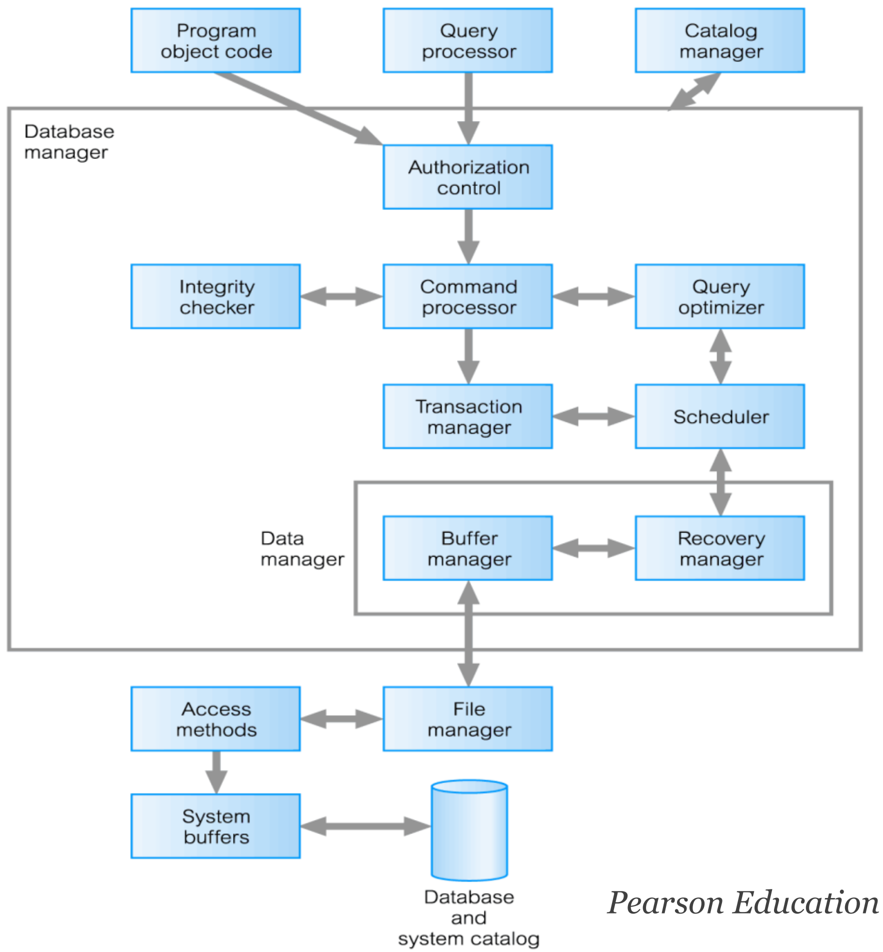
WHERE ARE WE NOW?

W34: Introduction, DBMSs and Relational Databases
W35: Developing Database Systems
W36: SQL –Part I
W37: SQL –Part II and Relational Algebra
W38: Data Modelling
W39: Data Modelling
W40: Database Design
W41: Normalisation and Stored Procedures
W42: XML and Web Technology
W43: Processing XML Data
W44: XML Validation
W45: Beyond relational databases and XML
W46: File Organisations and Indexes
W47: Database Security and Administration
W48: Transaction Processing and Wrap-up

CONTENT

- *Database file organisation and access methods:*
  - » Introduction
  - » Heaps
  - » Ordered files
  - » Hash files
- *Indexes and index file organisation:*
  - » Primary and secondary indexes
  - » B+-trees
  - » Bitmap indexes
- *Managing Indexes in SQL*

DATABASE MANAGER



STORAGE MEDIA (1)

- Primary vs secondary storage:
  - *Primary storage:*
    - » Fast random access (ns)
    - » Volatile
    - » Expensive
  - *Secondary storage:*
    - » “Fast” (ms) – six orders of magnitude slower than main memory
    - » Persistent (non-volatile) data storage
    - » “Inexpensive” - two orders of magnitude cheaper than main memory

STORAGE MEDIA(2)

- DBMSs store data on disks:
  - *Although some real-time database systems rely on in-memory databases*
- Disk characteristics:
  - *Unit of read/write operations:*
    - » a logical block/page storing several rows
  - *Access time:*
    - » seek time + rotation time + transfer time
  - *Sequential I/O much faster than random I/O*

