Implementation of DBMS - Exercise Sheet 4
Instructor: Ali, WS 2024 / 2025

Exercise 1: Field Alignment and Record Size Calculation

1. Suppose a record contains the following fields in order:

- **Employee Name**: A character string of length 25 bytes
- **Employee ID**: An integer of 4 bytes
- **Date of Joining**: An SQL date requiring 10 bytes
- **Salary**: A floating-point number of 8 bytes

Questions:
a) Calculate the total record size if fields can start at **any byte**.
b) Calculate the record size if fields must start at a byte that is a **multiple of 4**.
c) