Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 9 – MM/Search Extensions for Object-Relational DBMS

(OR-)DBMS-Support for Content-based Search

- Search engine coupling
  - separate, external search engine for content-based retrieval
- Integrated search support
  - utilization of "conventional" index support (e.g., b*-trees)
    - "high-level indexing"
  - specialized (multi-dimensional) index support
- Extensible indexing support

Coupling of External Search Engines

- Shallow Integration
  - Loose coupling of DB search and content-specific search/retrieval
    - content search engine is not integrated into DBMS
    - well-defined interfaces and interaction
  - Index data for media objects may still reside outside the DBMS
    - index is accessed during query evaluation by calling out to external search engine
  - Location/storage of media objects is not impacted
    - inside the DB, or outside (e.g., as a file)
- Motivation
  - lack of appropriate index support in the DBMS
    - adding new index support is complex, expensive
  - optimized external search engine exists
    - costly migration
- Potential problems
  - integrity
  - usability (in search and administration)
  - performance

Example: DB2 Text Extender

- Text is stored in char column (long varchar, CLOB, ...) or anything that can be used to "produce" character strings
  - structured type (SQL/MM)
  - external storage
- Search using scalar UDFs
  - CONTAINS
  - SCORE
  - NUMBER_OF_MATCHES
- Remember: preprocessing is expensive!
  - CONTAINS function always needs to access an external text index
    - performance issues
  - updates, inserts on text columns become expensive
    - asynchronous index updates
    - decoupled from column update

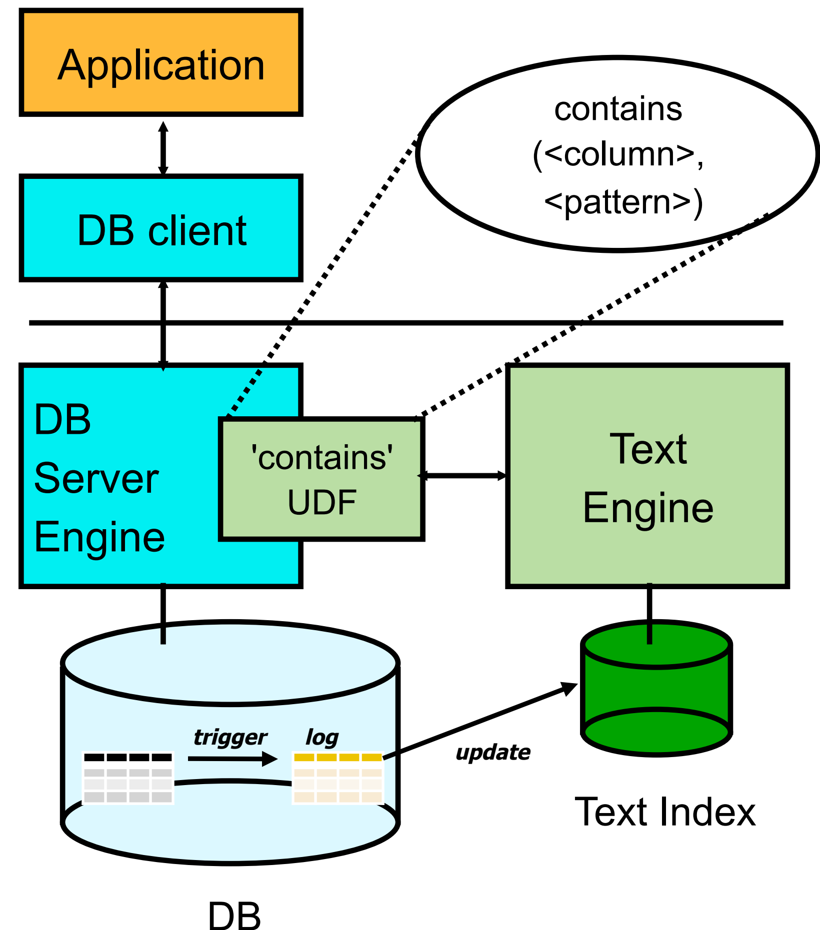
Projects

proj-no	title	budget	description
123	'....'	200000	'This project ..'
456	'....'	400000	'Objects are...'
789	'....'	700000	'A database...'

```
SELECT proj-no, title
FROM compschema.projects
WHERE contains (description,
   ' "database"
     IN SAME SENTENCE AS
   "object-relational" ') = 1
```

Text Extender System Architecture

- Text index is stored outside the DB under control of text search engine (TSE)
  - index scope: all documents in a text column
  - primary key of table used as document id
- Separate TSE process for building and incrementally updating the text index
  - configuration of update interval, etc.
- Each index has a log table in the DB
  - holds information about updates that need to be reflected in the index
  - populated using DB-triggers
  - used by asynchronous index update process
- Search UDF (implemented as a C function) calls text engine, which returns a list of doc-ids
  - may "miss" the latest updates!



