Week 11
Databases – Designing for Performance

LM Data Structures, Algorithms, and Databases (34141)

Dr Ahmad Ibrahim
a.ibrahim@bham.ac.uk
April 26, 2024

Based on lecture notes by Achim Jung & Uday Reddy

Topics by Week

Week	Date	Topic
1	15 Jan	Searching algorithms
2	22 Jan	Binary Search Tree
3	29 Jan	Balancing Trees – AVL Tree
4	5 Feb	Databases – Conceptual Design
5	14 Feb	Databases – Logical Design & Relational Algebra
6	19 Feb	Consolidation Week
7	26 Feb	Complexity analysis, Stacks, Queues, Heaps
8	4 Mar	Sorting Algorithms, Hash tables
9	11 Mar	Graph Algorithms
10	18 Mar	Databases – Normalization
		Easter break and Eid break
11	22 Apr	Databases – Physical Design
12	29 Apr	Assessment Support Week

Past Exam papers and Week 12

Past exam papers are available at:

https://canvas.bham.ac.uk/courses/73245

Week 12 (assessment support week)

LM Data Structures, Algorithms, and Databases (34141)
Date: Monday 29th April
Time: 6:00-8:00pm
Venue: Auditorium

This Week

Designing for Performance

➡ Physical Design

Evaluation of Queries

Pre-Calculating

Splitting

Indexes

Denormalisation

Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Three phases of database design

Conceptual design

Studying the problem domain, and identifying entities and relationships.

Logical design

Deciding tables for the entities and relationships.
Removing redundancies by normalisation.

Physical design

Adjusting the table design for performance, by considering how the data should be stored on disk and optimising the tables.

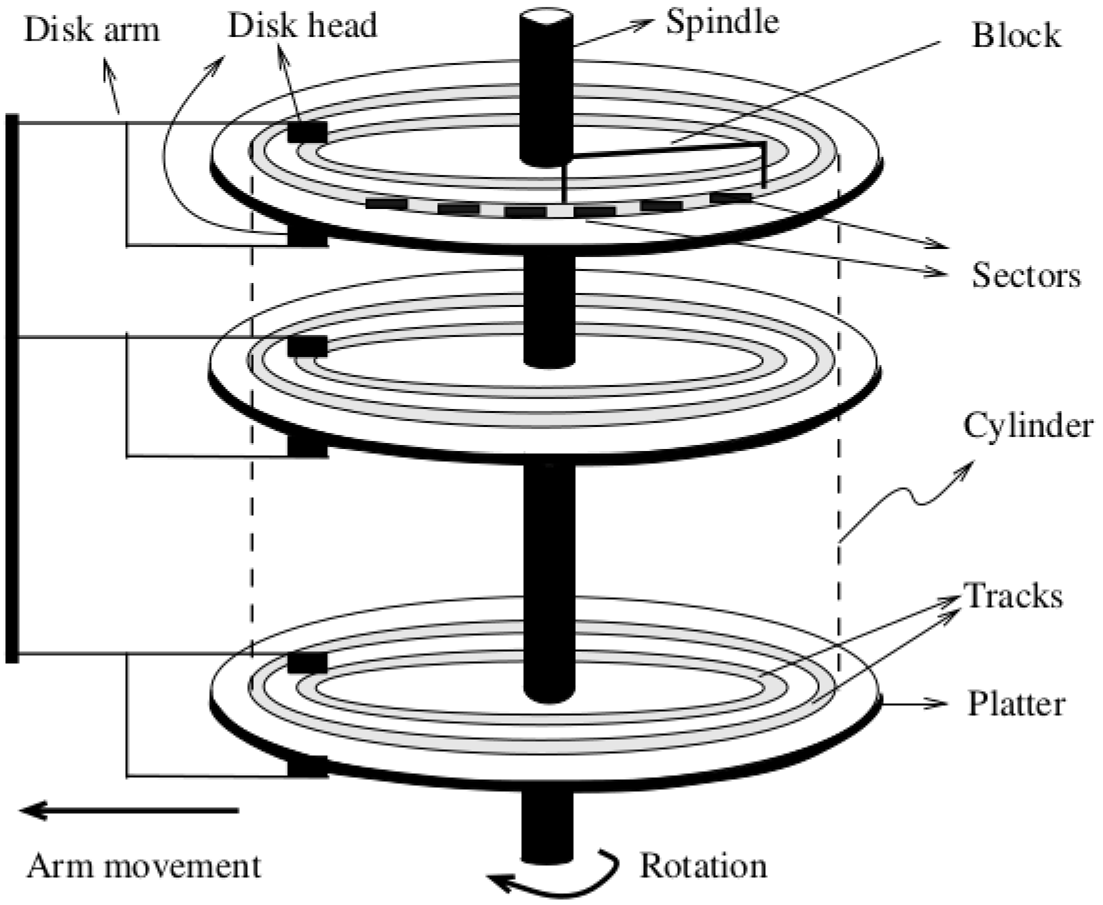
Physical Design – Purpose?

- Whenever we build a software system we have two concerns: *correctness* and *efficiency*.
  - there are a number of ways in which we can enhance efficiency.
  - this is what physical design is all about (aka “database tuning”).
- Implementing a DBMS is a truly major task
  - the objective is to give you enough insight so that you can understand which problems we address during the physical design phase and why certain tuning techniques actually work.

