INF115 Lecture 11: *Files and Indices*

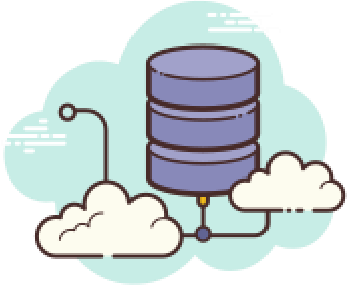
Adriaan Ludl
Department of Informatics
University of Bergen

Spring Semester 2021

Second Hand-In from 19th to 29th of March

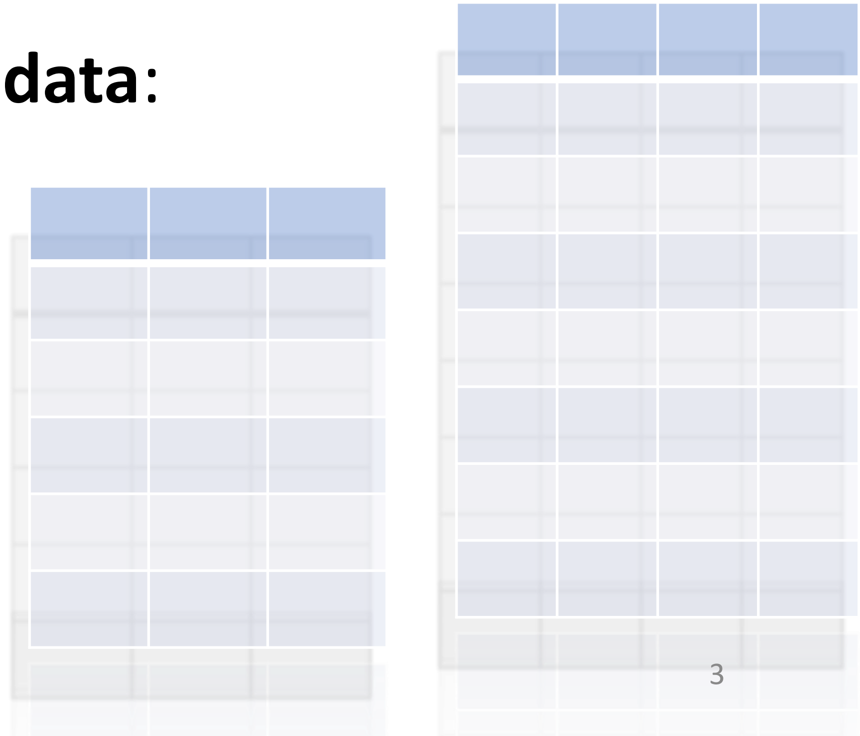
- This assignment is new and different from previous years.
- Go to your group sessions next week to work on it together with your peers.
- If you use outside resources cite them !
- If you work in groups put the names of your collaborators on the document that you hand in.

Chapter 9: *Files and Indices*



Learning Goals:

- Understand the **properties** of relevant **physical storage media**
- **How are different storage media used** in a database system
- Important **techniques for storage and use of data**:
  - **File** structures
  - **Search** techniques
  - **Indices**
- **Use SQL to define indices**



Topics

- Storage Media
  - Properties and use cases
  - RAM and disk
- Files
  - Blocks
  - Non-sorted files (heap files) and sorted files
  - Sequential search and binary search
- Indices
  - Dense and sparse indices
  - Multi-level indices and B^+ trees
  - Define indices with SQL
- Choice of a physical database design

Storage Media

Examples:

- **Memory** (RAM)
- Flash memory
- Solid State Disk (**SSD**)
- **Hard disk**
- Optical media (DVD, CD, …)
- Magnetic tape (magnetbånd)

Properties:

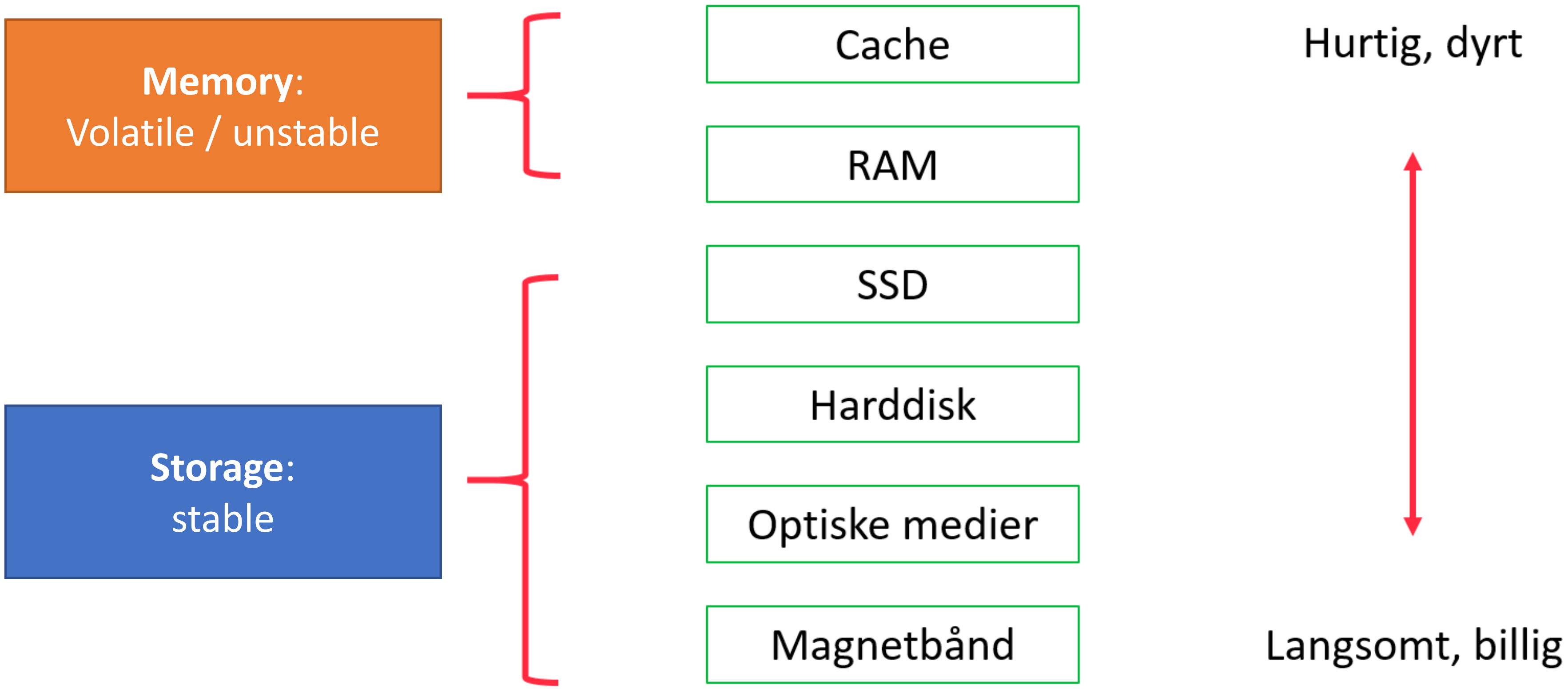
- Access **speed**
- **Price** per byte
- **Stability** / **Volatility** (depending on electric current)
- Sequential / **direct** access
- **Noise**, **heat**, **robustness**, and more !

High speed

= high price

(as for ski …)