© Prof.Dr.-Ing. Stefan Deßloch

Search – Performance Problems

- Scalar UDFs (contains, etc.)
  - calls text search engine to retrieve search result list
  - checks whether the document id is in that list
    - or returns the score, #of matches
  - for each tuple in PROJECTS!
- First (big) improvement:
  - UDF can 'buffer' the result list to reduce text engine API calls
  - TSE API only needs to be invoked for the first tuple in PROJECTS
- But: the DB engine still calls the UDF for every row!
  - table scan
  - invocation overhead

Projects

proj-no	title	budget	description
123	'....'	200000	'This project ..'
456	'....'	400000	'Objects are...'
789	'....'	700000	'A database...'

```
SELECT proj-no, title
FROM compschema.projects
WHERE contains (description,
    ' "database"
      IN SAME SENTENCE AS
    "object-relational" ') = 1
```

Search Using Table Functions

- Table function (TF)
  - (user-defined) function that returns a table structure instead of a single scalar value
  - can be invoked in the FROM clause of a SELECT statement using special syntax
- Text search TF interacts with TSE, returns search results for specific index
  - input parameters for
    - scope of the text search (external index name or name of indexed table, column)
    - text search pattern
    - optional parameters for limiting the result set
  - result table has columns holding
    - primary key value (document id)
    - score (optional), number of matches (optional)
- Example

```
SELECT p.pro-no, p.title
FROM compschema.projects p,
  TABLE(containstable('COMPSCHEMA', 'PROJECTS', 'DESCRIPTION,
        ' "database" IN SAME SENTENCE AS "object-relational" ')) AS restab
WHERE p.proj-no = restab.primarykey
```

Table Functions - Evaluation

- Advantages
  - performance!
    - avoids table scan
- Disadvantages: usability!
  - asks end-user to make choice for the optimizer
    - in some situations, using the scalar function results in a better plan
  - lack of transparency
    - two different syntax alternatives for the same query
  - use of table functions
    - not "intuitive" to write
    - requires join of function result with document table, complicating the quers
    - potential lack of support by query tools, data access tools
  - view transparency lost
    - view definition may access multiple tables with text columns
      - involves multiple indexes, based on the base table columns
    - user has to know view definitions

Query Rewrite/Optimization Support

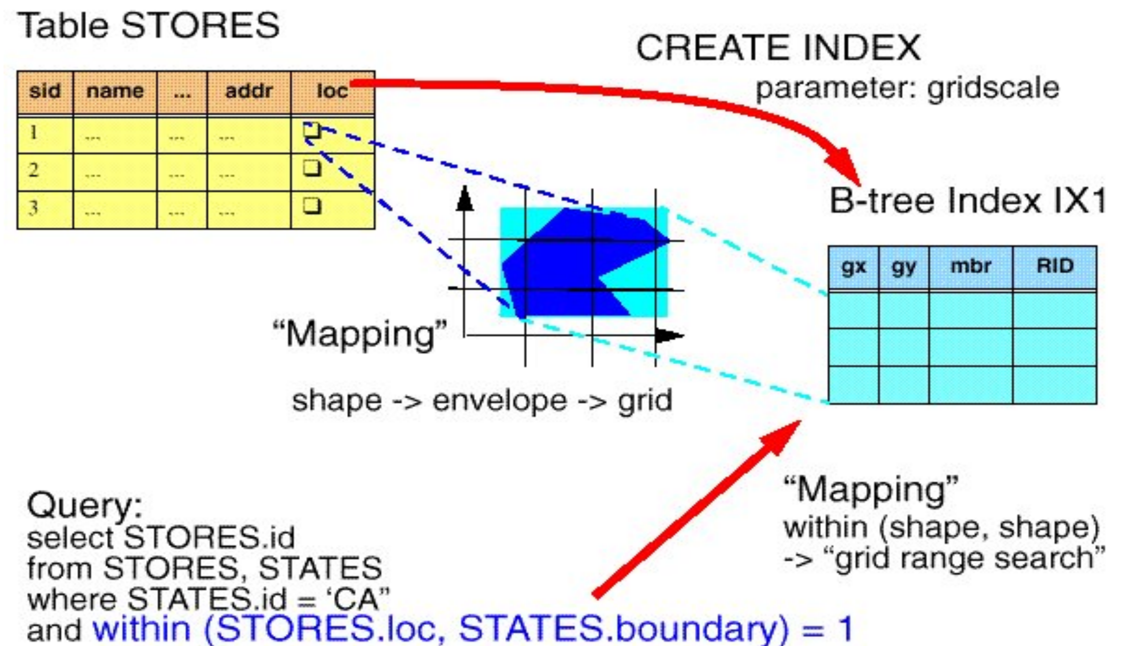
- Optimizer is made aware of additional rewrite options
  - users continue to use scalar functions for better usability
  - internally, the query is rewritten to exploit table function for better performance
- Based on correspondences
  - scalar function to table function
  - parameter correspondences
    - search argument
    - document/primary key columns
  - meta data
    - table/columns names as TF parameters
  - either hard-wired, or through syntax extensions in "CREATE FUNCTION"
- Further optimization opportunities
  - multiple scalar functions in the same query mapped to the same TF
  - predicate/sorting "push-down"

```
SELECT proj-no, title
FROM compschema.projects
WHERE contains (description,
' "database" IN SAME SENTENCE AS
"object-relational" ') = 1
```

```
SELECT p.pro-no, p.title
FROM compschema.projects p,
        TABLE(containstable(
        'COMPSCHEMA', 'PROJECTS',
        'DESCRIPTION',
        ' "database" IN SAME SENTENCE AS
          "object-relational" ')) AS restab
WHERE p.proj-no = restab.primarykey
```