Some Implementation Details

- Database records are stored in *special files* on permanent storage media (such as hard disks).
  - these files are typically not read in a linear fashion.
  - the DBMS loads individual pages from the file as and when needed.
  - Database files are also special in that they are typically *very large*, and do not fit into the main memory as a whole.
- The size of a "page" depends on the underlying architecture but could be anything from 2KB to 32KB.
  - the hard disk controller is built in such a way that it can retrieve and store whole pages somewhat efficiently.
  - however, it is very slow when compared to the speed of processing inside the processor or even the main memory.

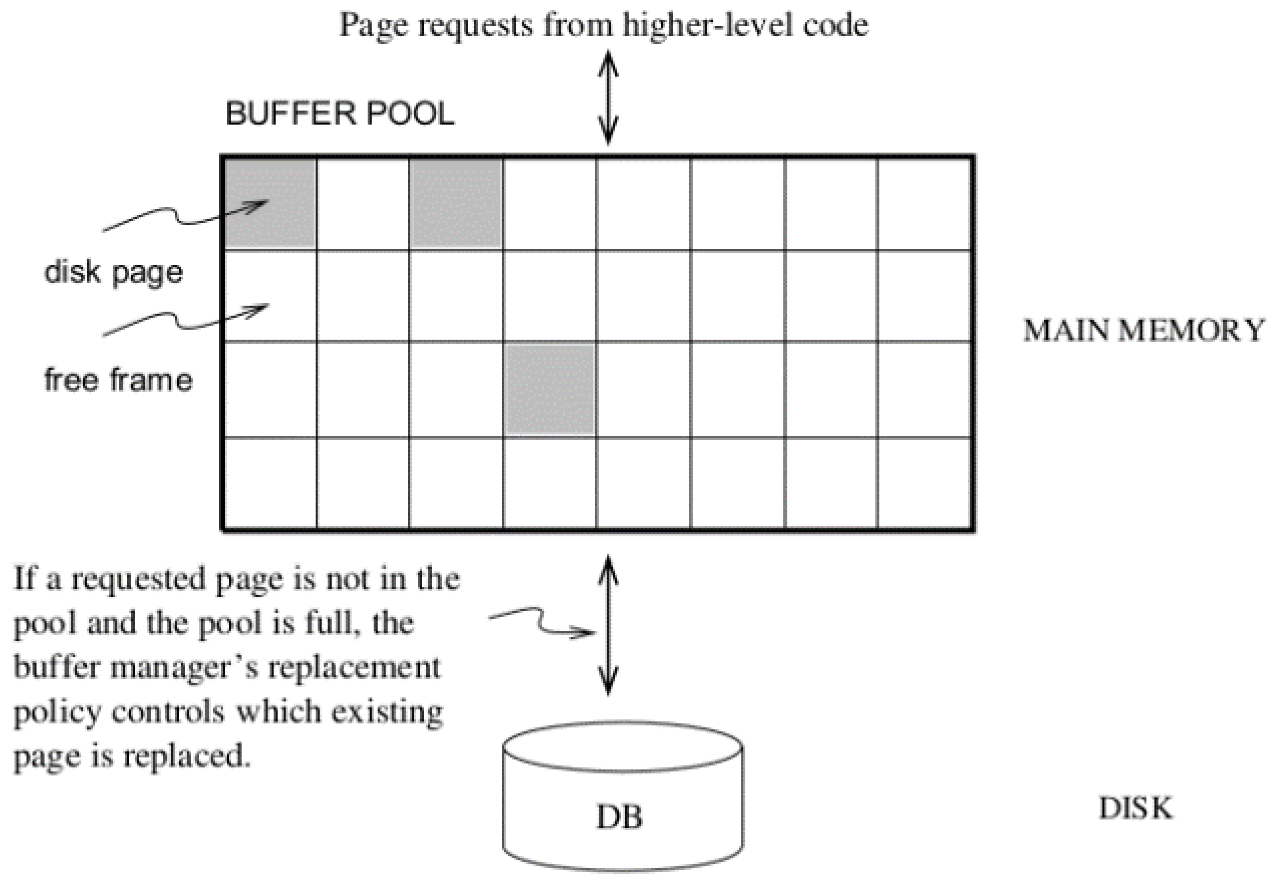
Disk Structure



Typical Rotation Speed: 7200 RPM
1 Rotation Time ~= 10 ms
Typical CPU Processing Time / page ~= 10 us

Some Implementation Details (2)

- Inside main memory, the DBMS maintains a buffer pool which is a collection of buffers the size of one page.
  - Once a page has been fetched and processed, it is not immediately discarded but kept until the DBMS needs the space for some other purpose.
  - This will obviously improve performance but you can not assume that the number of buffers is large enough to hold your entire database.



Page requests from higher-level code
BUFFER POOL
disk page
free frame
MAIN MEMORY
If a requested page is not in the pool and the pool is full, the buffer manager's replacement policy controls which existing page is replaced.
DB
DISK

Some Implementation Details (3)

We draw the following conclusions from this:

a) The number of pages that are needed to store a table should be minimised.

b) An operation which requires the DBMS to look at every record of a large table is expensive.

For more information on how records are stored in a DBMS we recommend Chapter 7: “Storing Data: Disks and Files” of the book by Ramakrishnan and Gehrke.