Storage Media

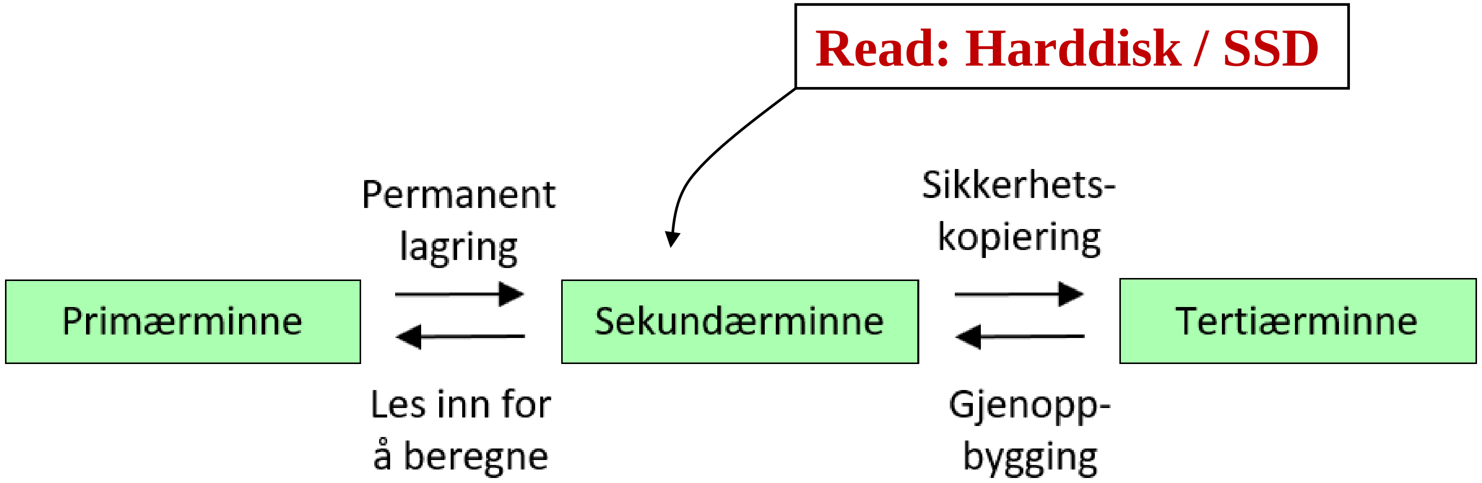


Use of Storage Media

In general, the **memory** is *not large enough to hold the full database*.

- ➢ Data is read into memory from the disk to run calcualtions (fast).
- ➢ Updated data is written back to the disk (stable storage medium).

The DBMS continually moves data around between memory and disk.

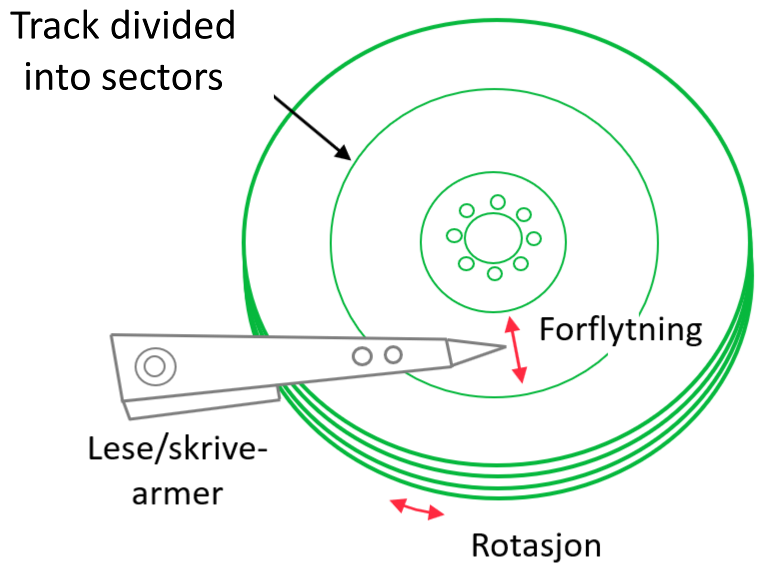


Disk

- **Disk = permanent storage**

- **Hard disk drive (HDD):**
  - Rotating magnetic platters.
  - Mechanical movement takes time
- Time it takes to read / write:
  - **Positioning** read/write arms +
  - Wait for the right sector to **pass** under the head+
  - Transfer data from/to RAM.

- **Solid State Disk (SSD)**
  - No movable parts
  - Much faster for single lookup
  - **Replacing hard disks in many areas**



Hard disk

I/O operations

Relative difference in access time (single lookup = «enkeltoppslag»)

- RAM: **50 nano**seconds
- SSD: **0.05** **milli**seconds = 50 microseconds
- Hard disk: **5** **milli**seconds
- 1 nanosecond = 10^{-9} seconds, 1 millisecond = 10^{-3} seconds

Data transfer rate («read in one file»)

- Lesser difference between hard disk and SSD

Anyhow: **RAM is much faster than disk!**

- **I/O operations:**
  - **Reading** data from the hard disk is called an **input operation**
  - **Writing** data to the hard disk is called an **output operation**
  - **I/O operations** are the combined designation for **reading/writing**.
- **Strategy for the DBMS**: **Minimise the number of I/O operations !**

Representation of tables

Any storage medium is a **numbered sequence of bytes**.

How do we store a table (2 dimensional) as a byte sequence on the disk ?

- Row by row or column by column ?
- Every simple value may require several bytes.
- A simple character can need more than one byte (in Unicode).
- A database stored on external storage is organised in one or several **files**.

7
P
e
r
…
8
…

A file is composed of **records** (norsk: poster) which are made of **fields**.