High-Level Indexing - Motivation

- Existing DB index mechanisms (e.g., b*-tree) may not support content search predicates directly (e.g., within (shape, shape), overlaps (shape, shape) for spatial)
- But it may be possible to exploit them to a certain degree
- Example: spatial search
  - define a coordinate grid
  - b*-tree index entries for shape
    - grid cell coordinates
    - min. bound. rectangle (mbr)
  - for each cell touched by mbr
- Search can be done in stages
  - compute grid cells, mbr for search argument ('CA')
  - search index with cell coordinates as arguments
  - filter false positives based on mbr, eliminate duplicates
  - compute final result using exact shape



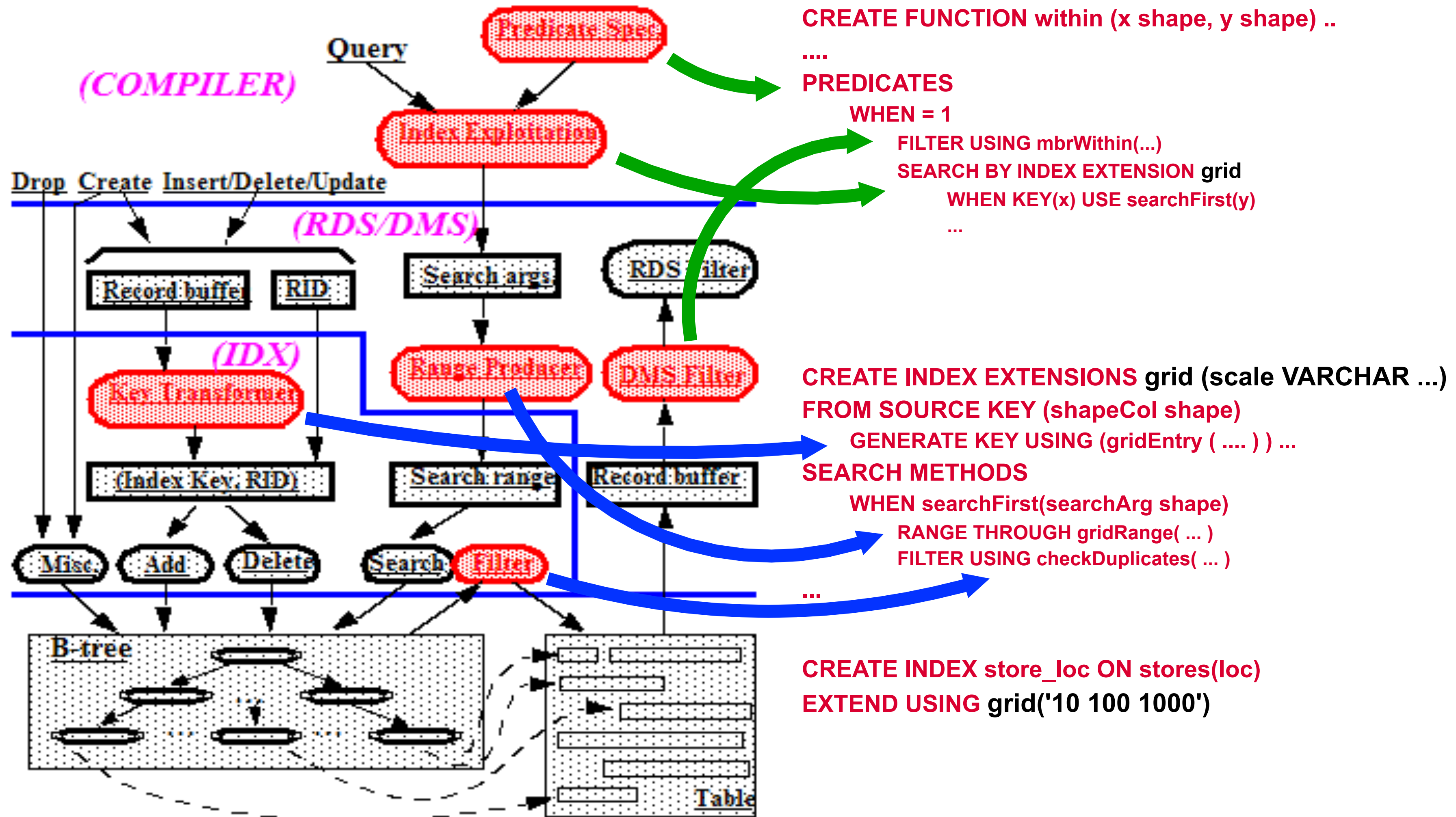
Spatial Indexing Requirements

- Index type for 'reuse' in index creation
- Index entries
  - Not useful to store complete shape object in the index
  - Should contain information extracted from a shape object
    - grid cell coordinates
    - minimum bounding rectangle information
  - Multiple index entries for a single shape object have to be supported
    - a shape object may 'appear' in multiple grid cells
- Index exploitation (search)
  - flexible search method for mapping a 'query shape' to a range search on the index
