7. Hash-based Access Paths

Theo Härder
www.haerder.de

Goals and Restrictions

- Use of key transformation as design principle for access paths to the records of a table, for which a search criterion is supported
- Principles to make hashing dynamic
- Limitation to key access, no range search, etc.

Main reference:

Theo Härder, Erhard Rahm: Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer, 2001, Chapter 7.

Jim Gray, Andreas Reuter: Transaction Processing – Concepts and Techniques, 5th printing, Morgan Kaufmann Publ., 1993, Chapter 15.

Hash-based Access Paths

- **Faster key access requires hashing methods**
  - Hashing methods on external storage
    - Static methods
    - Dynamic Hashing
  - (Only) direct access
  - Ideally a single page access
- **Extendible Hashing**
  - Combination of concepts concerning digital trees and B-trees
  - Extendible Hashing supports strongly growing data volumes ($\leq$ 2 page accesses needed)
- **External Hashing without overflow areas**
- **Linear Hashing**
- **Important parameters:**
  - n = #records of a record type
  - b = #records/bucket (capacity)
  - N = #buckets
  - β = occupancy factor

Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Scattered Storage Structures (Hashing Methods)

- **Direct computation** of record address via key (key transformation)
- Hashing function

h: S → {0, 1, ..., N-1} S = key space
N = size of the static hashing area in pages (***buckets***)

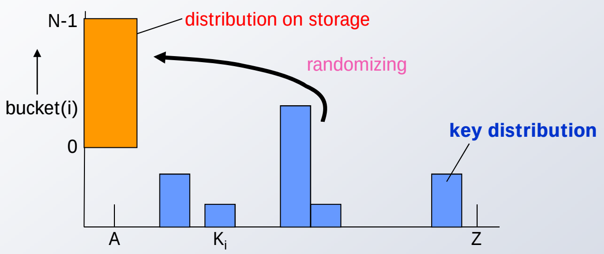
- **Ideal case: h is injective (no collisions)**
  - Application only in exceptional cases possible ('dense' key set required)
  - Each record can be located with a single page access

Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Scattered Storage Structures (2)

- **Static hashing areas with collision handling**
  - Existing set of keys K (K ⊆ S) should be distributed as uniform as possible onto N buckets



  - Handling of synonyms
    - Insertion in the same bucket if possible
    - Allocation and chaining of overflow pages, if necessary
  - Typical access factor: 1.1 to 1.4

- **Multiplicity of hashing functions applicable**
  e.g. division-remainder method, folding, coding method, ...

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

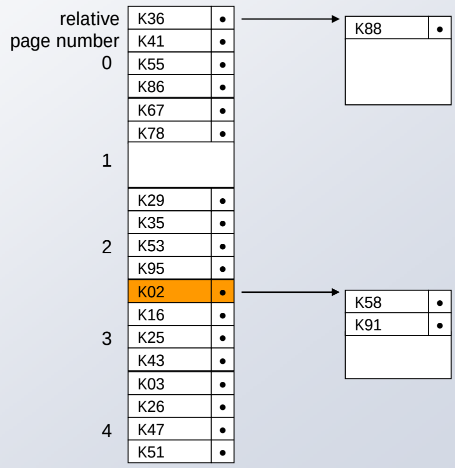
Static Hashing Method with Overflow Areas : Example