STORAGE MEDIA(3)

- Methods to improve main memory-disk data transfer:
  - *General approaches:*
    - » Improve the disk technology
    - » Use faster disks (more RPMs)
    - » Parallelization (RAID) + some redundancy
    - » Other (OS based improvements): disk scheduling (elevator algorithm, batch writes, etc)
  - *DBMS approaches:*
    - » Good file organization
    - » Avoid unnecessary reads from disk
    - » Buffer management: go to buffer instead of disk

INTRODUCTION TO FILE ORGANIZATION (1)

- Basics:
  - *A database holds a collection of files, file holds a collection of records, record (tuple) is a collection of fields (attributes)*
  - *Some database systems (e.g., MySQL and Oracle) allow files to be grouped into tablespaces*
  - *Files are stored on disks*

- Two important issues:
  - *Representation of each record*
  - *Grouping/Ordering of records and storage in blocks*

INTRODUCTION TO FILE ORGANIZATION (2)

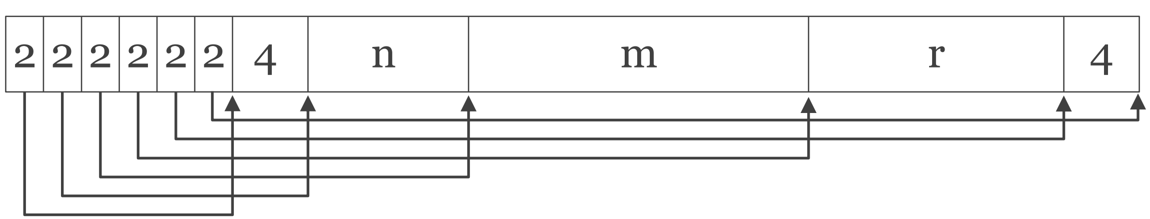
- Goals and considerations:
  - *Compactness*
  - *Overhead of insertion/deletion*
  - *Retrieval speed:*
    - » sometime we prefer to bring more tuples than necessary into main memory and use CPU to filter out the unnecessary ones!

DATABASE RECORD REPRESENTATION (1)

- Fixed-size records:
  - *Store record* i *starting from byte* $n * (i - 1)$, *where* n *is the size of each record.*
  - *Might not allow records to cross block boundaries*
  - *Record access is simple*
  - *Example:*
    - » Ordre (kundeid int, dato date, varenummer char(16), antall int)
    - » Size n in bytes: 4 + 3 + 16 + 4 = 27
    - » A 4KB block can hold $\lfloor 4096/27 \rfloor$ = 151 records
      - *Leaving 19 unused bytes per block (4096 – 151*27)*

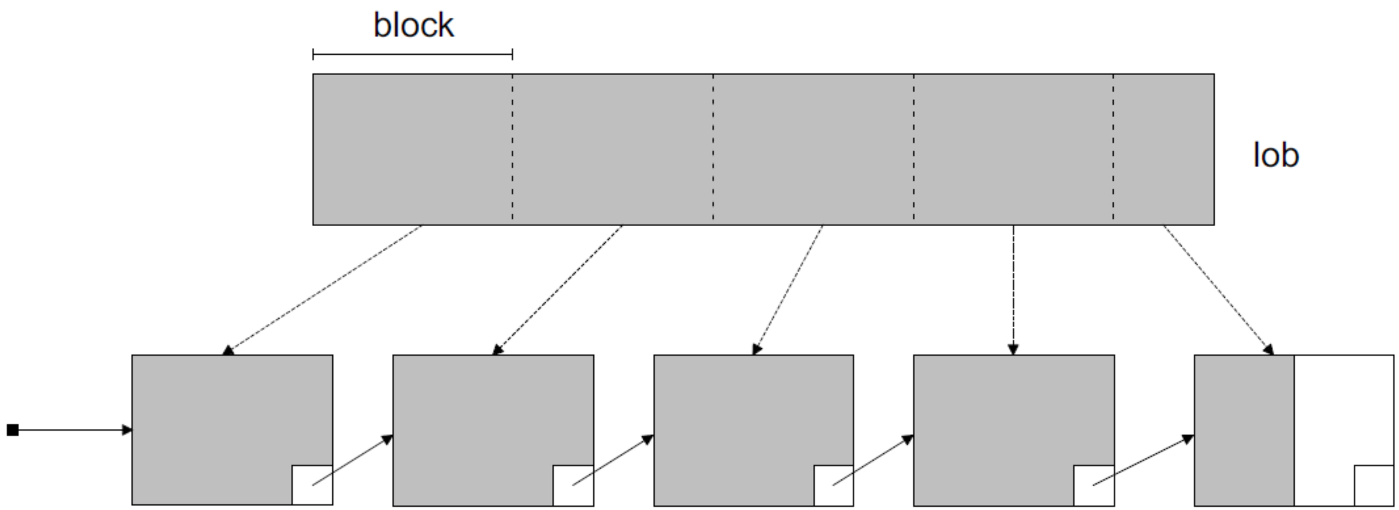
DATABASE RECORD REPRESENTATION (2)