  - multiple levels of search (result set filtering)
    - grid coordinate match
    - mbr overlap or containment
    - full geometric overlap or containment
  - multiple search methods for the same index extension
    - overlap or containment
- Index parameters
  - grid levels (determines granularity of the grid)
  - grid levels may vary for individual indexes

Index Extension Support

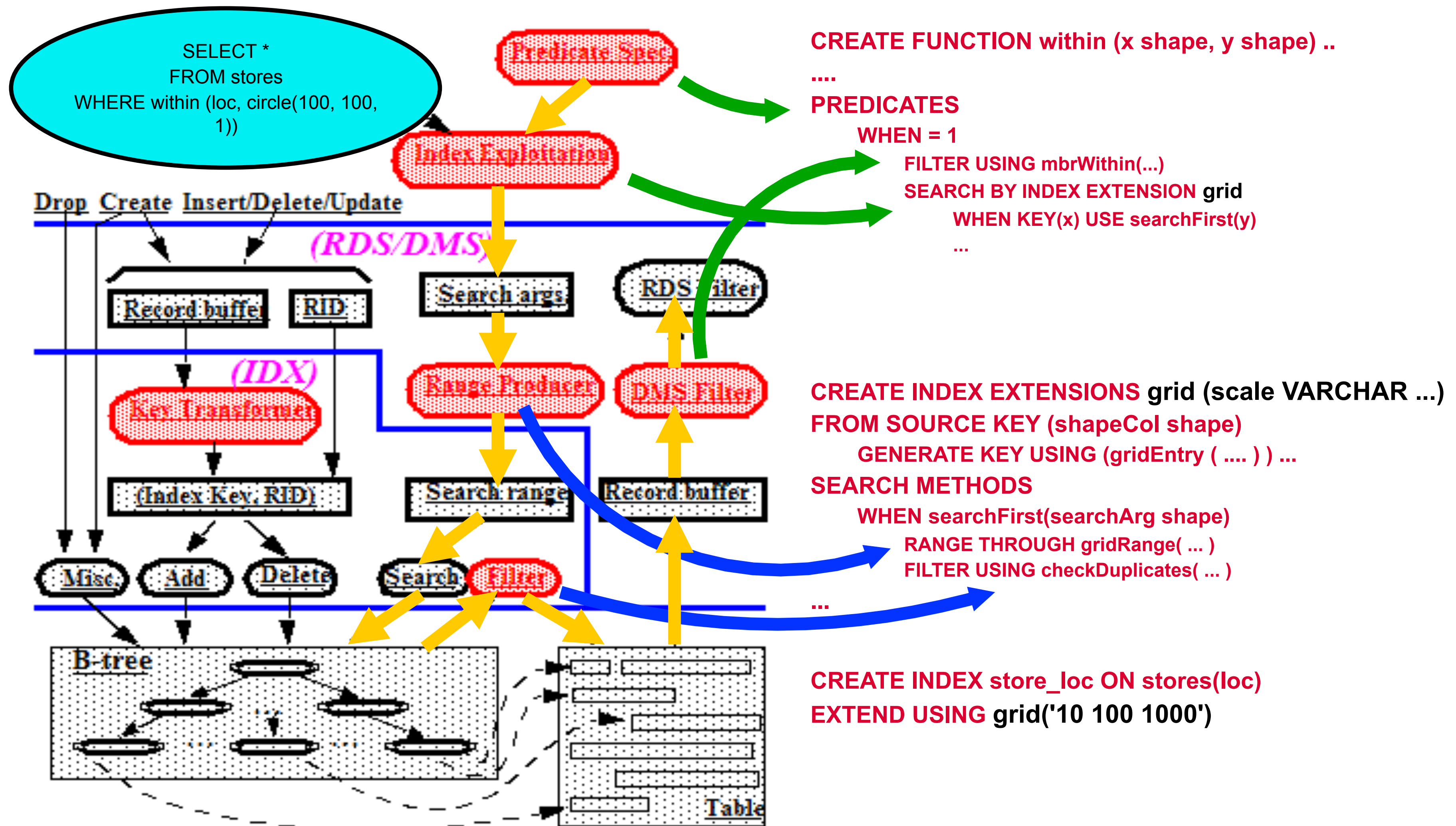
- Builds on top of existing B-Tree support
  - index plug-ins
- DDL for creating named index extensions
  - define parameters to be supplied at ‘create index’ time
  - specify mapping of UDT to (multiple) index entries (Key Transformer)
  - define search methods that map a 'query literal' to a set of ranges over the index (Range Producers)
  - provide filter functionality that further reduces answer set during index lookup (IDX Filter)
- Specify how search UDFs can be mapped to search methods of the index extension
  - extended CREATE FUNCTION syntax to provide Predicate Specification, Index Exploitation
  - provide filter functionality that further reduces answer set during DMS predicate evaluation (DMS Filter)
- Extended ‘CREATE INDEX’ to allow usage of index extensions
  - create index using an index extension
  - supply required parameters (e.g., grid scale)