- **Address computation for key K02:**

1101 0010
⊕ 1111 0000
⊕ 1111 0010

1101 0000 = 208_{10}

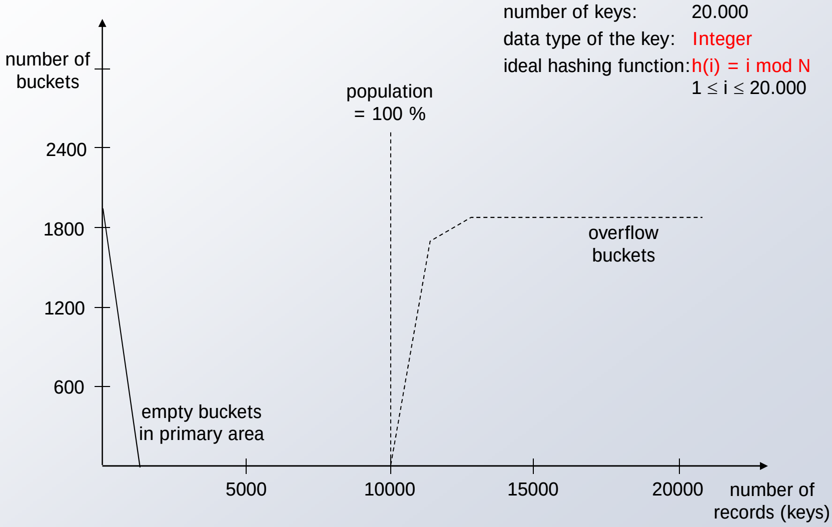
208 mod 5 = 3



Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Occupancy of Hash Areas – Measurement (1)

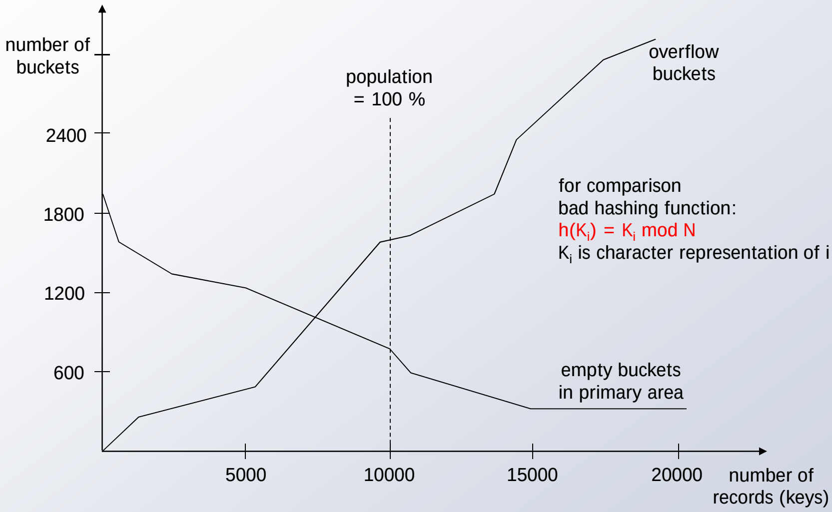
primary area:	2000 buckets
occupancy factor b:	5
number of keys:	20.000
data type of the key:	Integer
ideal hashing function:	h(i) = i mod N $1 \le i \le 20.000$



Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Occupancy of Hash Areas – Measurement (2)



Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Analysis of the Hashing Function

- **Collision if**

  $K_i \bmod N = K_j \bmod N = (K_i + l \cdot N) \bmod N; \quad l = 1, 2, 3, \ldots$

  key allocation is assumed to be $K_i = K_i + j \cdot \Delta k; \quad j = 1, 2, 3, \ldots$

  → critical relationship: $j \cdot \Delta k = l \cdot N$

- **Which distance $j \cdot \Delta k$ causes a collision?**

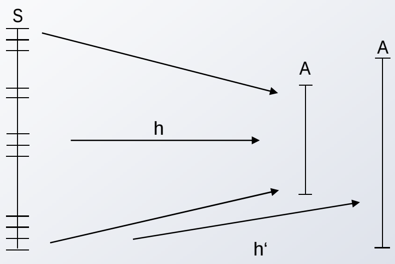
Example: $N = 576$, $\Delta k = 256$

$j = (l \cdot N) / \Delta k$

Dynamic Hashing Methods

- **Growth problem for static methods**
  - Static allocation of storage areas: storage occupancy?
  - In case of address space expansion: Rehashing

  ➥ Cost, availability, addressability



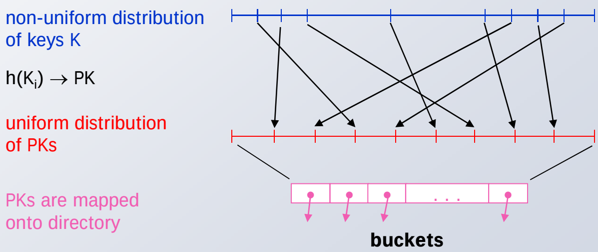
➥ All records obtain a **new address**

- **Design goals**
  - Dynamic structure enables growth and shrinkage of the hash area (file)
  - No overflow techniques
  - Access factor ≤ 2 for direct search

