10. Record-Oriented DB Interface

Theo Härder
www.haerder.de

Goals

- Design principles for record-oriented processing and navigation on logical access paths
- Development of a scan technique and a sort operator

Main reference:
Theo Härder, Erhard Rahm: Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer, 2001, Chapter 10.

Goetz Graefe: Implementing Sorting in Database Systems, ACM Computing Surveys 38:3, Article 10, 37 pages, Sept. 2006.

© 2011 AG DBIS

Logical Access Paths

- **Characterization of the mapping**

```
STORE <record>
FETCH <record> USING <attr1> = 400 AND <attr2> >=7
CONNECT <record> TO <set>
```

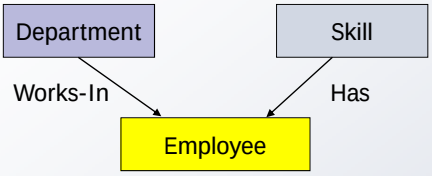
Mapping functions
- Physical record <-> External record
- External record <-> related access paths
- Search expression -> supporting access paths

Insert <record> at ...
Add <entry> to...
Retrieve <address-list> from...
Retrieve <record> with

- **Properties of the upper interface**
  - Logical records (dynamic format conversion)
  - Logical access paths
    (content-addressable storage, hierarchical relationships between record types)
  - One-record-at-a-time access
  - Application programming interface (API) using navigational access
    (e.g. network data model or object-oriented data models)

© 2011 AG DBIS

Network Data Model - An Example



1. **Simple content addressability**
   FETCH Employee USING Eno=12345

2. **... combined with hierarchical access**
   FETCH Department USING Dept-Name ='Sales'
   FETCH NEXT Employee WITHIN Works-In

3. **... with two hierarchical relationships**
   FETCH Skill USING Job = 'Programmer'
   FETCH NEXT Employee WITHIN Has
   FETCH OWNER WITHIN Works-In

© 2011 AG DBIS

Mapping of External Records

- Description of external records by subschema concept
- **Tasks of subschema concept**
  - Adjustment of data types
  - Selective specification of attributes
  - Mapping of an external record to internal records of one/several record types
- **Partitioned storage of large record sets:**
  allocation of disjoint record sets to separate storage units
  - Performance reasons: I/O parallelism
  - Availability: creation of copies, migration, . . .
  - Partition is unit of reorganization, backup, archiving, loading, access methods, etc. in DB2
  - Specification of partitioning via values (key ranges, hashing) or via procedures (user exit)
- **Storage options for internal records**
  - Partition and allocation of fields according to access frequencies
  - Redundant storage
  - Compression of fields and records

➥ **Opportunities for adjustment / improvement of performance (tuning)**

© 2011 AG DBIS

Record-Oriented Processing

- How is **record-oriented processing** supported by the storage structure layer?

  → see chapters 5 – 9

- **Processing concepts**
  - Context-free operations
  - Record-at-a-time navigation via existing access paths
  - Reordering of a record set, if suitable sequence does not exist

- **Processing primitives**
  1. Direct access (hashing, trees, ...)
  2. Navigational access via scan (on which objects?)
  3. Sorting of a record set and scan on it (which record sets?)
  4. Record-at-a-time modifications (Insert, Delete, Update)

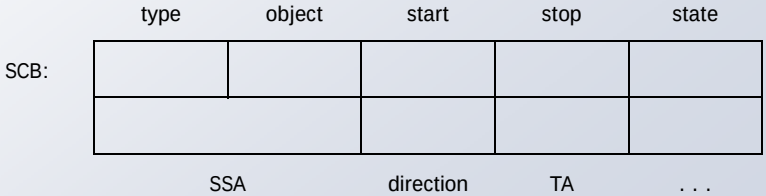
- **Introduction of a navigation concept**
  - Provision and maintenance of transaction-related **processing positions for the navigation**
  - Use of different scan types