This Week

Designing for Performance

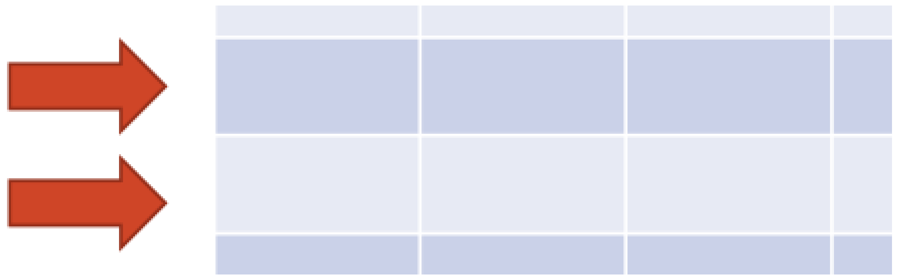
- Physical Design
- ➡ Evaluation of Queries
- Pre-Calculating
- Splitting
- Indexes
- Denormalisation

Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Evaluation of Queries

- We start with the selection operation $\boldsymbol{\sigma}_C$ which picks certain rows and rejects others
  - In general, this requires the system to look at every row to see whether the selection condition C applies or not.
  - If the select condition is very simple, then the system *may know* on which page(s) the relevant records are located.



Evaluation of Queries

- The main operation in DBMSs is the natural join $T_1 \bowtie T_2$
  - Conceptually, it requires the system to compare every row of T_1 with every row of T_2 and to check whether they agree on the common attribute(s).
  - The goal is to compare only those rows where there is likely to be a match and to avoid even attempting as many non-matches as possible.

Students

sID	sName
1	Alice
2	Bob
3	Charlie
4	John

Marks

sID	mID	assignMarks	examMarks
1	1	15	80
2	1	12	81
5	2	18	66
6	2	11	70

Students $\bowtie$ Marks

Students.sID	sName	mID	assignMarks	examMarks
1	Alice	1	15	80
2	Bob	1	12	81

Evaluation of Queries (2)

- However, optimisation can not take place unless the resulting table is itself reasonably small.
- We draw the following conclusions:
  a) Joining tables together is expensive.
  b) Where tables need to be joined it is important that the result table is of a size not exceeding that of T_1 or T_2 by a large factor.
  c) The system may need guidance to be able to find matching rows quickly and to avoid non-matching ones.

This Week

Designing for Performance

- Physical Design
- Evaluation of Queries
- ➡ Pre-Calculating
- Splitting
- Indexes
- Denormalisation

Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Optimisation Technique#1: “**Pre-calculating**” to avoid queries

- Consider a database for a library and suppose that you frequently need to know how many books have been borrowed by some user already (because there is an upper limit on how many books a user may take out).
- This information can be calculated from the current list of borrowings:

```
SELECT COUNT(*)
FROM
current_borrowings
WHERE uid = ’0123456’;
```

Optimisation Technique#1: “**Pre-calculating**” to avoid queries

- Since we are likely to run this query every time a book is taken out, it makes sense to run it once for all users and then to just update it as we go along:

```
CREATE TABLE number_of_borrowed_items(
        uid CHAR(7) PRIMARY KEY,
        no_items INT CHECK(no_items >= 0));

INSERT INTO number_of_borrowed_items
        SELECT uid, COUNT(*)
        FROM current_borrowings
        GROUP BY uid;
```

Optimisation Technique#1:
“**Pre-calculating**” to avoid queries

- Every time a book is taken out we run the two queries:

```
SELECT no_items
   FROM number_of_borrowed_items
   WHERE uid = ’0123456’;

   UPDATE number_of_borrowed_items
   SET no_items = no_items + 1
   WHERE uid = ’0123456’;
```

- These are likely to be faster than the aggregation query we had at the beginning because it touches only a single record of a smaller table.
- It does not makes sense to keep values which are *easy to calculate*!

This Week

Designing for Performance

- Physical Design
- Evaluation of Queries
- Pre-Calculating
- ➡ Splitting
- Indexes
- Denormalisation

Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

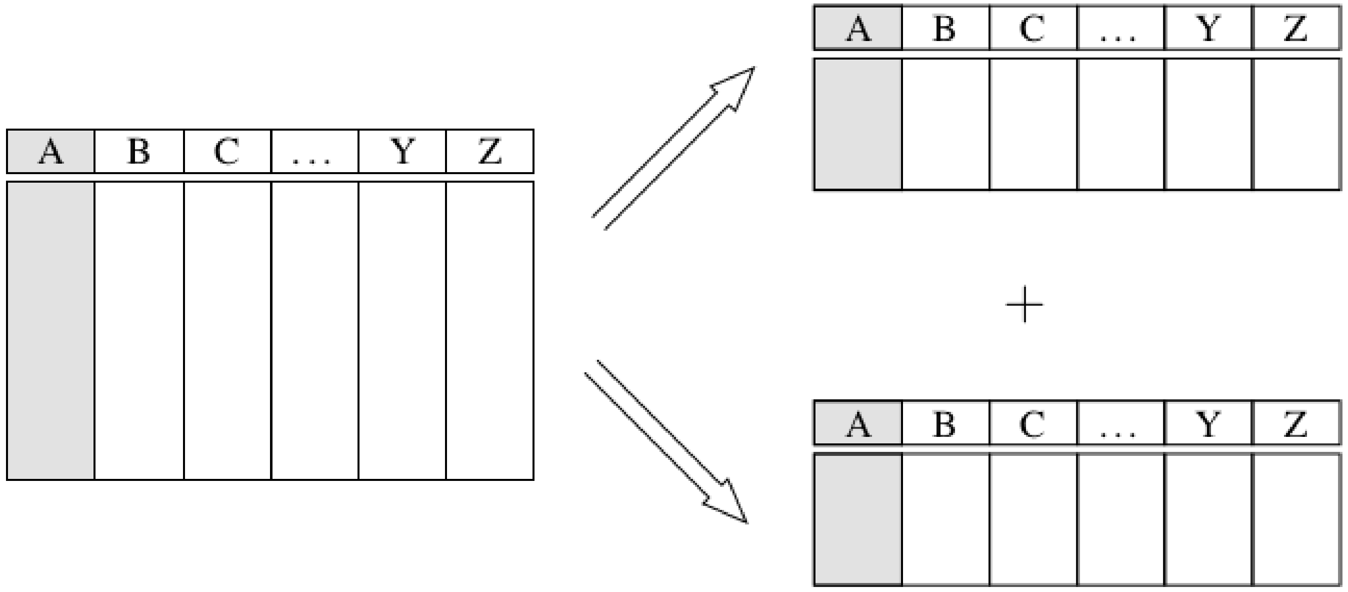
Optimisation Technique#2: “**Splitting**” to keep tables small