Simplification:

- Every table is stored in its own file.
- Every row is stored in one record.
- Every value in one field.

Quizz on *Files and Indices* (part 1)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

Link:

- https://mitt.uib.no/courses/27455/quizzes

Terminology

From **SQL** to **relational model** to **ER** to **physical files** :

SQL	**Rel.mod.**	**ER**	**Filer**
Table	Relation	Entity type	File
Row	Tuple	Entity instance	Post
Column	Attribute	Attribute	Field

Note: These concepts are not identical.

Fixed and variable record length

The datatype VARCHAR(n) gives a field of **variable length**.

- If one or several fields have variable length then the whole record needs to have variable length.
- Storing several types of record in the same file can also yield variable record length.

If **all fields** have datatypes that give **fixed length**:

- In a file with **record length *n*** can record 1 be stored from address 1,
  record 2 from address *n+1*,
  record 3 from address 2n+1, and so on ...
- Simple an effective to find a record again.

Given variable record length it is not so simple.

- We can use separators between fields and records.
- Searching is required in such cases to find records again.

Blocks

- A hard disk (HD) is much slower than the RAM.
- When we read from or write to the HD it takes some time.

- A **block** is the smallest unit to transfer data between the RAM and the external storage.
  - Typical block size: : 4KB (4 096 byte).
  - A block can in general contain many records (rows).

- The DBMS keeps an overview over which data is stored in which blocks.

A file divided into blocks

	KNr	Fornavn	Etternavn	Telefon	
Post 1	1	Elias	Hansen	99 88 77 66	Blokk 1
Post 2	2	Hulda	Akselsen	31 45 88 21	
Post 3	5	John	Boine	23 94 53 18	
	...	...	...	...	
Post 58	63	Mari	Gygre	55 66 77 88	
Post 59	64	Michael	Svensen	19 82 37 64	Blokk 2
Post 60	69	Robert	Romman	91 28 73 46	
Post 61	72	Laura	Eika	64 37 82 19	
	...	...	...	...	
Post 116	129	Andreas	Karlsen	51 15 83 38	
	...	...	...	...	

A file is stored in a number of blocks.

We can keep aside some space in each block.

- This helps when inserting or removing records.
- The filling ratio (eg. 80%) can be controlled in some DBMS.

Computing disk space usage

We should know/estimate how much space a database requires.

How much space does the *Products* table (Vare) of *Hobbyhuset* need *?*

Vare(VNr, Betegnelse, Pris, KatNr, Antall, Hylle)

Computing disk space usage

We should know/estimate how much space a database requires.

How much space does the *Products* table (Vare) of *Hobbyhuset* need ?

Vare(VNr, Betegnelse, Pris, KatNr, Antall, Hylle)

- Expect ca. 3000 products.
- Every character in CHAR/VARCHAR takes 2 bytes (Unicode).
- Suppose that VARCHAR(n) needs $2 \times n$ bytes.
- **Record length** (see appendix B for datatypes in Vare):
  $2 \times 5 + 2 \times 50 + 8 + 2 + 4 + 2 \times 3 = 130$ byter.

- Suppose block size = 4KB and filling ratio 80%.
- Records per block: $4000 \times 0.8/130 = 25$ (round 4096 to 4000)
- **Number of blocks**: 3000/25 = 120.
- **Required space: $120 \times 4000 = 480\ 000$**

So the Vare table needs ca. 480 KB.

Break !
Lecture resumes in 15 minutes

File structures

- **Non-sorted files** (Heap files): «random» order.
  - New records are added at the end

Every box represents a record

87	27	7	16	41	19	**50**			

- **Sorted files (**sequential files): Sorted, e.g. with respect to VNr.

7	16	19	27	41	**50**	87			

A number of records in every block

Sequential search and binary search

Suppose we are to find a specific value in a file with **100,000 records**.

- Assume the records can be stored in **2000** blocks.
- Only the number of I / O operations (block read-ins) counts!

Using sequential search:

- Start with block 1 and continue…
- On average, we need **2000/2** = 1000 read operations.

Sequential search and binary search

Suppose we are to find a specific value in a file with **100,000 records**.

- Assume the records can be stored in **2000** blocks.
- Only the number of I / O operations (block read-ins) counts!

Using sequential search:

- Start with block 1 and continue…
- On average, we need **2000/2** = **1000** read operations.

If the file is **sorted** (in the field we are searching in) we can use **binary search**.

