



File Organization

File Organization
Competition

- Cost Model
- Scan
- Equality Test
- Range Selection
- Insertion
- Deletion

Indexes

- Index Entries
- Clustered / Unclustered
- Dense / Sparse

Chapter 3

Indexing

Navigate and Search Large Data Volumes

Architecture and Implementation of Database Systems

Summer 2014



File Organization and Indexes

- A **heap file** provides just enough structure to maintain a collection of records (of a table).
- The heap file supports **sequential scans** (`openScan(.)`) over the collection, *e.g.*

SQL query leading to a sequential scan

```
1 SELECT A,B
2 FROM R
```

But: No further operations receive specific support from the heap file.

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

File Organization and Indexes

- For queries of the following type, it would definitely be helpful if the SQL query processor could rely on a particular **file organization** of the records in the file for table R:

Queries calling for systematic file organization

```
1 SELECT A,B
2 FROM R
3 WHERE C > 42
```

```
1 SELECT A,B
2 FROM R
3 ORDER BY C ASC
```

File organization for table R

Which organization of records in the file for table R could speed up the evaluation of **both** queries above?

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

File Organization and Indexes

- For queries of the following type, it would definitely be helpful if the SQL query processor could rely on a particular **file organization** of the records in the file for table R:

Queries calling for systematic file organization

```
1 SELECT A,B
2 FROM R
3 WHERE C > 42
```

```
1 SELECT A,B
2 FROM R
3 ORDER BY C ASC
```

File organization for table R

Which organization of records in the file for table R could speed up the evaluation of **both** queries above?

Allocate records of table R in **ascending order of C attribute values**, place records on neighboring pages. (Only include columns A, B, C in records.)

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

A Competition of File Organizations

- This chapter presents a comparison of three **file organizations**:
 - 1 Files of **randomly ordered** records (heap files)
 - 2 Files **sorted** on some record field(s)
 - 3 Files **hashed** on some record field(s)
- A file organization is tuned to make a certain query (class) efficient, but if we have to support **more than one query class**, we may be in trouble. Consider:

Query Q calling for organization sorted on column A

```
1 SELECT A,B,C
2 FROM R
3 WHERE A > 0 AND A < 100
```

- If the file for table R is sorted on C, this does not buy us anything for query Q.
- If Q is an important query but is *not* supported by R's file organization, we can build a support data structure, an **index**, to speed up (queries similar to) Q.

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

A Competition of File Organizations

- This competition assesses 3 file organizations in 5 disciplines:

- 1 Scan:** read all records in a given file.
- 2 Search with equality test**

Query calling for equality test support

```
1 SELECT *  
2 FROM R  
3 WHERE C = 42
```

- 3 Search with range selection** (upper or lower bound might be unspecified)

Query calling for range selection support

```
1 SELECT *  
2 FROM R  
3 WHERE A > 0 AND A < 100
```

- 4 Insert** a given record in the file, respecting the file's organization.
- 5 Delete** a record (identified by its *rid*), maintain the file's organization.

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Simple Cost Model

- Performing these 5 database operations clearly involves **block I/O**, the major cost factor.
- However, we have to additionally pay for CPU time used to **search inside a page, compare a record field to a selection constant**, etc.
- To analyze cost more accurately, we introduce the following parameters:

Simple cost model parameters

Parameter	Description
b	# of pages in the file
r	# of records on a page
D	time needed to read/write a d isk page
C	C PU time needed to process a record (e.g., compare a field value)
H	CPU time taken to apply a function to a record (e.g., a comparison or h ash function)

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Simple Cost Model

Simple cost model parameters

Parameter	Description
b	# of pages in the file
r	# of records on a page
D	time needed to read/write a d isk page
C	C PU time needed to process a record (e.g., compare a field value)
H	CPU time taken to apply a function to a record (e.g., a comparison or h ash function)

Remarks:

- $D \approx 15$ ms
- $C \approx H \approx 0.1$ μ s
- This is a coarse model to **estimate** the actual execution time (this does *not* model network access, cache effects, burst I/O, ...).

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Aside: Hashing

A simple hash function

A **hashed file** uses a **hash function** h to map a given record onto a specific page of the file.