- This can be achieved in two ways. **Vertical splitting** refers to separating out subsets of rows into different tables.
- The split should be built on a *semantic principle*:

a) Time: Keep last year’s sales separate from those that started in the current year.

b) Status: Move all completed sales to an archive table.

c) Location: Have separate tables for customers in different countries.



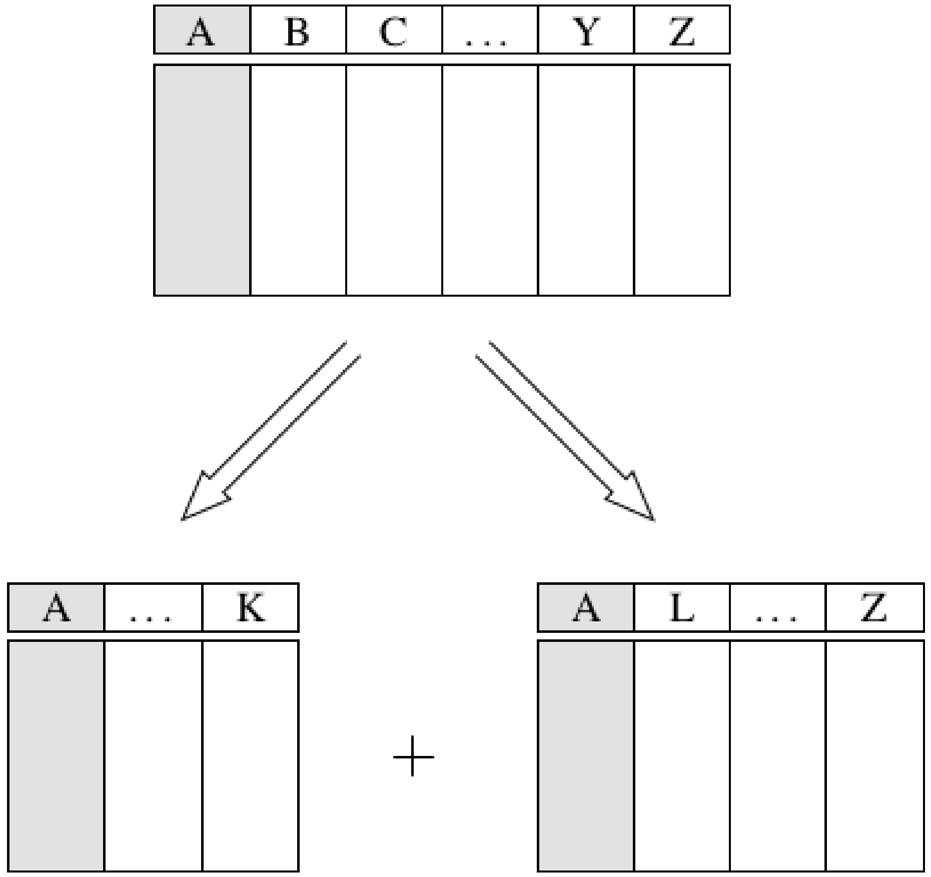
It is important that the split agrees with common queries i.e. they typically only require one to *search one of the tables*.

Optimisation Technique#2: "**Splitting**" to keep tables small

- In **horizontal splitting** we attempt to make a table narrower by separating out some attributes into a different table.

For example:

- The University student table has over 50 attributes. Clearly, very few of these are in frequent use, and information about a student's UCAS code and entry qualifications, for example, will rarely be needed once a student has arrived.
- A use-case analysis is required, to ensure there will be few queries which require the joining together of the two tables.



This Week

Designing for Performance

- Physical Design
- Evaluation of Queries
- Pre-Calculating
- Splitting
- ➡ Indexes
- Denormalisation

Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Optimisation Technique#3: “**Indexes**” to avoid linear search

- Recall that searching for a particular entry in a sorted array is much, much faster than searching in an unsorted file.
  - Binary search takes only 24 steps in an array of 10 million, as compared to linear search that takes 5 million steps on average.
- However, a list of database records can only be kept in order according to one attribute, not several ones.
  - Example: telephone directories: they are sorted alphabetically according to the last name of customers but not according to telephone number.
  - Given a phone number, how can you find the name of customer?