Extendible Hashing

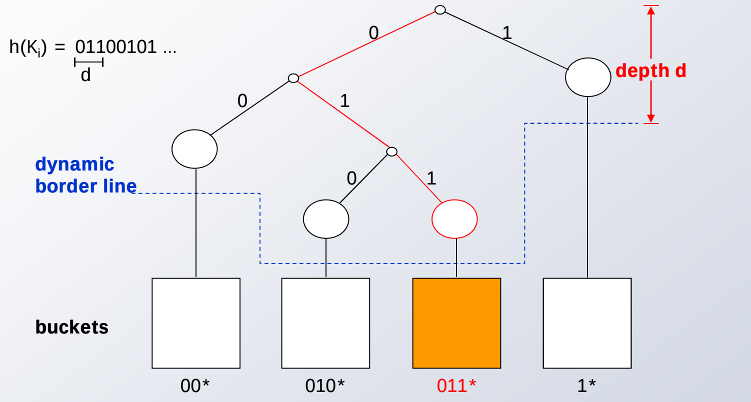
- **Principal approach**
  - The single bits of a keys govern the path through the digital tree used for addressing
  - $K_i = (b_0, b_1, b_2, \ldots)$. In principle, it is possible to directly use the bit sequence of K_i for addressing. Non-uniform key distribution produces an unbalanced digital tree
  - Because digital trees have no balancing mechanism for the height, balance must be enforced from "outside"
  - $h(K_i) = (b_0, b_1, b_2, \ldots)$. The use of $h(K_i)$ as so-called pseudo key (PK) should guarantee better uniform distribution

- **Uniform distribution of PKs**
  implies minimal height of the digital tree



Extendible Hashing* (2)

Principal mapping of the pseudo keys



- d bits are required for bucket addressing which results in a dynamic border line of varying depth, in general
- Digital tree addressing stops as soon as a bucket can accommodate the entire subtree

➥ Balanced digital tree guarantees minimal d_{max}

*Fagin, R., et. al: Extendible hashing – a fast access method for dynamic files. ACM Trans. Database Syst. 4:3. 1979. 315-344

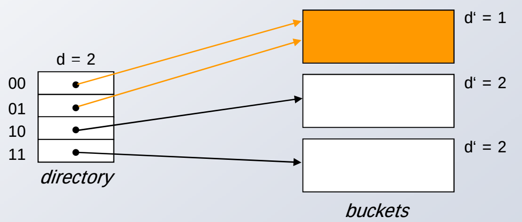
Extendible Hashing (3)

Method does not need overflow areas, but access occurs via *directory* (index)

- Binary digital tree of height d is implemented by a (2^d) digital tree of height 1 (Trie of height 1 with 2^d entries)
- d is determined by the longest path in the binary digital tree
- In a bucket, only records are stored whose PK match in the first d' bits (d' = local depth)
- d = MAX (d'): d bits of PK are used for addressing (d = global depth)
- Directory contains 2^d entries

Storage structure

The Trie can be considered as directory or addressing table. The d bits of $h(K_i)$ refer in the directory to an entry containing the address of the bucket which carries key K_i. If d' < d, (adjacent) entries can refer to the same bucket.