Example: h uses the lower 3 bits of the first field (of type INTEGER) of the record to compute the corresponding page number:

$$\begin{aligned}h(\langle 42, \text{true}, \text{'foo'} \rangle) &\rightarrow 2 && (42 = 101010_2) \\h(\langle 14, \text{true}, \text{'bar'} \rangle) &\rightarrow 6 && (14 = 1110_2) \\h(\langle 26, \text{false}, \text{'baz'} \rangle) &\rightarrow 2 && (26 = 11010_2)\end{aligned}$$



Scan
Equality Test
Range Selection
Insertion
Deletion

Index Entries
Clustered / Unclustered
Dense / Sparse

Aside: Hashing

A simple hash function

A **hashed file** uses a **hash function** h to map a given record onto a specific page of the file.

Example: h uses the lower 3 bits of the first field (of type INTEGER) of the record to compute the corresponding page number:

$$\begin{aligned}h(\langle 42, \text{true}, \text{'foo'} \rangle) &\rightarrow 2 && (42 = 101010_2) \\h(\langle 14, \text{true}, \text{'bar'} \rangle) &\rightarrow 6 && (14 = 1110_2) \\h(\langle 26, \text{false}, \text{'baz'} \rangle) &\rightarrow 2 && (26 = 11010_2)\end{aligned}$$

- The hash function determines the page number only; record placement inside a page is not prescribed.
- If a page p is filled to capacity, a chain of **overflow** pages is maintained to store additional records with $h(\langle \dots \rangle) = p$.
- To avoid immediate overflowing when a new record is inserted, pages are typically filled to 80% only when a heap file is initially (re)organized as a hashed file.



Scan Cost

1 Heap file

Scanning the records of a file involves **reading all b pages** as well as **processing each of the r records on each page**:

$$\text{Scan}_{\text{heap}} = b \cdot (D + r \cdot C)$$

2 Sorted file

The sort order does not help much here. However, the scan retrieves the records in sorted order (which can be big plus later on):

$$\text{Scan}_{\text{sort}} = b \cdot (D + r \cdot C)$$

3 Hashed file

Again, the hash function does not help. We simply scan from the beginning (skipping over the spare free space typically found in hashed files):

$$\text{Scan}_{\text{hash}} = \underbrace{(100/80)}_{=1.25} \cdot b \cdot (D + r \cdot C)$$

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Cost for Search With Equality Test ($A = \text{const}$)

1 Heap file

Consider (a) the equality test is on a *primary key*, (b) the equality test is *not* on a *primary key*:

$$(a) \text{ Search}_{\text{heap}} = 1/2 \cdot b \cdot (D + r \cdot (C + H))$$

$$(b) \text{ Search}_{\text{heap}} = b \cdot (D + r \cdot (C + H))$$

2 Sorted file (sorted on A)

We assume the equality test to be on the field determining the sort order. The sort order enables us to use **binary search**:

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Cost for Search With Equality Test ($A = const$)

1 Heap file

Consider (a) the equality test is on a *primary key*, (b) the equality test is *not* on a *primary key*:

$$(a) \text{ Search}_{\text{heap}} = 1/2 \cdot b \cdot (D + r \cdot (C + H))$$

$$(b) \text{ Search}_{\text{heap}} = b \cdot (D + r \cdot (C + H))$$

2 Sorted file (sorted on A)

We assume the equality test to be on the field determining the sort order. The sort order enables us to use **binary search**:

$$\text{Search}_{\text{sort}} = \log_2 b \cdot (D + \log_2 r \cdot (C + H))$$

If more than one record qualifies, all other matches are stored right after the first hit.

(Nevertheless, **no DBMS** will implement binary search for value lookup.)



Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Cost for Search With Equality Test ($A = \text{const}$)

- 3 **Hashed file** (hashed on A)
Hashed files support equality searching best. The hash function directly leads us to the page containing the hit (overflow chains ignored here).

Consider (a) the equality test is on a *primary key*, (b) the equality test is *not* on a *primary key*:

$$(a) \text{ Search}_{\text{hash}} = H + D + 1/2 \cdot r \cdot C$$

$$(b) \text{ Search}_{\text{hash}} = H + D + r \cdot C$$

No dependence on file size b here. (All qualifying records live on the same page or, if present, in its overflow chain.)