Index Extension Architecture - DDL



© Prof.Dr.-Ing. Stefan Deßloch

Index Extension Architecture - Query

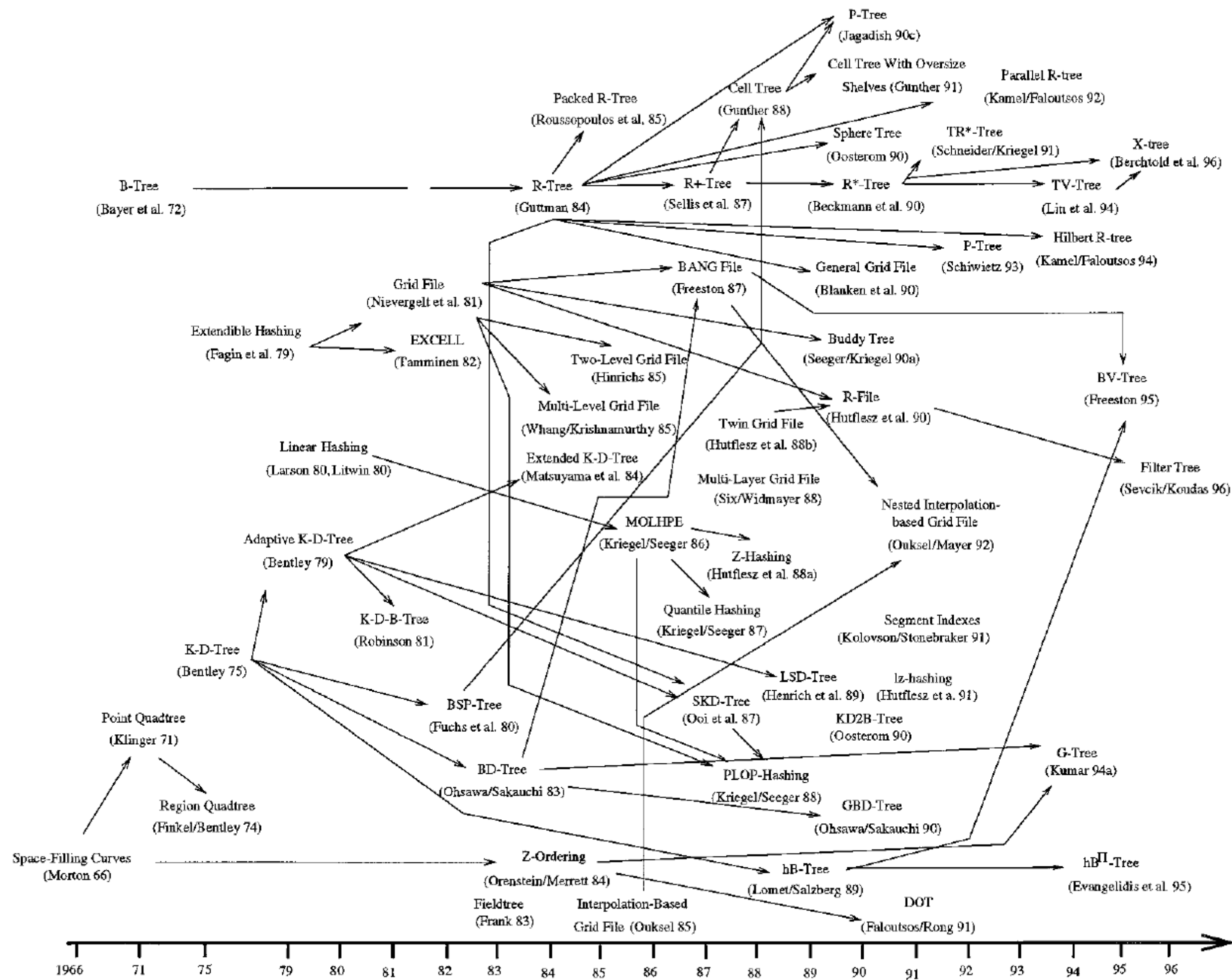


Specialized Index Support

- Most interesting queries over media objects are
  - range queries, or
  - nearest-neighbor queries (top-k) involving similarity/distance measure
  - involving multiple dimensions
- "Classic" index structures in DBMS (e.g., B-tree)
  - limited to a single dimension
  - can be leveraged only in a restricted manned, not suitable for high-dimensional space
- Multi-dimensional index methods
  - large number of methods proposed over the last years
  - no clear winner
    - complexity (hard to understand/compare)
    - numerous criteria for optimality, performance
    - strong dependency on data/query
  - commercial systems
    - optimized, highly tuned implementation of a simple and robust index method
    - most popular: R-tree
- More details: course on realization of database systems

Multi-dimensional Access Methods - History

Gaede, V., Günther, O.:
Multidimensional Access Methods
ACM Computing Survey 30:2, 1998



Extensibility – User-defined Access Methods

- ORDBMS provides support for user-defined access methods
  - primary access methods
    - relational table interface for direct read/write access
    - data may be stored outside the DB
  - secondary access methods
    - index structure to support key-based retrieval of rows in a table
    - index entries may reside outside the DB
- Based on generic interfaces
  - developers can supply their own implementation of access method APIs
  - implementation may utilize storage services of the DBMS (e.g., BLOB storage)
- Example: IBM Informix Dynamic Server virtual tables/indexes
  - pioneered in POSTGRES DBMS
  - Oracle (Data Options) offers similar capabilities

Virtual Indexes in Informix Dynamic Server

- Virtual Index Interface
  - purpose functions
    - functionality to build, connect to, populate, query, and update the index
      - includes cost information for the optimizer
    - called by DBMS server to pass SQL statement specifications to the access method
    - example: CREATE INDEX …