- Variable-size records:
  - *There are alternative representations involving end-of-field characters and/or pointers*
  - *Common approach:*
    - » 1- or 2-byte pointers in the header point to the start/end of each field
    - » Example:
      - *Kunde(kundeId int, fornavn varchar(128), etternavn varchar(128), epost varchar(128), postnummer unsigned zerofilled smallint)*



DATABASE RECORD REPRESENTATION (3)

- Long attribute values (BLOB, CLOB, TEXT, …) may be stored in a separate linked structure



Hammer & Schneider 2001

A NUT TO CRACK

- Another way to store records is to store them column-wise:
  - *Each column in different files*
  - *Advantages?*
  - *Disadvantages?*
- Column-oriented databases are being used for:
  - *Data warehousing*
  - *Data mining*
  - …

HEAPS

- Simplest type
- Records stored in the order in which they were inserted
- New records added at the end of the file
- Suitable when:
  - *bulk loading*
  - *when typical access is a scan through all records*
- Records are marked as deleted when deleted:
  - *Record space is typically not reused*
  - *A periodic reorganisation may be required*

ORDERED FILES

- Records sorted on some key attribute(s):
  - *Guarantees record uniqueness if sorted on the primary key*
- New records added in correct position:
  - *May require a major move of records to create space*
- Suitable when:
  - *records must be returned in some order*
  - *a "range" of records needs to be retrieved*
- Records are reorganized when records are deleted

HASH FILES

- Records inserted in block based on hash value:
  - *By applying a hash function on some field(s)*
- New records added in correct block:
  - *Overflow mechanisms needed on collision:*
    - » open addressing
    - » unchained overflow
    - » chained overflow
    - » multiple hashing
- Suitable when:
  - *retrieving based on equality on the hash fields*
- Records may need to be reorganized when records are deleted

A NUT TO CRACK

- Assume that we need to do a complete sequential scan through all database records
- How will a hash file perform when compared to a heap file:
  - *Better*
  - *Worse*
  - *Similarly*