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan

Equality Test

Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Cost for Range Selection ($A \geq \text{lower}$ AND $A \leq \text{upper}$)

1 Heap file

Qualifying records can appear anywhere in the file:

$$\text{Range}_{\text{heap}} = b \cdot (D + r \cdot (C + 2 \cdot H))$$

2 Sorted file (sorted on A)

Use equality search (with $A = \text{lower}$), **then sequentially scan** the file until a record with $A > \text{upper}$ is encountered:

$$\text{Range}_{\text{sort}} = \log_2 b \cdot (D + \log_2 r \cdot (C + H)) + \lceil n/r \rceil \cdot D + n \cdot (C + H)$$

($n + 1$ overall hits in the range, $n \geq 0$)



Cost for Range Selection ($A \geq \text{lower}$ AND $A \leq \text{upper}$)

3 Hashed file (hashed on A)

Hashing offers no help here as hash functions are designed to **scatter** records all over the hashed file (e.g., for the h we considered earlier: $h(\langle 7, \dots \rangle) = 7, h(\langle 8, \dots \rangle) = 0$):

$$\text{Range}_{\text{hash}} = (100/80) \cdot b \cdot (D + r \cdot (C + H))$$

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan
Equality Test

Range Selection

Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Insertion Cost

1 Heap file

We can add the record to some **arbitrary** free page (finding that page is not accounted for here).

This involves reading and writing that page:

$$\text{Insert}_{\text{heap}} = 2 \cdot D + C$$

2 Sorted file

On average, the new record will belong in the middle of the file. After insertion, we have to **shift all subsequent records** (in the second half of the file):

$$\text{Insert}_{\text{sort}} = \log_2 b \cdot (D + \log_2 r \cdot (C + H)) + 1/2 \cdot b \cdot (\underbrace{D + r \cdot C + D}_{\text{read + shift + write}})$$

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan
Equality Test
Range Selection

Insertion

Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Insertion Cost

3 Hashed file

We pretend to search for the record, then read and write the page determined by the hash function (here we assume the spare 20 % space on the page is sufficient to hold the new record):

$$\text{Insert}_{\text{hash}} = \underbrace{H + D}_{\text{search}} + C + D$$

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan
Equality Test
Range Selection

Insertion

Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Deletion Cost (Record Specified by its *rid*)

1 Heap file

If we do not try to compact the file after we have found and removed the record (because the file uses free space management), the cost is:

$$\text{Delete}_{\text{heap}} = \underbrace{D}_{\text{search by } rid} + C + D$$

2 Sorted file

Again, we access the record's page and then (on average) shift the second half the file to compact the file:

$$\text{Delete}_{\text{sort}} = D + 1/2 \cdot b \cdot (D + r \cdot C + D)$$

3 Hashed file

Accessing the page using the *rid* is even faster than the hash function, so the hashed file behaves like the heap file:

$$\text{Delete}_{\text{hash}} = D + C + D$$

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

The “Best” File Organization?

- There is **no single file organization** that responds equally fast to all 5 operations.
- This is a dilemma, because more advanced file organizations can really make a difference in speed.

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

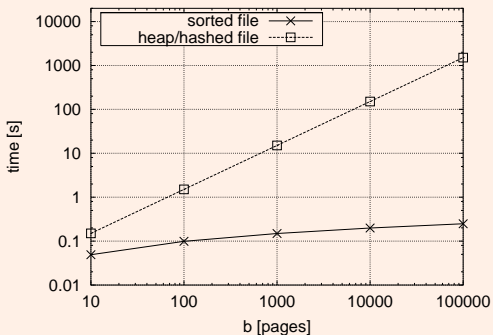
Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Range selection performance

- Performance of **range selections** for files of increasing size ($D = 15 \text{ ms}$, $C = 0.1 \mu\text{s}$, $r = 100$, $n = 10$):