© 2011 AG DBIS

Spectrum of Scan Types

- **Scan types**
  - Record-type scan (table scan) to locate all records of a record type (a table)
  - Index scan to locate records in a value-dependent sequence
  - Link scan to locate records in a user-defined insertion sequence
  - k-d scan to locate records via a k-dimensional index*

- **Implementation of scans**
  - Explicit definition/release: OPEN/CLOSE SCAN
  - Navigation: NEXT TUPLE
  - Scans are defined on access paths
  - Options: start-, stop-, and search condition (Simple Search Argument) search direction: NEXT/PRIOR, FIRST/LAST, n-th
  - Scan control block (SCB): information about type, state, position, etc.



* It is desirable but difficult to provide a uniform evaluation model for all multi-dimensional access paths; e.g., via a temporary result structure

© 2011 AG DBIS

Table Scan

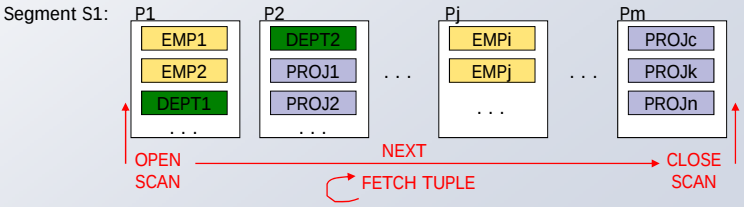
- **Query example:**

  SELECT * FROM Emp
  WHERE Dno BETWEEN ‘K28’ AND ‘K67’ AND Job = ‘Programmer’

- **Scan Options**
  - Start condition (SC): BOS (Begin of S1)
  - Stop condition (STC): EOS (End of S1)
  - Search direction: NEXT
  - Search condition (SSA): Dno ≥ ‘K28’ AND Dno ≤ ‘K67’ AND Job = ‘Programmer’

- **Table scan**

```
OPEN SCAN (Emp, BOS, EOS)                                   /* SCB1 */
WHILE (NOT FINISHED)
DO
        FETCH TUPLE (SCB1, NEXT, Dno ≥ ‘K28’ AND Dno ≤ ‘K67’ AND Job = ‘Programmer’)
        . . .
END
CLOSE SCAN (SCB1)
```



© 2011 AG DBIS

Index Scan

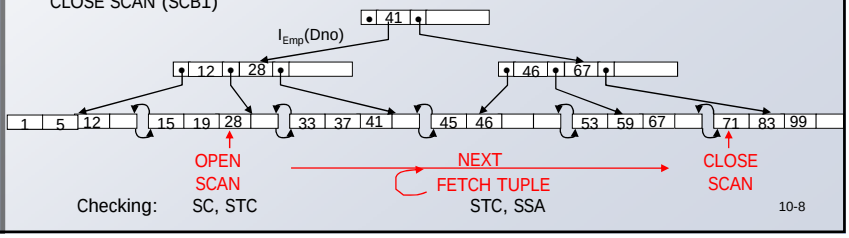
- **Query example:**

  SELECT * FROM Emp
  WHERE Dno BETWEEN ‘K28’ AND ‘K67’ AND Job = ‘Programmer’

- **Scan options**
  - Start condition: Dno ≥ ‘K28’
  - Stop condition: Dno > ‘K67’
  - Search direction: NEXT
  - Search condition: Job = ‘Programmer’

- **Index scan**

```
OPEN SCAN (I_EMP(Dno), Dno ≥ ‘K28’, Dno > ‘K67’)             /* SCB1 */
WHILE (NOT FINISHED)
DO
        FETCH TUPLE (SCB1, NEXT, Job = ‘Programmer’)
        . . .
END
CLOSE SCAN (SCB1)
```