CONTENT

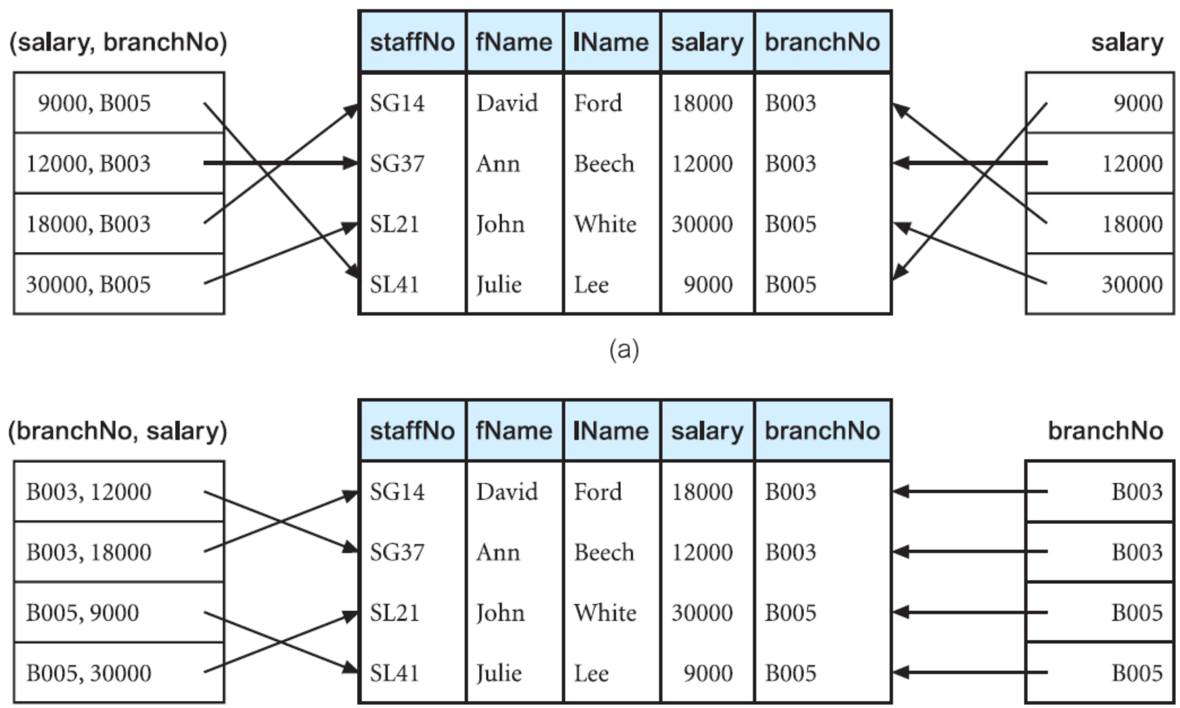
- *Database file organisation and access methods:*
  - » Introduction
  - » Heaps
  - » Ordered files
  - » Hash files
- *Indexes and index file organisation:*
  - » Primary and secondary indexes
  - » B+-trees
  - » Bitmap indexes
- *Managing Indexes in SQL*

INDEXES

- Primary indexes:
  - *The data file is sequentially ordered by the primary key*
  - *Guarantees uniqueness*
  - *Can be only one per file*
- Secondary indexes:
  - *An index defined on a non-ordering field of the data file*
  - *Does not have to be unique*
    - » But may to enforce uniqueness
  - *Can be many per file*
  - *Reduces access time significantly for large tables*

INDEXED SEQUENTIAL FILES (ISAM)

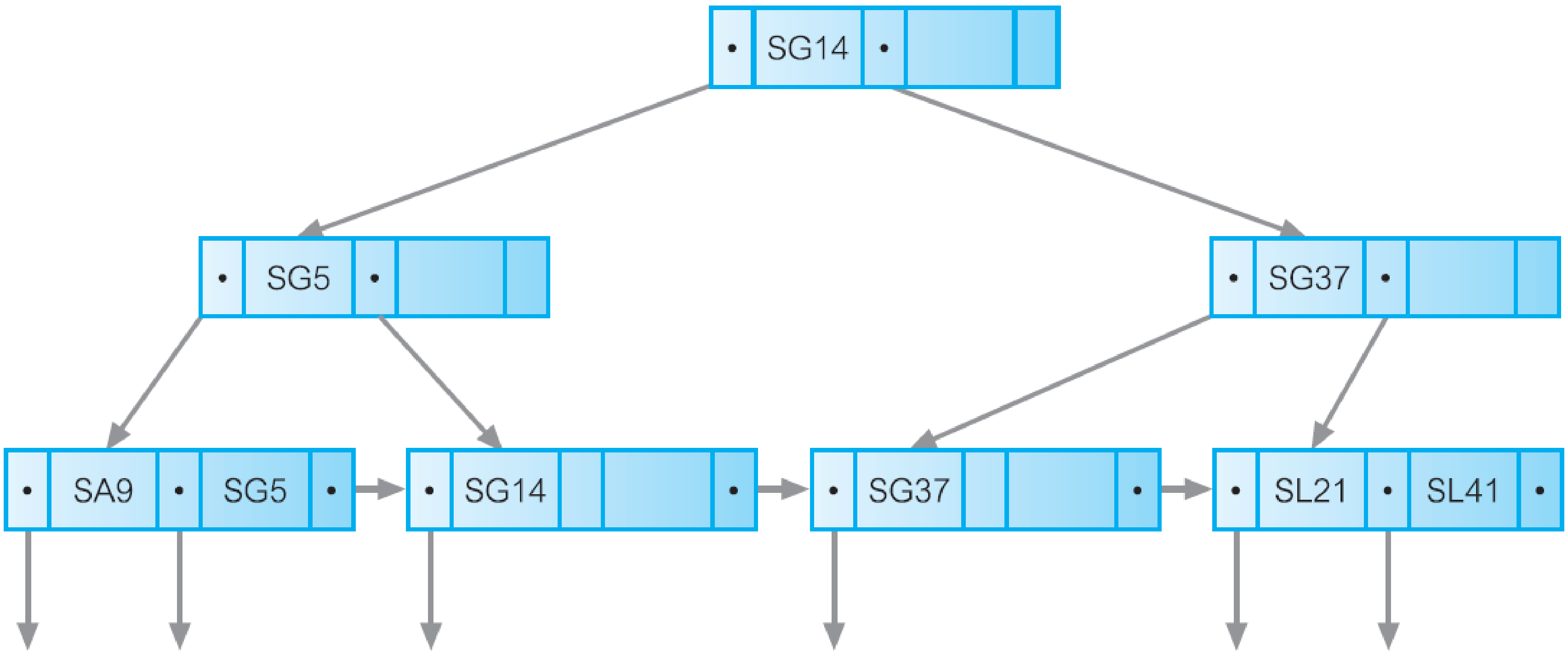
- Data files sorted on primary key
- Separate index or indexes for random access