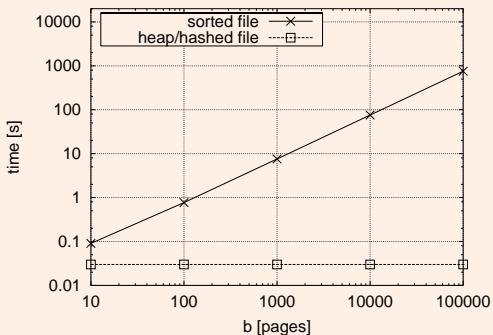
Range Selection Performance



Deletion Performance Comparison

- Performance of **deletions** for files of increasing size ($D = 15 \text{ ms}$, $C = 0.1 \mu\text{s}$, $r = 100$):

Deletion Performance



Indexes

- There exist **index structures** which offer all the advantages of a sorted file *and* support insertions/deletions efficiently¹: **B⁺-trees**.
- Before we turn to B⁺-trees in detail, the following sheds some light on indexes in general.

¹At the cost of a modest space overhead.

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Indexes

- If the basic organization of a file does not support a particular operation, we can **additionally maintain** an auxiliary structure, an **index**, which adds the needed support.
- We will use indexes like **guides**. Each guide is specialized to accelerate searches on a specific attribute A (or a combination of attributes) of the records in its associated file:

Index usage

- 1 Query the index for the location of a record with $A = k$ (k is the **search key**).
- 2 The index responds with an associated **index entry** k^* (k^* contains sufficient information to access the actual record in the file).

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Indexes

- If the basic organization of a file does not support a particular operation, we can **additionally maintain** an auxiliary structure, an **index**, which adds the needed support.
- We will use indexes like **guides**. Each guide is specialized to accelerate searches on a specific attribute A (or a combination of attributes) of the records in its associated file:

Index usage

- 1 Query the index for the location of a record with $A = k$ (k is the **search key**).
- 2 The index responds with an associated **index entry** k^* (k^* contains sufficient information to access the actual record in the file).
- 3 Read the actual record by using the guiding information in k^* ; the record will have an A-field with value k .



Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$
C	$\langle k, [rid_1, rid_2, \dots] \rangle$

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$
C	$\langle k, [rid_1, rid_2, \dots] \rangle$

Remarks:

- With variant A, there is *no* need to store the data records in addition to the index—the index itself is a special file organization.

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$
C	$\langle k, [rid_1, rid_2, \dots] \rangle$

Remarks:

- With variant A, there is *no* need to store the data records in addition to the index—the index itself is a special file organization.
- If we build multiple indexes for a file, at most one of these should use variant A



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$
C	$\langle k, [rid_1, rid_2, \dots] \rangle$

Remarks:

- With variant A, there is *no* need to store the data records in addition to the index—the index itself is a special file organization.
- If we build multiple indexes for a file, at most one of these should use variant A to avoid redundant storage of records.



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Index entries

- We can design the **index entries**, *i.e.*, the k^* , in various ways:

Index entry designs

Variant	Index entry k^*
A	$\langle k, \langle \dots, A = k, \dots \rangle \rangle$
B	$\langle k, rid \rangle$
C	$\langle k, [rid_1, rid_2, \dots] \rangle$

Remarks:

- With variant **A**, there is *no* need to store the data records in addition to the index—the index itself is a special file organization.
- If we build multiple indexes for a file, at most one of these should use variant **A** to avoid redundant storage of records.
- Variants **B** and **C** use *rid(s)* to point into the actual data file.
- Variant **C** leads to less index entries if multiple records match a search key k , but index entries are of variable length.



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Indexing Example

Example (Indexing example, see following slide)

- The data file contains $\langle \text{name, age, sal} \rangle$ records of table *employees*, the file itself (index entry variant **A**) is hashed on field *age* (hash function *h1*).
- The index file contains $\langle \text{sal, rid} \rangle$ index entries (variant **B**), pointing into the data file.
- This file organization + index efficiently supports equality searches on the **age** **and** *sal* keys.