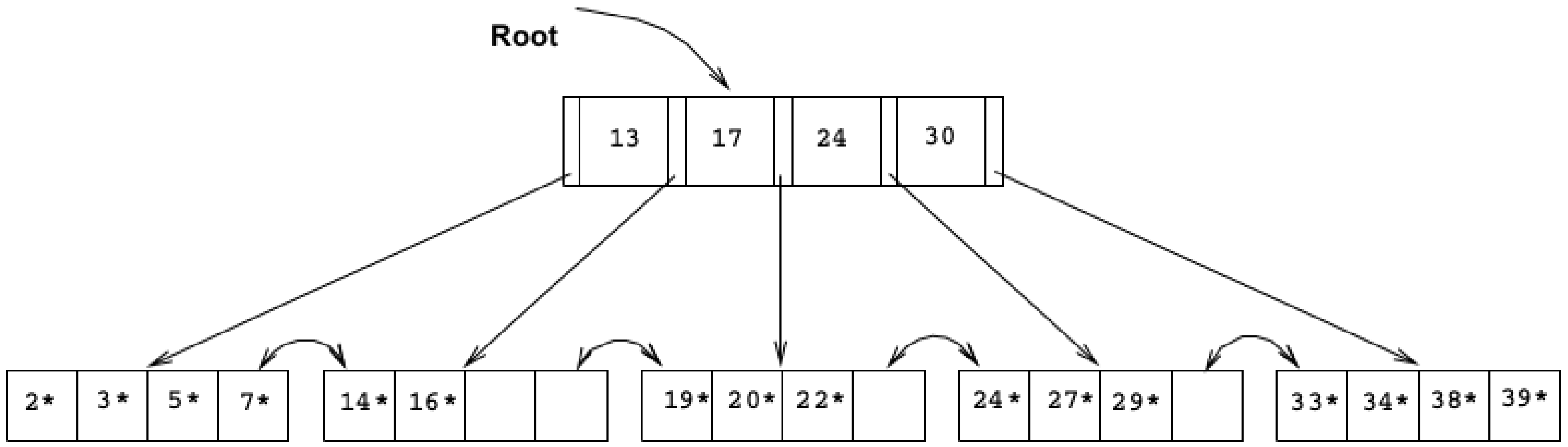
Optimisation Technique#3: “**Indexes**” to avoid linear search (2)

- The way to overcome this restriction is to create an index into the data which allows fast access according to an alternative attribute (or a combination of attributes).
  - The index does not repeat the full information; it just has a pointer to the data in the form of the page number.
  - Database indexes are contained in separate files and only contain the attribute values according to which we want fast access and pointers to the actual database records.
- There are two types of indexes, tree-based and hash-based.
  - The first variety is an optimised form of the binary search idea. The most common data structure used is that of the B+ tree.

Optimisation Technique#3: “**Indexes**” to avoid linear search (3)

- The B+ tree not only makes searching for a particular entry easy but also searching for a range of entries, as in:

```
SELECT firstname, lastname
FROM staff
WHERE office >= 15 AND office < 25;
```



Optimisation Technique#3: “**Indexes**” to avoid linear search (4)

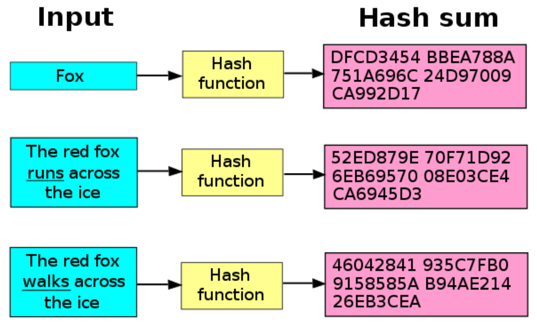
- The syntax for creating an index is not standardised. In PostgreSQL we would say:

```
CREATE INDEX some_name ON staff USING BTREE (office);
```

You will not need the name of the index unless you want to get rid of it again:

```
DROP INDEX some_name;
```

- Please note that by default PostgreSQL builds and maintains a B+ tree index on the primary key of each table.


Input
Hash sum
Fox
Hash function
DFCD3454 BBEA788A 751A696C 24D97009 CA992D17
The red fox runs across the ice
Hash function
52ED879E 70F71D92 6EB69570 08E03CE4 CA6945D3
The red fox walks across the ice
Hash function
46042841 935C7FB0 9158585A B94AE214 26EB3CEA

Optimisation Technique#3: “**Indexes**” to avoid linear search (5)

- The other type of index you can build is based on hashing.
  - A hash function computes a numeric value from some arbitrary input data (usually a string).
  - That number (hashcode) describes the place where more information about the corresponding record can be found.
- Hashing is very fast but does not group similar data together.
  - However, hash functions attempt to spread the input data uniformly across so-called “buckets” and therefore all information about similarity is lost.
  - Therefore a hash index is only helpful for finding individual records (“equality queries”), not for range queries.

This Week

Designing for Performance

- Physical Design
- Evaluation of Queries
- Pre-Calculating
- Splitting
- Indexes
- ➡ Denormalisation