- Start with the middle block. Continue left or right.
- On average, we need $\log_2$ (**2000**) read operations (2^{11}=2048).

- ➢ Can we reduce the number of blocks that need to be examined **further**?
- ➢ Can we apply this effectively on **several criteria**?
  - Note: A file can only be physically sorted on **one** field.
- ➢ Sorted files require extra **operations** for INSERT, UPDATE and DELETE!

Indices

An **index** is a data structure that can **optimise search**.

- We can make **several indices** for one table.
- But this requires additional **storage space**.
- It must be kept up-to-date: **additional operations for changes**.

An index is like a reference list at the back of a book.

- Contains **keywords** + **references** (= page numbers in a book).
- The list is sorted by **keywords**.
- In database indices, the references will be disk addresses.

It is faster to search the index than the data file, because:

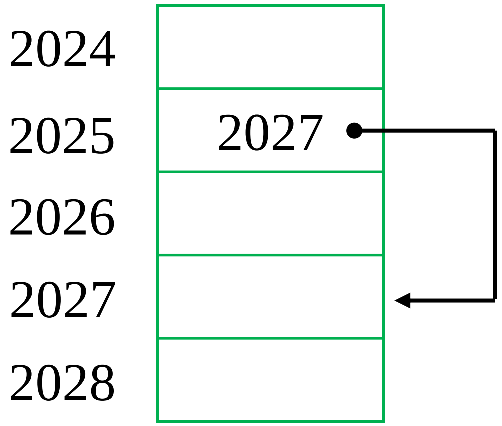
- The index is **smaller** and it is **sorted.**

References (pointers)

A storage medium is a (long) **sequence of bytes**.

Every **cell** has an **address** (a number).

- If one cell contains the address of another, we call it a **reference** (it refers / points to the other cell).
- We often draw references as **arrows**.



- The cell at adress 2025 contains the adress (number) 2027.
- It can be symbolised by an arrow from cell 2025 to cell 2027.

Visualisation of indices

Nøkkel	Adresse
Akselsen	
Boine	
Eika	
Gygre	
Hansen	
Karlsen	
Romman	
Svensen	

KNr	Fornavn	Etternavn	Telefon
1	Elias	Hansen	99 88 77 66
2	Hulda	Akselsen	31 45 88 21
5	John	Boine	23 94 53 18
63	Mari	Gygre	55 66 77 88
64	Michael	Svensen	19 82 37 64
69	Robert	Romman	91 28 73 46
72	Laura	Eika	64 37 82 19
129	Andreas	Karlsen	51 15 83 38

Make indices with SQL

The **simplest** variant:

```
CREATE INDEX VarenavnIdx
ON Vare ( Varenavn )
```

Indices can **avoid repetitions**:

```
CREATE UNIQUE INDEX EtternavnIdx
ON Ansatt ( Etternavn )
```

Indices can be **composite**:

```
CREATE INDEX NavnIdx
ON Ansatt ( Etternavn, Fornavn )
```

Here we get an index on Etternavn as well !

Dense and sparse indices

- A **dense index** (norsk: tett) has an entry for every single entry in the file.
- A **sparse index** (norsk: ikke-tett) has an entry for each block in the file.

- **Sparse indices are smaller** than dense ones.
  - Usually many records in each block.

- **At most <u>one sparse index</u> per data file**.
  - A data file can only be sorted physically in one way, eg. by Kundenr.
  - Only the index on Kundenr can be sparse.

Quizz on *Files and Indices* (part 2)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

Link:

- https://mitt.uib.no/courses/27455/quizzes

B^+ trees

Multi-level indices

- For large indices, you can create «indices-on-indices».
- The indices then form a **tree structure**.
- The indices at level 2, 3, 4, etc. may be sparse.
  - The level below is an index, which is sorted.
- Search goes from the «root» in the tree towards the «leaves».
- The number of disk accesses = the **depth of the tree**.

B^+ trees are a common form of multi-level index:

- ❖ Equally many disk access operations for all search keywords :
- ❖ The tree is **B**alansed.