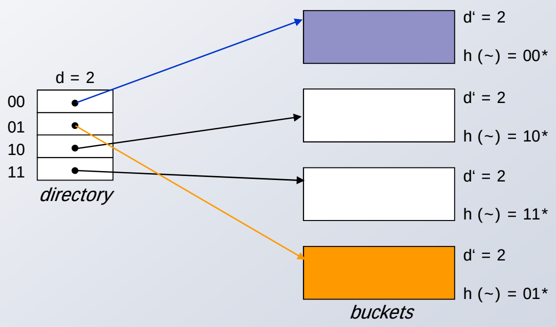


➥ Cost of direct search: **max. 2 page accesses**

Extendible Hashing: Splitting of Buckets (1)

- **Case 1: overflow of a bucket whose local depth is smaller than the global depth d**

➥ Allocation of a new bucket (Split) with
- Local redistribution of data
- Increase of local depth
- Local adjustment of the references in the directory



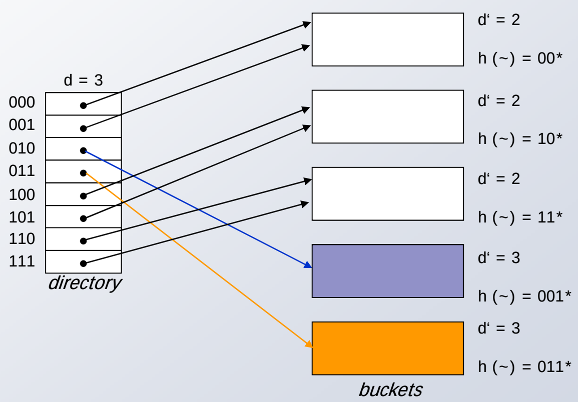
Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Extendible Hashing: Splitting of Buckets (2)

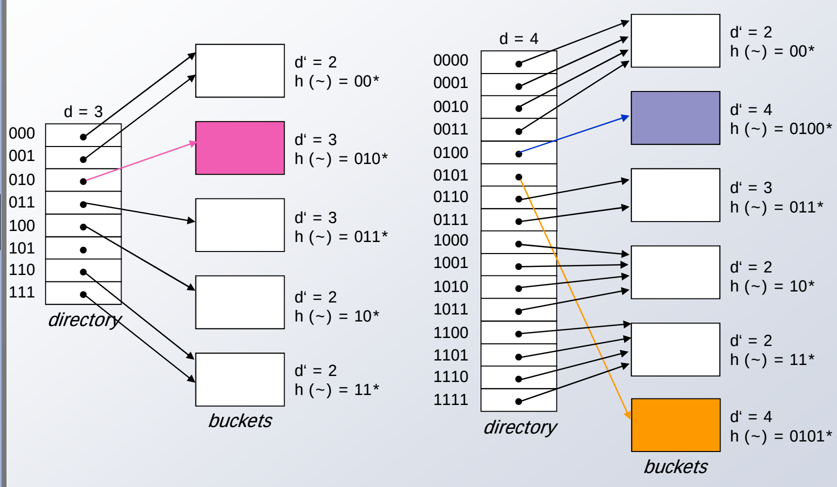
- **Case 2: overflow of a bucket whose local depth is equal to the global depth**

➥ Allocation of a new bucket (Split) with
- Local redistribution of the data (increase of local depth)
- Doubling of the directory (increase of global depth)
- Global adjustment/redistribution of the references in the directory



Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Extendible Hashing: Splitting of Buckets (3)



Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Extendible Hashing (4)

- **Dynamic growth and shrinkage of the hashing area**
  - Buckets are only allocated on demand
  - Nodes in differing depth refer to a bucket

  ➥ **High bucket occupancy possible**

- **Prefix addressing**

- **Suffix addressing**

© 2011 AG DBIS

External Hashing without Overflow Areas

- **Goal**
  - Each record can be located with **exactly one I/O access**
    - ➥ Chained overflow areas cannot be used
- **Static Hashing**
  - n records, N buckets with capacity b
  - Occupancy factor $\beta = \frac{n}{N \cdot b}$
- **Overflow handling**
  - Open Addressing (without chain or pointer)
  - Best known schemata: Linear Probing and Double Hashing
  - Probing sequence for a record with key k:
    - $H(k) = (h_1(k), h_2(k), \ldots, h_N(k))$
    - determines probing sequence of buckets (pages) for insertions and searches
    - is determined by k and a permutation of the set of bucket addresses {0, 1, ..., N – 1}

External Hashing without Overflow Areas (2)

- **First attempt**
  - Search or insertion of k = xy

8		27		99
ab		**uv**		**lm**
ij	. . .	**cd**	. . .	**xy**
gh		**no**		

(8, 27, 99: bucket address)

- **Probing sequence** is assumed to be H(xy) = (8, 27, 99, ...)
- Many I/O accesses
- How do insertions occur?