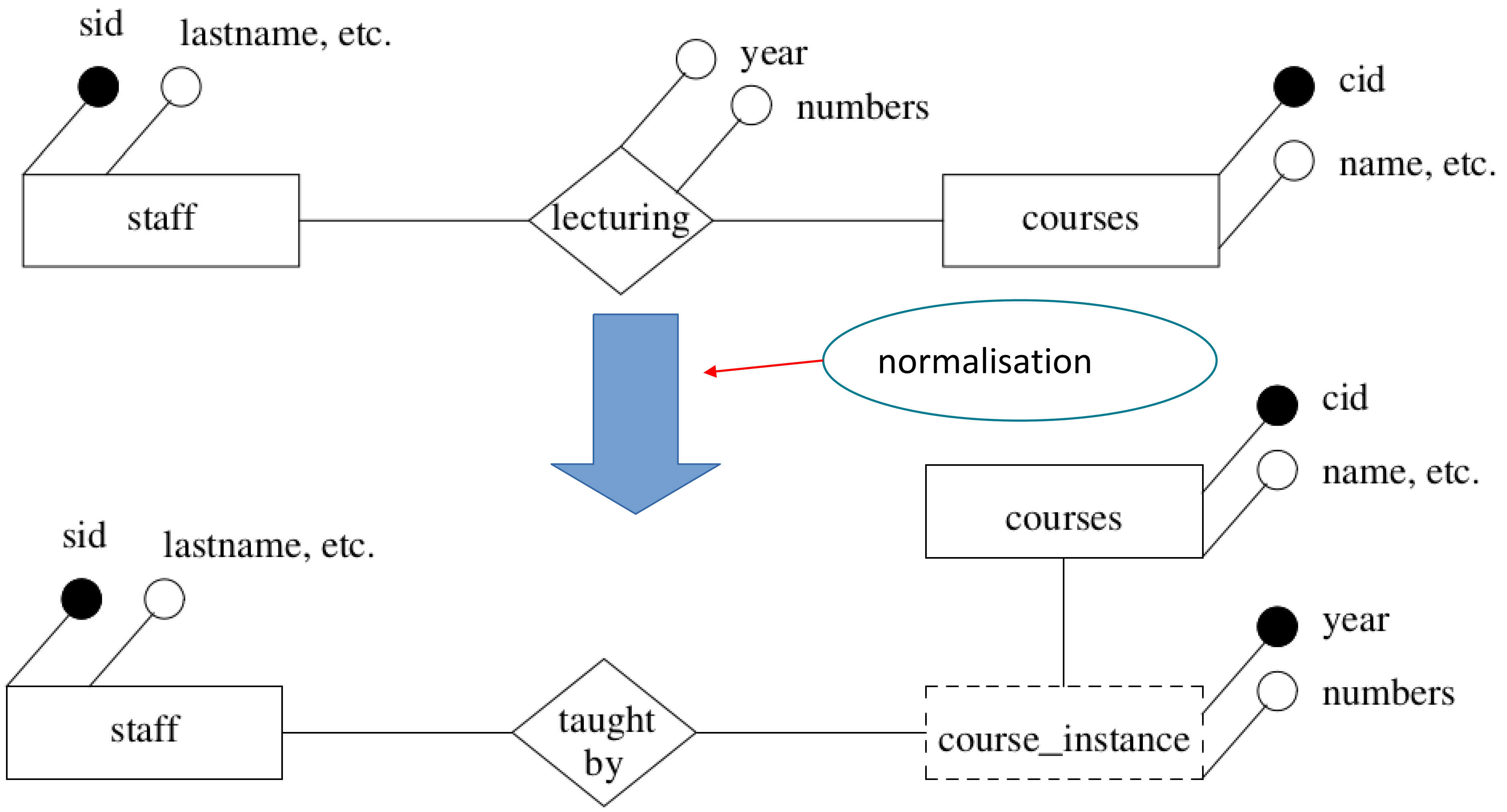
Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Optimisation Technique#4: “**De-normalisation**” to keep information together

- The natural join operation is the most costly one in a database system
  - It is also a common operation!
  - We can try to write queries that avoid natural join operation, but another way to increase efficiency would be to de-normalise tables.
- De-normalisation joins together tables which previous design steps have told us to keep separate in order to avoid redundancy and anomalies.
  - Therefore, we must build safeguards against the anomalies arising from unnormalised tables

Optimisation Technique#4: “**De-normalisation**” to keep information together



Optimisation Technique#4: “**De-normalisation**” to keep information together

- The tables that arise from breaking up “lecturing”, are

  course (cid, bc, name, level, credits)

  course_instance (cid, year, numbers)

  taught_by (cid, year, sid)

- course_instance is a weak entity dependent on “courses”, which has additional information about the course such as “name”, “level”, “semester”, and “bc”.
  - To get the full information about the course being taught, we would often need to do a natural join of “courses” and “course_instances”
  - To avoid this join, we might consider duplicating some of the “courses” information straight into the “course_instances” table, e.g. name.

Optimisation Technique#4:
“**De-normalisation**” to keep information together

- By duplicating the name attribute to course_instance, we will save one natural join every time we seek to link a lecturer with (the name of) a course.
- Note that this would re-introduce redundancy, the name of the course is duplicated in each year.
- We would need to ensure that these copies are kept consistent.

Summary

Let’s review our goals and check which optimisation technique can be used to support them:

- Minimising the number of pages for a table → splitting techniques
- Avoiding linear scans of tables → additional indexes & pre-calculated values
- Avoiding natural joins → de-normalisation, or run separate queries on the two tables and join the results together in the application program.
- Avoiding giant intermediary tables in queries → rewrite queries appropriately and make sure that the joining condition involves a key attribute of at least one of the tables involved.
- Speeding up unavoidable joins → creating additional indexes for the attributes involved in the join operation.

Summary (2)

Keep in mind that there is no optimisation technique which does not incur costs in some other way:

- Keeping additional table with pre-calculated values slows down insertions, deletions, and updates, and in addition takes up storage space.
- Splitting may require the duplication of queries in order to collect all information.
- Additional indexes need to be maintained after insertion and deletion. They also take up space in memory.
- De-normalised tables will contain redundancies which must be controlled by the interface program.

Note: Some of the optimisations conflict with each other e.g. splitting horizontally makes tables narrower whereas de-normalisation makes them wider.

This Week

Designing for Performance

- Physical Design
- Evaluation of Queries
- Pre-Calculating
- Splitting
- Indexes
- Denormalisation

➡ **Week 10 Exercise Sheet** Ex03.pdf

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Question 4: Working with functional dependencies

Consider a schema $T(A, B, C, D, E)$ with the following functional dependencies: $A \rightarrow B$, $BC \rightarrow E$, $ED \rightarrow A$.

a. Find *all* the candidate keys for the schema T.