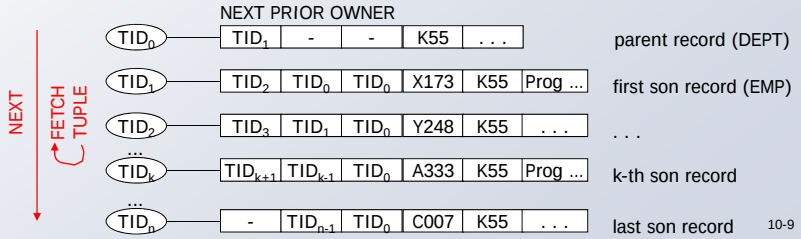
Checking: SC, STC STC, SSA

© 2011 AG DBIS

Link Scan

- **Query example:**
  SELECT * FROM EMP
  WHERE Dno BETWEEN 'K28' AND 'K67' AND Job = 'Programmer'
- **Location of parent**
  - Start condition already found (Dno is given)
  - Scanning in Dept required (Dno BETWEEN K28 AND K67)
- **Single link scan**

```
OPEN SCAN (L_Dept-Emp(Dno), NONE, EOL)          /* SCB1 */
WHILE (NOT FINISHED)
DO
        FETCH TUPLE (SCB1, NEXT, Job = 'Programmer')
        . . .
END
CLOSE SCAN (SCB1)
```



© 2011 AG DBIS

Link Scan (2)

- **Location of parent**
  - Use of an index structure
  - Nesting of index scan (Dno BETWEEN K28 AND K67) and link scan
- **Scan options**

	index scan	link scan
• Start condition:	Dno ≥ 'K28'	--
• Stop condition:	Dno > 'K67'	EOL
• Search direction:	NEXT	NEXT
• Search condition:	--	Job = 'Programmer'

- **Index- and link scan**

```
OPEN SCAN (I_Dept(Dno), Dno ≥ 'K28', Dno > 'K67')      /* SCB1 */
WHILE (NOT FINISHED)
DO
        FETCH TUPLE (SCB1, NEXT, NONE)
        . . .
        OPEN SCAN (L_Dept-Emp(Dno), NONE, EOL)          /* SCB2 */
        . . .
        WHILE (NOT FINISHED)
        DO
                FETCH TUPLE (SCB2, NEXT,
                        Job = 'Programmer')
                . . .
        END
        CLOSE SCAN (SCB2)
END
CLOSE SCAN (SCB1)
```


© 2011 AG DBIS

k-d Scan

- **Query example:**

```
SELECT   *  FROM  EMP
WHERE       Dno BETWEEN 'K28' AND 'K67' AND Age BETWEEN 20 AND 30
   AND      Job = 'Programmer'
```

- **Scan options**

	Dimension 1	Dimension 2
• Start condition:	Dno ≥ 'K28'	Age ≥ 20
• Stop condition:	Dno > 'K67'	Age > 30
• Search direction:	NEXT	NEXT
• Search condition:	Job = 'Programmer' (is evaluated on the EMP records)	

- **2-d Scan**

```
OPEN SCAN (I_Emp(Dno, Age), Dno ≥ 'K28' AND Age ≥ 20, Dno > 'K67' AND Age >30)   /* SCB1 */
. . .
WHILE (NOT (Age > 30))
DO
   /* saving of SCB1 position in SCANPOS                                          */
   WHILE (NOT (Dno > 'K67'))
   DO
      FETCH TUPLE (SCB1, NEXT(Dno),
                   Job = 'Programmer')
         . . .
   END
   /* reset of SCB1 position to SCANPOS                                          */
      . . .
   MOVE SCB1 TO NEXT(Age)
END
CLOSE SCAN (SCB1)
```

© 2011 AG DBIS

Application of Scans

- **Problem**

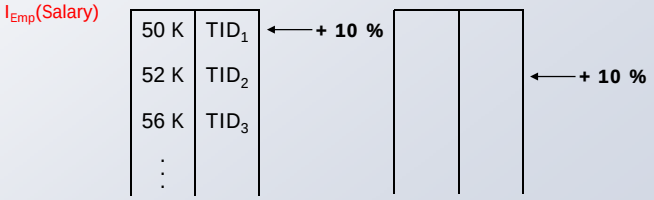
  set-oriented specification <---> record-oriented evaluation:

  Navigational evaluation of a declarative SQL statement via a scan can cause processing problems, especially if the object to be updated (table, access paths) is used by the scan (prohibition?)

