OraGIST – How to Make User-Defined Indexing Become Usable and Useful

Carsten Kleiner, Udo Lipeck

Universität Hannover

BTW 2003, 27.02.2003
**The Situation (+)**

Object-relational DBMSs are extensible by:

- **user-defined datatypes (UDT)**
- together with corresponding query **operators**
- **user-defined indexing**
- together with corresponding user-defined optimizer tuning

⇒ ”data blades”, ”cartridges”, ”extenders”, ...
The Situation (−)

But user-defined indexing

• needs expensive implementation

• needs experimental selection for new query scenarios

• is restricted:
  a user-defined index supports only single operators, no combinations

• assumes that an operator is supported by at most one index
Our Proposal

• Use **combined datatypes and operators** to allow index support for attribute combinations

• Provide medium-dimensional index structures

• Apply a **generic indexing framework**
  * based on **generalized search trees (GiST)**
  * easily specializable into indexes for particular UDTs and operators
  * coupled with DB storage and index definition

⇒ prototypical tool **OraGiST** (GiST for Oracle)
Generalized Search Trees (GiST) — Class Hierarchy

GiST
- #insert(e:GiSTEntry, level:int)
- #chooseSubtree(e:GiSTEntry, level:int)
- #split(n:GiSTNode, e:GiSTEntry)
- #adjustKeys(n:GiSTNode)
- #delete(e:GiSTEntry)

OrderedGiST
- #findMin(q:predicate)
- #next(q:predicate, e:GiSTEntry)

BTreeGiST
- #consistent(e:GiSTEntry, q:predicate)
- #union(l:ListOfGiSTEntry)
- #penalty(e1:GiSTEntry, e2:GiSTEntry)
- #pickSplit(l:ListOfGiSTEntry)
- #compare(e1:GiSTEntry, e2:GiSTEntry)

RTreeGiST
- #consistent(e:GiSTEntry, q:predicate)
- #union(l:ListOfGiSTEntry)
- #penalty(e1:GiSTEntry, e2:GiSTEntry)
- #pickSplit(l:ListOfGiSTEntry)

RStarTreeGiST
- #penalty(e1:GiSTEntry, e2:GiSTEntry)
- #pickSplit(l:ListOfGiSTEntry)

SSTreeGiST
- #penalty(e1:GiSTEntry, e2:GiSTEntry)

RSSTreeGiST
- #penalty(e1:GiSTEntry, e2:GiSTEntry)
Index Usage

CREATE INDEXTYPE polygonRTree 
FOR overlaps(polygon,polygon) USING overlaps_fct;

CREATE TABLE county 
(id NUMBER, population NUMBER, ... 
 shape polygon, ...);

CREATE INDEX idx_geoCounty ON county(shape) 
INDEXTYPE IS polygonRTree PARAMETERS(...);

SELECT * FROM county 
WHERE overlaps(shape,polygon(rectangle(0,0,20,10)))) 
  = true;
Extensible Indexing in ORDBS

- Sequence of index operations over index lifetime (e.g.):
  - ODCIIndexCreate
  - ODCIIndexInsert
  - ODCIIndexDrop
  - for each tuple

- Sequence of index operations in query execution:
  - ODCIIndexStart
  - ODCIIndexFetch
  - ODCIIndexClose
  - until all tuples fetched
    - set up structures
    - read and modify
    - delete structures
OraGiST — Architecture and Functionality

libgist library

GiST

GiSTIndexFile

GiSTExtension

GiSTEntry

Oracle ORDBS

DBSExtensibleIndexing

DBSIndexTable

UserDefinedIndexStructure

DBSUserDefinedObject

OraGiST Extension

+getExtension()

+getQuery()

+needVerify()

TypeMap

+approximate()
OraGiST — Architecture and Functionality

libgist library

GiST

GiSTIndexFile

OracleExtensibleIndexing

OracleIndexTable

UserDefinedRTree

GeometryObject

Oracle ORDBS

GiST

RTree

RTreeEntry

getExtension

getQuery

approximate

OraGiST Toolbox
Case Study — Spatial/Thematic Analysis

- Typical query: *Find all counties overlapping a given window where the median rent is below ... and the population is higher than ...*

- Definition of combined datatypes and operators:

```sql
CREATE TYPE integerGeometry AS OBJECT (shape OGCGeometry, theme INTEGER);

CREATE TYPE twoIntegerGeometry AS OBJECT (shape OGCGeometry, theme1,theme2 INTEGER);

CREATE OPERATOR [two]BetweenOverlaps ... ;

CREATE TABLE county OF [two]IntegerGeometry;
```
Performance Evaluation on 2D Spatial Data
... on 2D spatial data + one thematic dimension
... on 2D spatial data + two thematic dimensions
Conclusions

- R*- and RSS-trees can be recommended as (medium-dimensional) index types for combined query scenarios.

- Combined scenarios occur often in spatio-temporal-thematic DBS; think of, e.g., "validtimeInteger"!

- For new UDTs/operators, for combined UDTs/operators, and for experimental index selection, a framework for **adaptable indexing in ORDBS** is required.
Conclusions (cont.)

- OraGiST is such an indexing framework:
  - extensible library + toolbox
  - coupling the GiST-family with an ORDBMS
  - for adoption of existing index types
  - for fast development of new index types by object-oriented specialization
Future Questions

- How to improve behaviour on 'complicated' objects?
- How to include user-defined cost/selectivity estimation?
- How to generate and hide the combined operators?