Hall GB, Leahy MG (2008) Open source approaches in spatial data handling. Springer, Berlin Heidelberg New York
9. Hellerstein JM, Naughton JF, Pfeffer A (1995) Generalized search trees for database systems. In: Proc the 21st int'l conf on very large data bases, Zurich, September, pp 562–573
10. Hjaltason GR, Samet H (1999) Distance browsing in spatial databases. ACM Trans Database Syst 24(2):265–318
11. ISO/IEC (1999) Database language—SQL3—part 2: foundation (SQL/Foundation). ISO/IEC 9075-2, December
12. Kornacker M, Banks D (1995) High-concurrency locking in R-trees. In: Proc the 21st int'l conf on very large data bases, Zurich, September, pp 134–145
13. Kothuri RKV, Ravada S, Abugov D (2002) Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. In: Proc 2002 ACM SIGMOD int'l conf on management of data, Madison, June, pp 546–547
14. Lee M, Whang K, Han W, Song I (2006) Transform-space view: performing spatial join in the transform space using original-space indexes. IEEE Trans Knowl Data Eng 18(2):1–16
15. Open GIS Consortium Inc (1999) OpenGIS simple features specification for SQL, rev 1.1, OpenGIS project document 99-049, May
16. Oracle (2002) Oracle spatial user's guide and reference release 9.2, March
17. Orenstein A, Merrett T (1984) A class of data structures for associative searching. In: Proc 3rd ACM SIGACT-SIGMOD symp on principles of database systems, Waterloo, April, pp 181–190
18. Seeger B, Kriegel H-P (1988) Techniques for design and implementation of efficient spatial access methods. In: Proc 14th int'l conf on very large data bases, Los Angeles, August/September, pp 360–371
19. Song J, Whang K, Lee Y, Kim S (1999) Spatial join processing using corner transformation. IEEE Trans Knowl Data Eng 11(4):688–698
20. Srinivasan J, Murthy R, Sundara S, Agarwal N, DeFazio S (2000) Extensible indexing: a framework for integrating domain-specific indexing schemes into Oracle8i. In: Proc 16th int'l conf on data engineering, San Diego, February/March, pp 91–100
21. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: Proc 28th int'l conf on very large data bases, Hong Kong, August, pp 287–298
22. Whang K, Krishnamurthy R (1985) Multilevel grid files. IBM research report RC11516, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, November
23. Whang K, Kim S, Wiederhold G (1994) Dynamic maintenance of data distribution for selectivity estimation. VLDB J 3(1):29–51
24. Whang K, Park B, Han W, Lee Y (2002) An inverted index storage structure using subindexes and large objects for tight coupling of information retrieval with database management systems. US Patent no 6,349,308, 19 February 2002, Appl no 09/250,487, 15 February 1999
25. Whang K, Lee M, Lee J, Kim M, Han W (2005) Odysseus: a high-performance ORDBMS tightly-coupled with IR features. In: Proc 21st int'l conf on data engineering, Tokyo, Japan, April, pp 1104–1105 (This paper received the best demonstration award)

26. Whang K, Lee J, Kim M, Lee M, Lee K (2007) Odysseus: a high-performance ORDBMS tightly-coupled with spatial database features. In: Proc 23rd int'l conf on data engineering, Istanbul, April, pp 1493–1494