- **Example**

  statement: **UPDATE Emp**
  SET Salary = 1.1 * Salary

  execution:
  salary increase by 10% is performed by means of a scan via index Salary



- **Halloween problem**

© 2011 AG DBIS

Use of a Sort Operator

- Explicit reordering of records according to given search key (ORDER clause)
- Reordering and restriction

```
SELECT * FROM  Emp
WHERE Dno > 'K50'
ORDER BY Salary DESC
```

- Partitioning of record sets

```
SELECT Dno, AVG (Salary)
FROM Emp
GROUP BY Dno
```

- Duplicate elimination in a record set

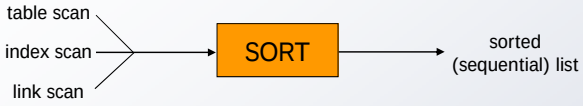
```
SELECT DISTINCT Job
FROM Emp
WHERE Dno > 'K50' AND Dno < 'K56'
```

- Support of set- and join operations
- Reordering of pointers to optimize evaluation or access sequence
- Dynamic creation of index structures ("bottom-up" construction of B*-trees)
- Creation of clustering upon loading and during reorganization

➥ **Reduction of complexity** of $O(N^2)$ to $O(N \log N)$ for set- and join operations

© 2011 AG DBIS

SORT Operator - Options and Application



- **Scans can be restricted by search conditions** (SSAs)
- **SORT options for duplicate elimination:**

  N = no elimination
  K = duplicate elimination w.r.t. sort criterion
  S = STOP as soon as duplicate is detected

- **SORT**
  - serves as base operator
  - for operations at higher levels

Example: use of scan- and sort operator

```
OPEN SCAN (R1, SC1, STC1)                    /* SCB1 */
SORT       R1 INTO S1 USING SCAN (SCB1)
CLOSE SCAN (SCB1)
OPEN SCAN (R2, SC2, STC2)                    /* SCB2 */
SORT       R2 INTO S2 USING SCAN (SCB2)
CLOSE SCAN (SCB2)
OPEN SCAN (S1, BOS, EOS)              /* SCB3, sorted */
OPEN SCAN (S2, BOS, EOS)              /* SCB4, sorted */
WHILE (NOT FINSHED)
DO
           FETCH TUPLE (SCB3, NEXT, NONE)
           FETCH TUPLE (SCB4, NEXT, NONE)
           . . .
END
```


© 2011 AG DBIS

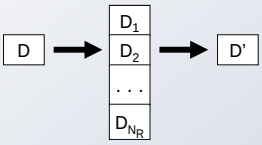
External Sorting

How is to be sorted?

- In general, data set (n records) to be sorted is much larger than available memory for a sort run (q records) (DB buffer or special working memory)
- Creation of so-called runs ($N_R \geq 1$) with an internal sort method
- Which sort algorithms are appropriate?

Repeated execution of sort operator

- Decomposition of input into several runs
- Sorting and buffering the sorted runs in files
- Reading of input data from file D and writing of runs to D_i



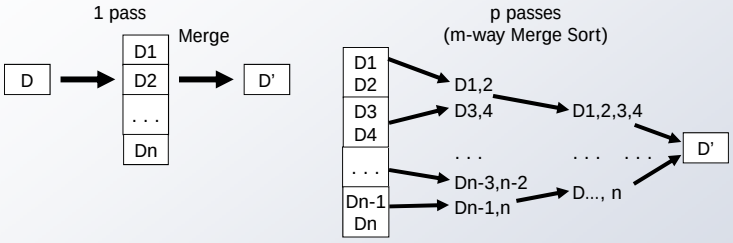
merging of all N_R initial runs required