B^+ trees

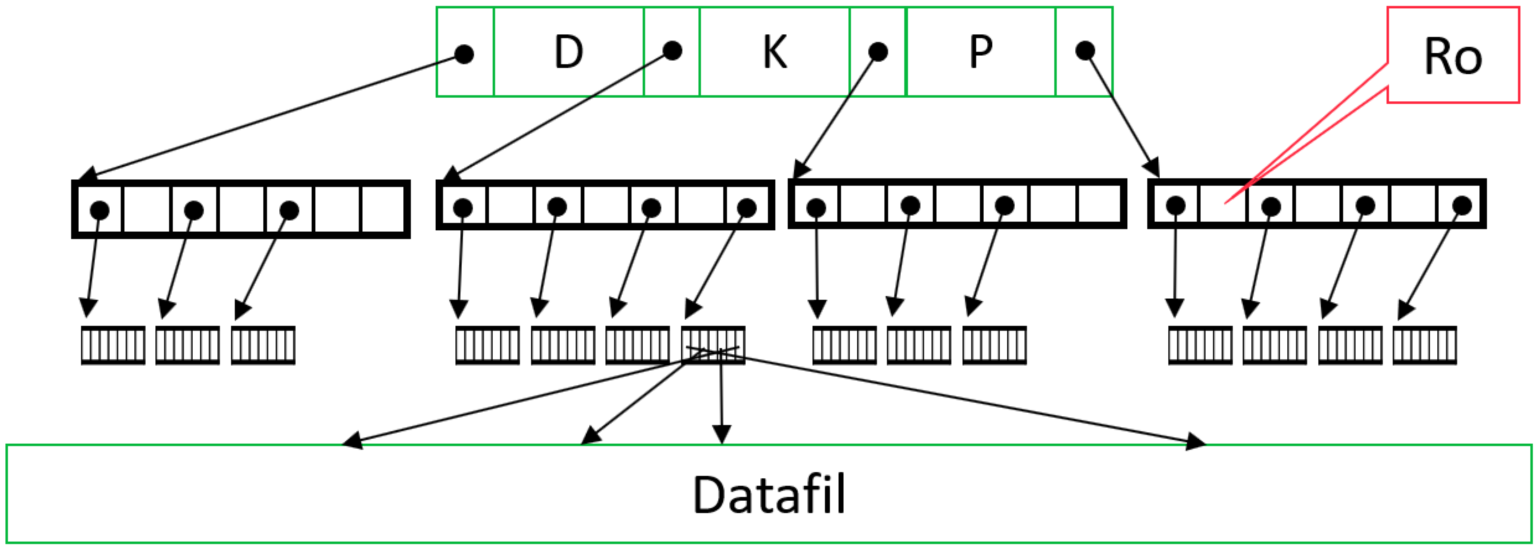
Number of read operations

1

2

3

4



Suppose we look for the record where *Etternavn* = «Roheim».

- Read in the root block.
- Search the root block: Here we will follow the pointer on the right.
- Repeat search at intermediate levels …
- … Read the correct block from the data file.

Choice of datatypes

Text

- Use CHAR (n) if the values have a **fixed length** (birth number, registration number for cars), and VARCHAR (n) otherwise.
- Set large enough upper limit.
- Dedicated data types for very long texts.

Numbers

- Choose as "small" a data type as possible, but so that all conceivable values can fit.
- Integers or decimals?
- Fixed number of decimal places? Krone amount?

Text or numbers?

- Are you going to compute on the values?

Time

- Date and/or time?

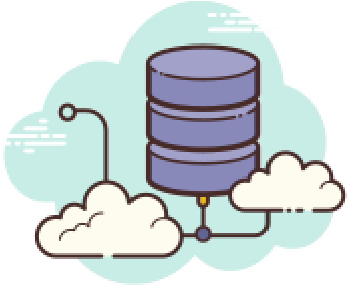
Selection of indices

- It is **advantageous to index** the following types of columns
  - Columns you often search in, or sort on.
  - Primary keys (usually automatically indexed)
  - Foreign keys (can also be automatically indexed)
  - Columns that are often used as join columns in queries.

- The following columns **should not / cannot** be indexed:
  - Columns that contain only a few different values / many equal values,
    - for example Yes / No columns (bitmap indexes possible)
  - Pictures, video, sound clips
  - Long documents (there are special techniques, cf. search engines)

- Indexes require **machine resources**:
  - Indexes take up space.
  - Indices must be maintained when updating in the tables.

Chapter 9: *Files and Indices*



Summary:

- ➢ Understand the **properties** of relevant **physical storage media**
- ➢ **How are different storage media used** in a database system
- ➢ Important **techniques for storage and use of data**:
  - **File** structures
  - **Search** techniques
  - **Indices**
- ➢ **Use SQL to define indices**