    - to be implemented by the access method developer
  - descriptors
    - predefined data types used to exchange information
      - e.g., qualification descriptor contains a data structure describing the content of the WHERE-clause
    - parameters for API calls
  - accessor functions
    - obtain specific information from the descriptors
    - supplied by the DBMS
- Programmer is responsible for implementing
  - index functionality (see above)
  - concurrency control on index
  - logging/recovery, unless index data resides in DB BLOBs

Important Purpose Functions - Overview

Invoking Statement	Purpose Function
all	am_open(MI_AM_TABLE_DESC *) am_close(MI_AM_TABLE_DESC *)
CREATE INDEX	am_create(MI_AM_TABLE_DESC *) am_insert(MI_AM_TABLE_DESC *, MI_ROW *, MI_AM_ROWID_DESC *)
DROP INDEX	am_drop(MI_AM_TABLE_DESC *)
INSERT	am_insert(MI_AM_TABLE_DESC *, MI_ROW *, MI_AM_ROWID_DESC *)
DELETE	am_delete(MI_AM_TABLE_DESC *, MI_ROW *, MI_AM_ROWID_DESC *)
SELECT INSERT, UPDATE, DELETE WHERE...	am_scancost(MI_AM_TABLE_DESC *, MI_AM_QUAL_DESC *) am_beginscan(MI_AM_SCAN_DESC *) am_getnext(MI_AM_SCAN_DESC *, MI_ROW **, MI_AM_ROWID_DESC *) am_endscan(MI_AM_SCAN_DESC *)
SELECT with join	am_rescan(MI_AM_SCAN_DESC *)
UPDATE	am_update(MI_AM_TABLE_DESC *, MI_ROW *, MI_AM_ROWID_DESC *, MI_ROW *,MI_AM_ROWID_DESC *)
UPDATE STATISTICS	am_stats(MI_AM_TABLE_DESC *,MI_AM_ISTATS_DESC *)

Operator Classes

- Operator class connects SQL operators, predicates, data types to an access method
  - which data types can be indexed using a specific secondary access method?
  - what predicates can be supported by the index?
  - how can the optimizer be provided with statistics?
- Two types of functions
  - strategy functions
    - needed for optimizer to decide whether an index can be used for a specific operation on a data type
    - lists operators that appear in SQL (e.g., "=", "contains", …) and are supported by the index
  - support functions
    - called by the access method, e.g., to traverse the index and obtain the results
    - example: "compare keys" for a B-tree index
- Similar to high-level indexing for B-trees, but supports user-defined access methods as well!