DBIS Datenbanken und Informationssysteme
© 2011 AG DBIS

External Sorting (2)

DBS: External Sorting

- Successive merging until one sorted run is created



m-way merging with disks

- m_{max} + 1 places (pages) are available in memory. m_{max} determines the maximal merge order.
- In general, N_R initial runs are distributed across several disks
- Each run is sequentially written, if possible. It can be randomly read at any time
- Typically, the initial runs are of uniform length

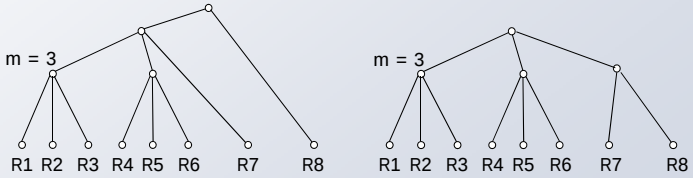
DBIS Datenbanken und Informationssysteme
© 2011 AG DBIS

External Sorting (3)

- **Optimal merge trees**
  - Optimization of seeks (access arm motions) on disks is very difficult: it is usually not considered
  - Goal: minimization of I/O and comparisons
  - Simple in case of ideal numbers $N_R = (m_{max})^p$

    The result is a complete merge tree of height p and degree m_{max}
  - When are **incomplete merge trees** optimal?

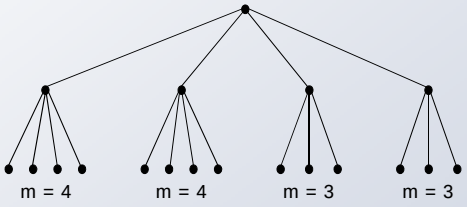
    $N_R = 8$, $R_L = 10$, $m_{max} = 3$



© 2011 AG DBIS

External Sorting (4)

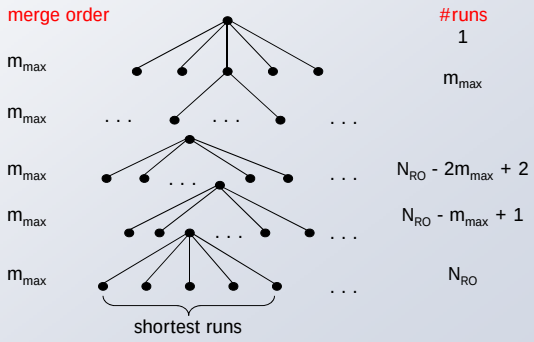
- **Generally holds:** $N_R \leq (m_{max})^{p_{min}}$
- **Assumptions:**
  Only 2 files for input/output, unweighted runs
- **Heuristic 1: Harmonize merge tree**
  (1) Determine p_{min}
  (2) Find smallest m such that holds:
  $N_R \leq m^{p_{min}}$
  (3) Apply only merge orders of m and m-1.
- **Example: $N_R = 14$, $m_{max} = 8$, $R_L = 10$**



© 2011 AG DBIS

External Sorting (5)

- **Assumptions:** no restriction for input/output, weighted runs
- **Heuristic 2:**
  Minimize number of in-/output and comparisons
  (1) $N_R \leq (m_{max} - 1)*p + 1$; determine minimal p
  (2) Create additional empty runs such that holds: $N_{R0} = (m_{max} - 1)*p + 1$
  (3) In each pass, choose the m_{max} shortest runs and use them in a new run (pass), until a single run remains.
- **Schema:**