(Hint: Note that the collection of all attributes $ABCDE$ forms a candidate key. You can progressively remove from this collection one attribute at a time, and determine whether it still remains a candidate key. A *minimal* candidate key is one from which you cannot remove any further attributes. You should attempt to find at least one minimal candidate key during the lab session, and hunt for the remaining ones at home.)

b. Is the table in Boyce-Codd normal form (BCNF)?

c. If it is not, decompose it into BCNF using the normalisation procedure of Section 9.

a.

1. A -> B, => ACDE
   BC -> E, => ABCD
   ED -> A, => BCDE

2. DE -> A, ACDE => CDE
   A ->B, ABCD => ACD
   BC -> E, BCDE => BCD

Candidate key: CDE, ACD, BCD

Question 4: Working with functional dependencies

Consider a schema $T(A, B, C, D, E)$ with the following functional dependencies: $A \rightarrow B$, $BC \rightarrow E$, $ED \rightarrow A$.

a. Find *all* the candidate keys for the schema T.

   (Hint: Note that the collection of all attributes $ABCDE$ forms a candidate key. You can progressively remove from this collection one attribute at a time, and determine whether it still remains a candidate key. A *minimal* candidate key is one from which you cannot remove any further attributes. You should attempt to find at least one minimal candidate key during the lab session, and hunt for the remaining ones at home.)

b. Is the table in Boyce-Codd normal form (BCNF)?

c. If it is not, decompose it into BCNF using the normalisation procedure of Section 9.

Question 4: Working with functional dependencies

Consider a schema $T(A, B, C, D, E)$ with the following functional dependencies: $A \to B$, $BC \to E$, $ED \to A$.

a. Find *all* the candidate keys for the schema T.

   (Hint: Note that the collection of all attributes $ABCDE$ forms a candidate key. You can progressively remove from this collection one attribute at a time, and determine whether it still remains a candidate key. A *minimal* candidate key is one from which you cannot remove any further attributes. You should attempt to find at least one minimal candidate key during the lab session, and hunt for the remaining ones at home.)

b. Is the table in Boyce-Codd normal form (BCNF)?

c. If it is not, decompose it into BCNF using the normalisation procedure of Section 9.

Question 5: Judging decompositions

a. Assume that the functional dependencies $AB \rightarrow C$, $AB \rightarrow D$ and $BC \rightarrow D$, hold for the schema (A, B, C, D). Which set(s) of attributes form a candidate key?

   Consider each of the following decompositions and determine whether it is lossless. If so, say whether it is redundancy reducing and dependency preserving. If all three properties hold determine whether the resulting tables are in Boyce-Codd Normal Form.

   i. (A, B) & (B, C, D)
   ii. (A, B, C) & (B, C, D)
   iii. (A, B, D) & (B, C, D)
   iv. (A, B, C, D) & (B, C, D)

Question 5: Judging decompositions

a. Assume that the functional dependencies $AB \rightarrow C$, $AB \rightarrow D$ and $BC \rightarrow D$, hold for the schema (A, B, C, D). Which set(s) of attributes form a candidate key?

i. [✓ (A, B) & (B, C, D)]

ii. (A, B, C) & (B, C, D)

iii. (A, B, D) & (B, C, D)

iv. (A, B, C, D) & (B, C, D)

Question 5: Judging decompositions

a. Assume that the functional dependencies $AB \rightarrow C$, $AB \rightarrow D$ and $BC \rightarrow D$, hold for the schema (A, B, C, D). Which set(s) of attributes form a candidate key?

i. (A, B) & (B, C, D)

[✓ ii. (A, B, C) & (B, C, D)]

iii. (A, B, D) & (B, C, D)

iv. (A, B, C, D) & (B, C, D)

Question 5: Judging decompositions

a. Assume that the functional dependencies $AB \rightarrow C$, $AB \rightarrow D$ and $BC \rightarrow D$, hold for the schema (A, B, C, D). Which set(s) of attributes form a candidate key?

i. (A, B) & (B, C, D)

ii. (A, B, C) & (B, C, D)

[✓ iii. (A, B, D) & (B, C, D)]

iv. (A, B, C, D) & (B, C, D)

Question 5: Judging decompositions

a. Assume that the functional dependencies $AB \rightarrow C$, $AB \rightarrow D$ and $BC \rightarrow D$, hold for the schema (A, B, C, D). Which set(s) of attributes form a candidate key?

i. (A, B) & (B, C, D)

ii. (A, B, C) & (B, C, D)

iii. (A, B, D) & (B, C, D)

[✓ iv. (A, B, C, D) & (B, C, D)]

Question 5: Judging decompositions

b. Like the previous item: Assume $A \to B$, $A \to C$, $A \to D$, $A \to E$, $B \to C$, and $CD \to E$ hold in schema (A, B, C, D, E), and consider the decompositions

   i. $\boxed{(A, B) \ \& \ (B, C) \ \& \ (C, D, E)}$

   ii. (A, B, D) & (B, C) & (B, D, E)

   iii. (A, B, C, D) & (B, C) & (C, D, E)

Question 5: Judging decompositions

b. Like the previous item: Assume $A \to B$, $A \to C$, $A \to D$, $A \to E$, $B \to C$, and $CD \to E$ hold in schema (A, B, C, D, E), and consider the decompositions

   i. (A, B) & (B, C) & (C, D, E)

   ii. $\boxed{(A, B, D) \text{ \& } (B, C) \text{ \& } (B, D, E)}$

   iii. (A, B, C, D) & (B, C) & (C, D, E)