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries

Clustered / Unclustered
Dense / Sparse

Indexing Example

Indexing

Torsten Grust



File Organization

File Organization
Competition

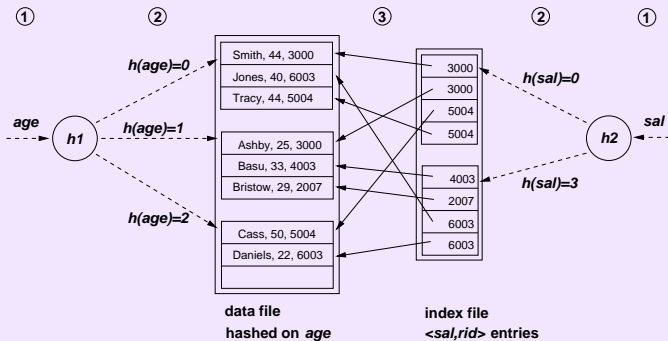
- Cost Model
- Scan
- Equality Test
- Range Selection
- Insertion
- Deletion

Indexes

Index Entries

- Clustered / Unclustered
- Dense / Sparse

Example (Indexing example)



Index Properties: Clustered vs. Unclustered

- Suppose, we have to support **range selections** on records such that $A \geq \textit{lower}$ AND $A \leq \textit{upper}$.
- If we maintain an index on the A-field, we can
 - ① **query the index** *once* for a record with $A = \textit{lower}$, and then
 - ② **sequentially scan the data file** from there until we encounter a record with field $A > \textit{upper}$.
- **However:** This switch from index to data file will only work provided that the **data file itself is sorted on the field A**.

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Index Properties: Clustered vs. Unclustered

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

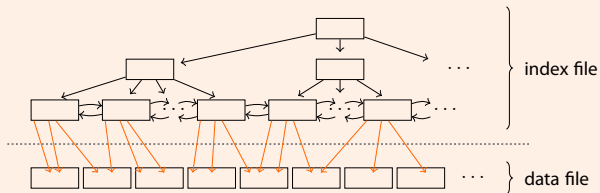
Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Index over a data file with matching sort order



Remark:

- In a B⁺-tree, for example, the index entries k^* stored in the leaves are sorted by key k .

Index Properties: Clustered vs. Unclustered

Definition (Clustered index)

If the **data file** associated with an index is **sorted on the index search key**, the index is said to be **clustered**.

In general, the cost for a range selection grows tremendously if the index on A is **unclustered**. In this case, proximity of index entries does *not* imply proximity in the data file:



- As before, we can query the index for a record with $A = \text{lower}$.

To continue the scan, however, we have to **revisit the index entries** which point us to **data pages scattered** all over the data file.

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

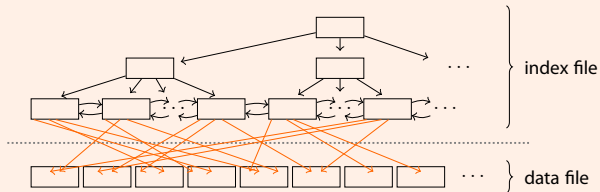
Index Entries

Clustered / Unclustered

Dense / Sparse

Index Properties: Clustered vs. Unclustered

Unclustered index



Remarks:

- If an index uses entries k^* of variant **A**, the index is obviously clustered by definition.

Variant **A** in Oracle 8i

```
CREATE TABLE ... (... PRIMARY KEY (...)) ORGANIZATION INDEX;
```

- A data file can have at most one clustered index (but any number of unclustered indexes).



Index Properties: Clustered vs. Unclustered

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

DB2. Clustered indexes

Create a clustered index IXR on table R, index key is attribute A:

```
CREATE INDEX IXR ON R(A ASC) CLUSTER
```

The DB2 V9.5 manual says:

*"[CLUSTER] specifies that the index is **the** clustering index of the table. The **cluster factor** of a clustering index is maintained or improved dynamically as data is inserted into the associated table, by attempting to **insert new rows physically close to the rows for which the key values of this index are in the same range**. Only one clustering index may exist for a table so CLUSTER may not be specified if it was used in the definition of any existing index on the table (SQLSTATE 55012). A clustering index may not be created on a table that is defined to use append mode (SQLSTATE 428D8)."*

Index Properties: Clustered vs. Unclustered



Cluster a table based on an existing index

Reorganize rows of table R so that their physical order matches the *existing* index IXR:

```
CLUSTER R USING IXR
```

- If IXR indexes attribute A of R, rows will be sorted in ascending A order.
- The evaluation of range queries will **touch less pages** (which additionally, will be physically adjacent).
- **Note:** Generally, future insertions will compromise the perfect A order.
 - May issue CLUSTER R again to re-cluster.
 - In CREATE TABLE, use WITH (fillfactor = f), $f \in 10 \dots 100$, to reserve page space for subsequent insertions.