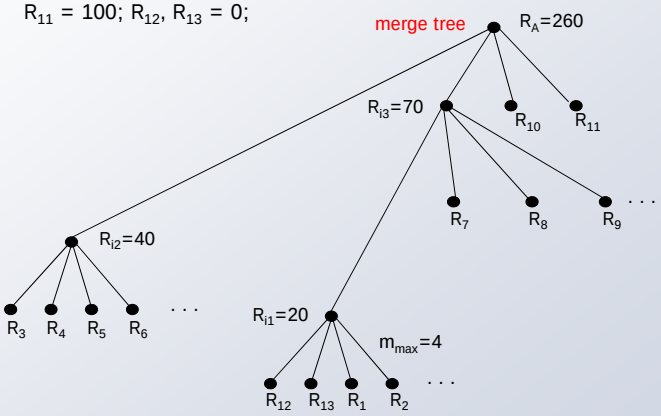
© 2011 AG DBIS

External Sorting (6)

- **Assumptions:** no restriction for input/output, weighted runs
- **Example for Heuristic 2:**

$N_R = 11$, $m_{max} = 4$; → $p = 4$, $N_{R0} = 13$;

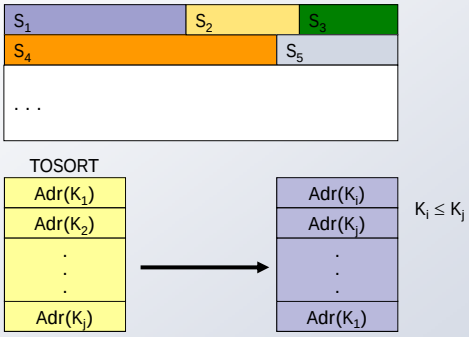
lengths of:
R_1, ... $R_7 = 10$; R_8, $R_9 = 20$; $R_{10} = 50$
$R_{11} = 100$; R_{12}, $R_{13} = 0$;



© 2011 AG DBIS

Sorting of Variable-Length Records

- **Data records in file are often of variable length**
  - Consisting of fixed (f) and variable-length (v) fields
  - As a consequence, the sort keys are of variable length
- **How should the sort area be organized in memory?**



→ When are the data records physically moved into the sort sequence?

© 2011 AG DBIS

Sorting of Variable-Length Records (2)

- **Further problems**
  - Output format deviates from input format

	f	v	v	f	v
input:	A1	A2	A3	A4	A5

	v	f	v	v
output:	A2	A1	A5	A3

  - Sort key consists of several fields which control the sort order: ascending (+) and/or descending (-) e.g. sort key: A5 +, A3 +, A1 –
- **Sort sequence for variable-length keys**

Ascending: `0´	Descending: AAA
A	AA
AA	A
AAA	`0´

- **How do we compare two composed keys of variable length?**

K be defined by A5+|A3+:

What is the comparison result of K1 and K2?

K1 = 12345|67

K2 = 123|6789

© 2011 AG DBIS

Sorting of Variable-Length Records (3)

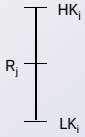
- **How can we exploit a degree of presortedness**
  (use of a Peerage technique)

  Run R_j owns HK_j (Highkey) and LK_j (Lowkey)

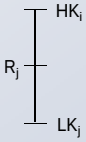
  1. Comparison of R_j with the newly created run R_i

  2. If possible, prolongation of R_j by R_i

a) $HK_i \le LK_j$:



b) $LK_i \ge HK_i$:



  3. Otherwise: separate storage of R_i

© 2011 AG DBIS

Summary

- **Purpose of the layer:**
  - Conceptual separation of internal and external records
  - Top-most layer at runtime
- **Transaction-related control- and surveillance tasks**
  - They need a layer-crossing information flow
  - Load control and -balancing is complex research topic
- **Mapping of external records**
  - Options for the record storage
  - Separation of internal and external records and flexible mapping concepts required
- **Scan technique**
  - Scan technique for record-at-a-time navigation on access paths
  - Flexible use by start-, stop-, and search conditions as well as search direction
- **Sort component**
  - Important for the implementation of relational operations
  - Large tables require sort-/merge methods

© 2011 AG DBIS