Kyu-Young Whang graduated (Summa Cum Laude) from Seoul National University in 1973 and received the M.S. degrees from Korea Advanced Institute of Science and Technology (KAIST) in 1975, and Stanford University in 1982. He earned the Ph.D. degree from Stanford University in 1984. From 1983 to 1991, he was a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, NY. In 1990, he joined KAIST, where he currently is a KAIST Distinguished Professor at the Department of Computer Science. His research interests encompass database systems/storage systems, object-oriented databases, multimedia databases, geographic information systems (GIS), data mining/data warehouses, XML databases, and data streaming. He is an author of over 100 papers in refereed international journals and conference proceedings (and over 150 papers in domestic ones). He served as an IEEE Distinguished Visitor from 1989 to 1990, received the Best Paper Award from the 6th IEEE International Conference on Data Engineering (ICDE) in 1990, served the ICDE six times as a program co-chair and vice chair from 1989 to 2003, and served program committees of over 110 international conferences including VLDB and ACM SIGMOD. He was the program chair (Asia and Pacific Rim) for COOPIS'98, the program chair (Asia, Pacific, and Australia) for VLDB 2000, and a program co-chair of ICDE2006. He was the general chair of VLDB2006, PAKDD 2003, and DASFAA 2004. He twice received the External Honor Recognition from IBM. Dr. Whang is the Coordinating Editor-in-Chief of the VLDB Journal having served the editorial board as a founding member for thirteen years. He was an associate editor of the IEEE Data Engineering Bulletin from 1990 to 1993, Distributed and Parallel Databases Journal from 1991 to 1995, Int'l J. of GIS from 1994 to 2007, and IEEE TKDE from 2002–2006. He is on the editorial board of the WWW Journal. He was a trustee of the VLDB Endowment from 1998 to 2004 and currently is the steering committee chair of the DASFAA Conference and a steering committee member of the IEEE ICDE, PAKDD, and APWeb Conferences. He served the IEEE Computer Society Asia/Pacific Activities Group as the Korean representative from 1993 to 1997. Dr. Whang is a Fellow of the IEEE, a member of the ACM, and a member of IFIP WG 2.6.

Jae-Gil Lee is a postdoctoral researcher in IBM Almaden Research Center. Before joining IBM, he was a postdoc research associate in the Department of Computer Science, University of Illinois at Urbana-Champaign. He earned his M.S. and Ph.D. in computer science at Korea Advanced Institute of Science and Technology (KAIST). His research interests encompass spatio-temporal data mining, data warehousing, information retrieval and search engines, and DB-IR integration. He is currently working on acceleration of data warehouse queries.

Min-Soo Kim is a postdoctoral fellow of computer science at University of Illinois at Urbana-Champaign (UIUC). His research interests include network/graph data mining, bio-informatics, indexing & query processing, and information retrieval & search engines. He has a PhD in computer science from Korea Advanced Institute of Science and Technology (KAIST).

Min-Jae Lee received the BS degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 1995 and the MS and PhD degrees in computer science from KAIST in 1997 and 2004, respectively. Until November 2004, he was a postdoctoral fellow at the Advanced Information Technology Information Center, KAIST. In December 2004, he joined Neowiz, Co., Ltd., in Korea as a research staff member. His research interests include spatial databases, access methods, information retrieval, query processing, database systems, and storage systems.

Ki-Hoon Lee received B.S. (2000), M.S. (2002), and Ph.D. (2009) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). He is currently a postdoctoral researcher of Computer Science at KAIST. His research interests include XML and web databases, IR and search engines, query optimization, object-relational database systems, and spatial databases and GIS.

Wook-Shin Han received the B.S. degree in Computer Engineering from Kyungpook National University in 1994, and the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), in 1996 and 2001, respectively. He is currently a tenured associate professor in the Department of Computer Engineering at Kyungpook National University. In the past, he has worked as a post-doctoral researcher at IBM Almaden Research Center working on parallel progressive optimization. His research interests include query processing and optimization, simiarity search, XML databases, object-oriented/object-relational databases, and information retrieval. He published at major international journals and conferences, including VLDB, SIGMOD, ICDE, WWW, IEEE TKDE, and VLDB Journal. He is the co-PC chair of APWeb 2010 and the workshop chair of CIKM 2009. He is an editorial board member of several international journals.

Jun-Sung Kim received the B.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 2006. He is currently a Ph.D. Candidate in the Department of Computer Science at KAIST. His research interests include spatial databases, geographic information systems, information retrieval, and database systems.