B+-TREES (1)

- Trees are known to provide efficient search structures
- In databases access times depend on number of disk accesses:
  - *Advantageous to have "bushy", shallow trees:*
    - » Many indexes stored in one block
    - » The three as shallow as possible
- The B+-tree
  - *A **B**alanced tree*

B+-TREES (2)



Pearson Education © 2009

B+-TREES (3)

- B+-tree rules:
  - *The root must have at least two children, unless being a leaf*
  - *Each "interior" node has between* n/2 *and* n *pointers and children*
  - *The tree is always balanced*
  - *Leaf nodes are linked in order of key values*

B+-TREES (4)

- The depth of a B+-tree:
  - *Assume that the field to be indexed is char(8), that pointers are 4 bytes large, and that the block size is 4096 B.*
  - *Size of a block containing* n *indexes:*
    - » *n*(4 + 8) + 4*
  - Number of indexes per block
    - » *n = 1 + (4096 - 4) / (4 + 8) = 342*
  - *Number of records that can be indexed:*
    - » *1 level: 342*
    - » *2 levels: 342 * 341 = 116,622*
    - » *3 levels: 342 * 342 * 341 = 39,884,724*
    - » …
    - » *h levels:* $n^h - n^{h-1}$

BITMAP INDEXES

- Useful for attributes having sparse domains
- Example: Music categories of a CD:

id	title	artist	genre	creationYear
1	Believe - Deluxe Edition (m/DVD)	Justin Bieber	Pop	2012
2	Contakt	Madcon	Hip	2012
3	Live Viking Stadion 9. Juni 2012	Mods	Rck	2012
4	Living Things	Linkin Park	HRk	2012
5	Odyssey - In Studio & In Concert (3CD)	Terje Rypdal	Kls	2012
6	Sexual Harassment - Limited Edition	Turboneger	Pun	2012
7	Slipp mine fløyter fri	Skruk	Kor	1990

Hip	HRk	Kls	Kor	Pop	Pun	Rck
0	0	0	0	1	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	1
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0

CONTENT

- *Database file organisation and access methods:*
  - » Introduction
  - » Heaps
  - » Ordered files
  - » Hash files
- *Indexes and index file organisation:*
  - » Primary and secondary indexes
  - » B+-trees
  - » Bitmap indexes
- *Managing Indexes in SQL*

INDEXES IN SQL

- Creation of indexes is **not** standard in SQL
  - *But all vendors offer a CREATE INDEX statement*
- Indexes may be unique but do not have to be
- MySQL statement syntax:
  - ```
    CREATE [UNIQUE] INDEX index_name
           ON tbl_name (index_col_name,...)
    ```
  - ```
    DROP INDEX index_name ON tbl_name
    ```
- We will cover indexes in more detail on Wednesday

RESOURCES

- C&B 7.3.5, Appendix F