Extensibility – Generalized Search Trees

- Generalized Search Tree (GiST)
  - generalization of tree-based index structures
    - e.g., B*-tree, R-tree can be seen as special cases
  - framework
    - provides implementation of common, generic index functionality
    - adapted by providing/registering a *key class* implementation with six methods
- Common GiST properties
  - balanced tree, high fanout
  - internal nodes are used as a directory
    - series of keys, pointers
  - leaf nodes point to the actual data
    - linked list storage
  - search for tuples that match a query predicate:
    - starting at the root, for each pointer on the node,
      if the associated key is consistent with the query predicate,
      then traverse the subtree
    - requirement: key must match every data item (transitively) stored "below" it

may be an arbitrary 'predicate' that holds for each datum below the key, e.g., an integer range, or a bounding box

GiST Key Methods and Tree Methods

- Key Methods (to be provided as an extension)
  - consistent (entry, predicate)
    - false, if conjunction of key and query predicate is unsatisfiable
    - may return false positives
  - union (set of entries)
    - return predicate that holds for the union of all tuples stored 'below' all of the entries
  - penalty (entry1, entry2)
    - domain-specific cost (penalty) for inserting entry2 into entry 1 subtree
    - aids split and insert algorithms
  - picksplit (set of entries)
    - splits set of entries into two sets of entries
  - compress
  - decompress
- Tree methods (provided by framework)
  - search
    - uses "consistent()"
  - search in linear ordered domains (findMin, Next)
    - uses "consistent()"
    - requires further ordering guarantees, "compare" method implementation
  - insert (insert, chooseSubtree, split, adjustKeys)
    - uses "penalty", "pickSplit", "union"
    - maintains tree balance
  - delete
    - uses "union"

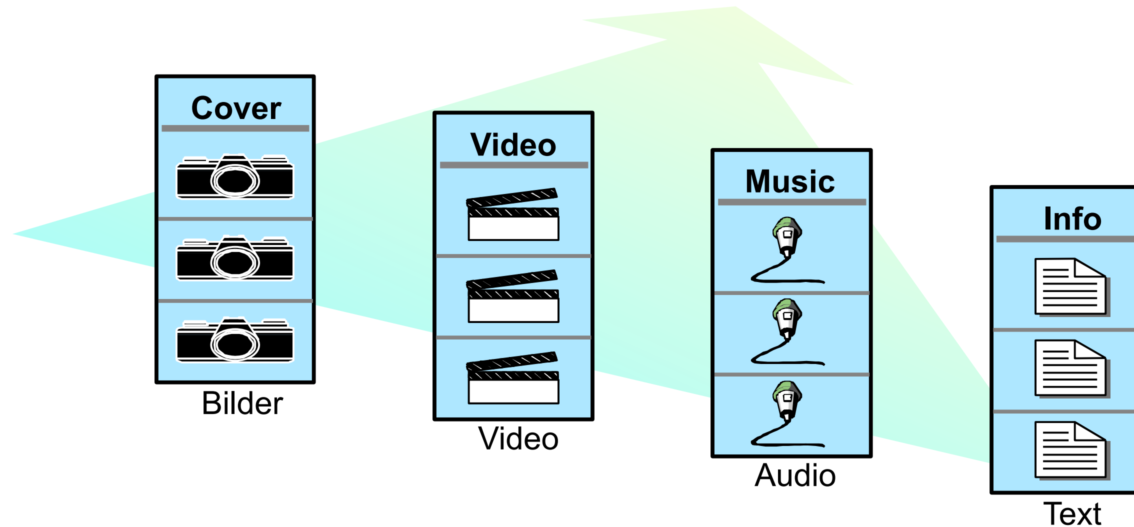
Commercial Systems

- Database Extenders (IBM DB2)
  - data types and functions
    - Text Search, Net Search Extender
    - Image, Audio, Video Extender
    - Spatial Extender (ESRI)
  - utilize search engine coupling, high-level indexing approaches
- Data Blades (Illustra, IBM Informix Universal Server)
  - collection of data types and associated functions
    - Text, Spatial, Geodetic, Image Foundation, Video Foundation, Visual Information Retrieval (Virage)
  - utilize virtual indexes, operator classes, R-tree specialized index structure
  - largely provided by business partners, certified by IBM/Informix
- Data Options (Oracle)
  - interMedia (text, image, audio, video), Spatial, Visual Information Retrieval
  - utilize virtual index approach
- Status
  - similar functionality, but only partial standardization