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Index Properties: Clustered vs. Unclustered

DB2. Inspect clustered index details

```
1 db2 => reorgchk update statistics on table grust.accel
```

```
2  
3 Doing RUNSTATS ...
```

```
4  
5 Table statistics:
```

```
6 .  
7 .
```

```
8 Index statistics:
```

```
9 F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
```

```
10  
11 SCHEMA.NAME                INDCARD  LEAF LVLS    KEYS  F4  REORG  
12 -----  
13 Table: GRUST.ACCEL  
14 Index: GRUST.IKIND          235052   529    3      3  99  ----  
15 Index: GRUST.IPAR           235052   675    3  67988  99  ----  
16 Index: GRUST.IPOST          235052   980    3 235052 100  ----  
17 Index: GRUST.ITAG           235052   535    3     75  80  *----  
18 Index: SYSIBM.SQLO80119120245000  
19                               235052   980    3 235052 100  ----  
20 -----
```



File Organization

File Organization Competition

- Cost Model
- Scan
- Equality Test
- Range Selection
- Insertion
- Deletion

Indexes

- Index Entries
- Clustered / Unclustered
- Dense / Sparse

Index Properties: Dense vs. Sparse

A **clustered index** comes with more advantages than the improved performance for range selections. We can additionally design the index to be **space efficient**:

Definition (Sparse index)

To keep the size of the index small, maintain **one index entry k^* per data file page** (not one index entry per data record). Key k is the **smallest key** on that page.

Indexes of this kind are called **sparse**. (Otherwise, indexes are referred to as **dense**.)

Indexing

Torsten Grust



File Organization

File Organization
Competition

Cost Model

Scan

Equality Test

Range Selection

Insertion

Deletion

Indexes

Index Entries

Clustered / Unclustered

Dense / Sparse

Index Properties: Dense vs. Sparse

Indexing

Torsten Grust



File Organization

File Organization Competition

- Cost Model
- Scan
- Equality Test
- Range Selection
- Insertion
- Deletion

Indexes

- Index Entries
- Clustered / Unclustered
- Dense / Sparse

To search a record with field $A = k$ in a sparse A -index,

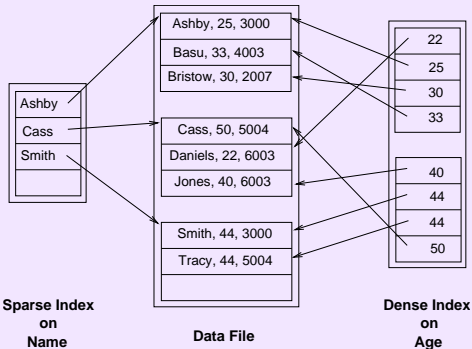
- 1 Locate the largest index entry k'^* such that $k' \leq k$, then
- 2 access the page pointed to by k'^* , and
- 3 scan this page (and the following pages, if needed) to find records with $\langle \dots, A = k, \dots \rangle$.

Since the data file is clustered (*i.e.*, sorted) on field A , we are guaranteed to find matching records in the proximity.

Index example (Dense vs. Sparse)

Example (Sparse index example)

- Again, the data file contains $\langle \text{name, age, sal} \rangle$ records. We maintain a **clustered sparse index** on field `name` and an **unclustered dense index** on field `age`. Both use index entry variant **B** to point into the data file.



Index Properties: Dense vs. Sparse

Note:

- Sparse indexes need 2–3 orders of magnitude less space than dense indexes (consider # records/page).
- We *cannot* build a sparse index that is unclustered (*i.e.*, there is at most one sparse index per file).

SQL queries and index exploitation

How do you propose to evaluate the following SQL queries?

```
1 SELECT MAX(age)
2 FROM employees
```

```
1 SELECT MAX(name)
2 FROM employees
```

Indexing

Torsten Grust



File Organization

File Organization Competition

Cost Model
Scan
Equality Test
Range Selection
Insertion
Deletion

Indexes

Index Entries
Clustered / Unclustered
Dense / Sparse