→ Solution: see External Hashing Using Separators*

* Larson, P.-A. and Kajla, A.: File organization: implementation of a method guaranteeing retrieval in one access, in: Comm. of the ACM 27,7 (1984), 670-677.

Linear Hashing*

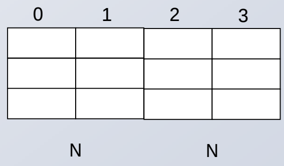
- **Dynamic growth and shrinking of the (primary) hashing area (file)**
  - Minimal administration data
  - No large directories for hash files
- **But: overflow records cannot be completely avoided!**
  - A high rate of overflow records is considered as an indicator that the file is overloaded (β is too high) and must therefore be extended
  - Buckets are split in a strictly specified sequence
  - ➥ Only information: next bucket to be split
- **Principal approach**
  - N: initial size of the file in buckets
  - Sequence of hashing functions

    where $h_0(k) \in \{0, 1, ..., N-1\}$

    and $h_{j+1}(k) = h_j(k)$ or

    $h_{j+1}(k) = h_j(k) + N \cdot 2^j$

    holds for all $j \geq 0$ and all keys k
  - Uniform probability desired for both cases of h_{j+1}



- **Example**
  - $h_j(k) = k \cdot (\text{mod } N \cdot 2^j), j = 0, 1, ...$

* Litwin, W.: Linear hashing: a new tool for files and tables implementation. Proc. 6th Int. Conf. on VLDB, 1980, 212-223.

Realization of DBS

Hashing – overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing without overflow
Linear hashing

Linear Hashing – Example

- **Principle: LH**

$h_0(k_i) \in \{0, ..., N-1\} \rightarrow \{0, 1\}$ for N = 2

$h_1(k_i) = h_0(k_i)$ or

$h_1(k_i) = h_0(k_i) + N \cdot 2^0$

} j = 0

<u>in general:</u>

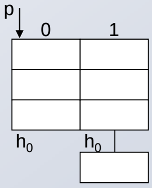
$h_{j+1}(k_i) = h_j(k_i)$ or

$h_{j+1}(k_i) = h_j(k_i) + N \cdot 2^j$ } displaced by $N \cdot 2^j$

- **$h_0(k_i) = k_i \bmod (2^0 * N)$**

insertion sequence: 3, 5, 7, 13, 10

N = 2
b = 3
$\beta_S = 0.8$



$$\beta = \frac{5}{6} = 0.83$$

Extension:

- if $\beta > \beta_S$,
  redistribution of bucket p: p := p+1
- Address computation: $h := h_L(k)$

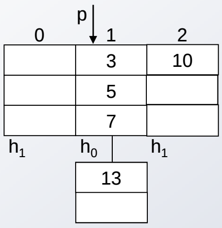
  if h < p then $h := h_{L+1}(k)$

(here: L = 0, number of duplications so far)

Linear Hashing – Example (2)

- $h_1(k_i) = k_i \bmod (2^1 * N)$

sequence: 12, 14, 15

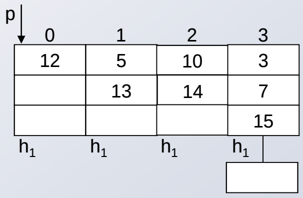


$\beta_S = 0.8$

$$\beta = \frac{8}{9} = 0.88$$

sequence: 19, 24

L = 1 (first duplication)



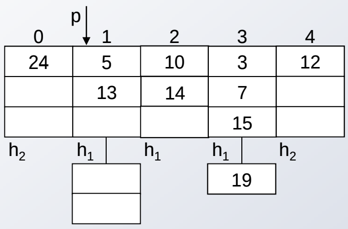
$\beta_S = 0.8$

$$\beta = \frac{10}{12} = 0.83$$

Linear Hashing (2)