DB2 Extender – Application View

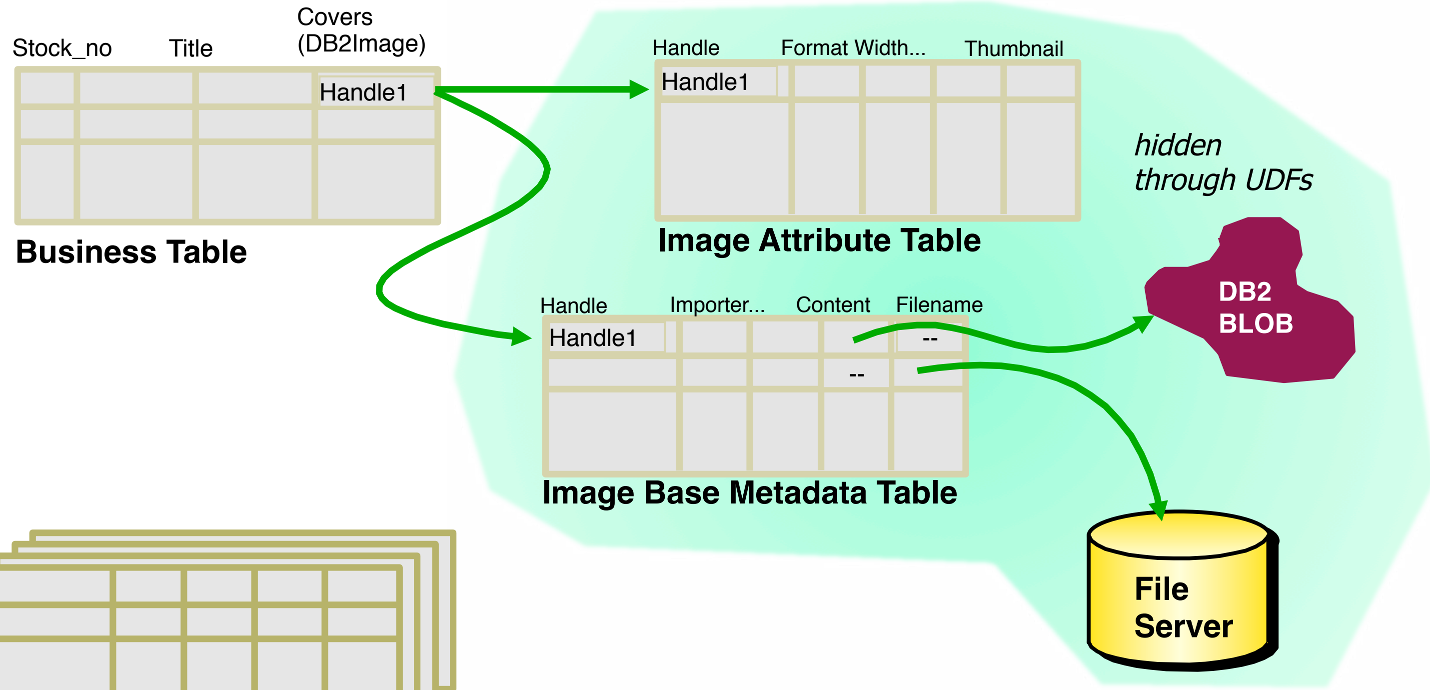
Non-MM-Daten ***Integration of MM-Daten***

Artist	Title	Sold	On-Hand	Rating				
Lizzi	*Decisions*	*165*	*52*	*1*				
Dwayne Miller	*Earthkids*	*76*	*100*	*3*				
Nitecry	*Run for Cover*	*65*	*30*	*7*				



- **Complex data types**
- **New functions**
- **Integrated search**
- **Open architecture**

Extended Data Model (Example: Image Extender)



MMDB Administration Tables (ExtenderInfo, MetaTableNames)

Image Extender: Overview

- Attributes stored in side tables, accessible through UDFs
  - format, thumbnail, length, width, ...
- Support for common image formats
  - (BMP, EPS, EP2, GIF, IMG, IPS, JPG, PCX, PGM, PS, PSC, PS2, TIF, YUG, ...)
- Format conversion routines
- Support for internal and external storage of media objects
- Utilizes the Query by Image Content (QBIC) search engine
  - average color, color histogram, positional color, texture
  - features are extracted in an explicit catalog run, then available for search
- Ranking (scoring) of search results

Video Extender: Overview

- Supported (searchable) attributes:
  - format, duration, number of frames, ...
- Support for AVI, MPEG1, MPEG2, QT
- DB-storage for store-and-forward playback
- External media-server storage for realtime playback
- Import, export and update of videos
- Support for Video Shot Change Detection
  - shot detection, management of a shot catalog in the DB, extraction of frames
- In combination with QBIC:
  - "Find the sunset shot in the video most similar to a given image and start playback at that shot"

Summary

- ORDBMS architectures and support for MM-search
  - cannot be limited to provision of data types and functions
  - requires additional infrastructure for efficient content-based search
- Search engine coupling
  - separate, external search engine for content-based retrieval
    - cost, utilization and protection of competitive, optimized search engines
    - table functions, query rewrite approaches for performance/usability improvements
- Integrated search support
  - utilization of "conventional" index support (e.g., b*-trees)
    - "high-level indexing" that provides mapping of UDTs and predicated to index capabilities
    - multi-level search
  - specialized (multi-dimensional) index support
- Extensible indexing support
  - virtual indexes/access methods
    - need to implement "purpose functions" for index operations, index maintenance
      - rather complex undertaking (locking, recovery, …)
    - most powerful and flexible approach
  - generalized search trees (GiST)
    - reduced programming effort through search tree framework, class library
- Commercial ORDBMS support
  - IBM DB2 Extenders, IBM Informix Data Blades, Oracle Data Options