Question 5: Judging decompositions

b. Like the previous item: Assume $A \to B$, $A \to C$, $A \to D$, $A \to E$, $B \to C$, and $CD \to E$ hold in schema (A, B, C, D, E), and consider the decompositions

   i. (A, B) & (B, C) & (C, D, E)

   ii. (A, B, D) & (B, C) & (B, D, E)

   iii. (A, B, C, D) & (B, C) & (C, D, E)

Question 6: Normalisation example

Consider the following schema for a table that arose in building a database system for aircraft testing in a amall airport. The tests are represented by an 'IATA number' (assigned by the International Air Travel Agency). The maximum score that can be received in the test is known. A particular airplane might receive smaller score than the maximum in an individual test.

test(IATAno, name, date, airplane, technician, duration, score, maxscore)

The attributes satisfy the following functional dependencies:

IATAno	→	name
IATAno	→	maxscore
IATAno, airplane, date	→	technician
IATAno, airplane, date	→	duration
IATAno, airplane, date	→	score

a. Use the normalisation procedure to decompose the table into BCNF.

b. Briefly describe how each of the anomalies mentioned in Section 9 might arise with the original table, but are avoided in the decomposed tables.

Question 6: Normalisation example

Consider the following schema for a table that arose in building a database system for aircraft testing in a amall airport. The tests are represented by an 'IATA number' (assigned by the International Air Travel Agency). The maximum score that can be received in the test is known. A particular airplane might receive smaller score than the maximum in an individual test.

test(IATAno, name, date, airplane, technician, duration, score, maxscore)

The attributes satisfy the following functional dependencies:

IATAno	$\rightarrow$	name
IATAno	$\rightarrow$	maxscore
IATAno, airplane, date	$\rightarrow$	technician
IATAno, airplane, date	$\rightarrow$	duration
IATAno, airplane, date	$\rightarrow$	score

a. Use the normalisation procedure to decompose the table into BCNF.

b. Briefly describe how each of the anomalies mentioned in Section 9 might arise with the original table, but are avoided in the decomposed tables.

Summary

Designing for Performance

- Physical Design
- Evaluation of Queries
- Pre-Calculating
- Splitting
- Indexes
- Denormalisation

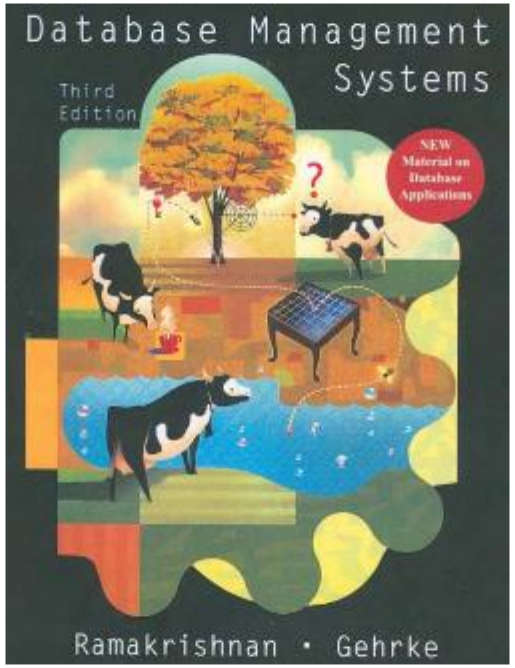
Week 10 Exercise Sheet

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

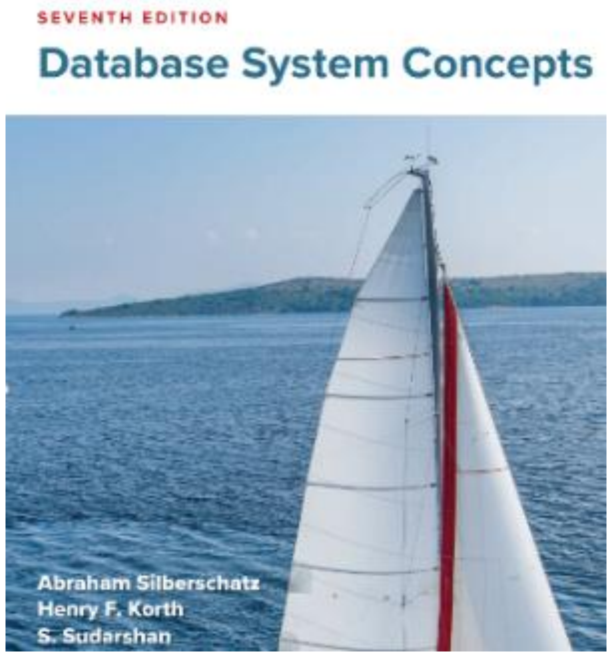
Reading material on Canvas
Handout 6

Weekly Reading

Textbook


Database Management Systems
Third Edition
NEW Material on Database Applications
Ramakrishnan • Gehrke

Chapter 20: “Physical Database Design and Tuning”



Part 5 and Part 6

Next Week

Week	Date	Topic
1	15 Jan	Searching algorithms
2	22 Jan	Binary Search Tree
3	29 Jan	Balancing Trees – AVL Tree
4	5 Feb	Databases – Conceptual Design
5	14 Feb	Databases – Logical Design & Relational Algebra
6	19 Feb	Consolidation Week
7	26 Feb	Complexity analysis, Stacks, Queues, Heaps
8	4 Mar	Sorting Algorithms, Hash tables
9	11 Mar	Graph Algorithms
10	18 Mar	Databases – Normalization
		Easter break and Eid break
11	22 Apr	Databases – Concurrency
12	29 Apr	Assessment Support Week

Thank you.

Questions?

UNIVERSITY OF BIRMINGHAM | DUBAI دبي

Attendance