

# CMSC 724: Extensible Systems; GiST

Amol Deshpande

University of Maryland, College Park

March 13, 2007

\*Adapted from Joe Hellerstein's Notes (<http://redbook.cs.berkeley.edu/redbook3/lecs.html>)

# Extensible Systems

- Need to support new data types/operators
- Object-oriented vs Object-relational
- Object-oriented
  - Shrink the impedance mismatch for app programmers
  - Semantically richer data models
  - New language features (e.g. complex programming)
- Object-relational
  - Storage and querying over complex data types
  - Many applications: web servers, text collections, time-series data, GIS, image DB

Query	RDBMS	ORDBMS
No Query	File System	OODBMS
	Simple Data	Complex Data

# Postgres

- INGRES-ADT → Post-INGRES
- Query language: “quel”
- Ability to add new data type. Type defined by:
  - storage size
  - input and output methods (functions to convert to-and-from strings)
  - other methods
- Ability to add new operators
  - input/output types
  - precedence
  - etc. . .
- Methods called via “function pointers”
- Dynamic linking vs static linking
- Trusted code ???

- PosgreSQL

- <http://pgsql.active-venture.com/xtypes.html>
- <http://pgsql.active-venture.com/xoper-example.html>
- <http://pgsql.active-venture.com/xaggr.html>

- Access Methods
  - B+-Trees can be used for any object that supports a set of functions
  - Similarly, hashing can be used if  $H(\text{key})$  is supported
  - Requirements, different data types, selectivities all stored in relations
  - GiST !
- Interactions with transaction management
  - The hardest part for access methods
  - Recall that concurrency/recovery highly customized for B+-Trees
  - Somewhat okay if you only physical logging
  - Otherwise, can expose some of the API

- Query Processing
  - must be able to compute selectivities
  - need to know if we can use sort-merge join, hash join
  - figure out which access methods can be used for executing a query
  - rules to specify these things
  - fairly easy to make dynamic programming (Selinger et al) work with it

- Generalized Search Tree
  - Allows extending data types as well as queries
  - A single data structure that can handle many different index structures
- Key insight:
  - An index structure partitions the input data hierarchically
  - GiST associates a “predicate” with each subtree, that is true for all data items in the subtree
  - Nodes contain between  $kM$  to  $M$  entries where  $2/M \leq k \leq 1/2$  (except root)
  - Leaf nodes:  $(p, ptr)$ 
    - $ptr$ : pointer to actual record
    - $p$ : predicate satisfied by the record
  - Non-leaf nodes:  $(p, ptr)$ 
    - $ptr$ : pointer to another node
    - $p$ : predicate satisfied by all records in the subtree below

- Need to define 6 functions for a new tree
  - Consistent( $E, q$ ): given a  $E = (ptr, p)$ , might  $q$  be satisfied by some tuple in the tree below  $ptr$
  - Union: Find new keys
  - Compress, Decompress: used for compressing the keys
    - Required to implement common optimizations
  - Penalty, PickSplit: Used for deciding where to insert a new object, and how to split a page if needed
    - Very similar to R-Trees
- Much work later on at Berkeley
  - Different types of trees; Visualization; Concurrency
  - Indexability theory
  - Formalisms for analysis: different types of inefficiencies
  - etc. . .