- $h_2(k_i) = k_i \bmod (2^2 * N)$

sequence: 17, 21, 25



$\beta_S = 0.8$

β = 13/15 = 0.87

- **Description of the file state**
  - L: number of duplications already done
  - p: points to next bucket to be split ($0 \le p < N \cdot 2^L$)
  - β: occupancy factor $\frac{n}{(N \cdot 2^L + p) \cdot b}$
  - n: number of records stored
  - b: capacity of a bucket

Linear Hashing (3)

- **Example: principle of linear hashing**
  - h_0 (k) = k mod 5
  - h_1 (k) = k mod 10, ...
  - b = 4, L = 0, N= 5
  - Splitting as soon as $\beta > \beta_s = 0.8$

p → bucket 0

Primary buckets:

0	1	2	3	4
105	111	512	413	144
790	076	477	243	
335		837	888	
995		002		

Overflow records:

0	1	2	3	4
055		117		
010				
h_0	h_0	h_0	h_0	h_0

Linear Hashing (4)

- Insertion of 888 increases occupancy to $\beta = 17/20 = 0.85$ and causes splitting

p → bucket 1

0	1	2	3	4	5
790	111	512	413	144	105
010	076	477	243		335
		837	888		995
		002			055
		117 (overflow)			
h_1	h_0	h_0	h_0	h_0	h_1

  - Insertion of 244, 399, and 100. Insertion of 100 causes splitting

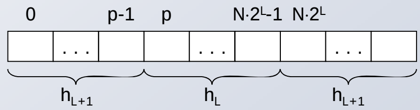
p → bucket 2

$\beta_S = 0.8$ $\beta = 20/24 = 0.83$

0	1	2	3	4	5	6
790	111	512	413	144	105	076
010		477	243	244	335	
100		837	888	399	995	
		002			055	
		117 (overflow)				
h_1	h_1	h_0	h_0	h_0	h_1	h_1

Linear Hashing (5)

- **Splitting**
  - Trigger: β, position: p
  - File is increased by 1
  - p is increased by 1: p : = (p + 1) mod ($N \cdot 2^L$)
  - If p is again set to Null (duplication of file completed), L is increased by 1
- **Address computation**
  - If $h_0(k) \geq p$, then the requested address is h_0
  - If $h_0(k) < p$, then the bucket was already split. $h_1(k)$ delivers the requested address
  - In general: $h := h_L(k)$;
    if $h < p$ then $h := h_{L+1}(k)$;



- **Split strategies**
  - Uncontrolled splitting
    - Splitting as soon as a record enters the overflow area
    - β ~ 0.6, faster location
  - Controlled splitting
    - Splitting if a record enters the overflow area and $\beta > \beta_s$
    - $\beta \sim \beta_s$, longer overflow chains possible

Comparison of the Most Important Access Methods

access method	storage structure	direct access	sequential processing	modification (without location)
sequential key comparison	sequential lists	$O(n) \approx 5 \cdot 10^3$	$O(n) \approx 10^4$	$O(1) \leq 2$
	chained lists	$O(n) \approx 5 \cdot 10^5$	$O(n) \approx 10^6$	$O(1) \leq 3$
tree-based key comparison	balanced binary trees	$O(\log_2 n) \approx 20$	$O(n) \approx 10^6$	O(1) = 2
	multi-way trees	$O(\log_k n) \approx 3 - 4$	$O(n) \approx 10^6$*	O(1) = 2
constant key transformation method	external hashing with separate overflow area	$O(1) \approx 1.1 – 1.4$	$O(n \log_2 n)$**	$O(1) \approx 1.1$
	external hashing with separators	O(1) = 1	$O(n \log_2 n)$**	O(1) = 1 (+D)
variable key transformation method	extendible Hashing	O(1) = 2	$O(n \log_2 n)$**	$O(1) \approx 1.1$ (+R)
	linear hashing	$O(1) = 1 + \delta$	$O(n \log_2 n)$**	O(1) < 2

Example costs based on $n = 10^6$ (D = domino effect, R = reorganization cost)
* in case of clustering up to a factor of 100 faster
** Physical sequential read, sorting and sequential processing of all records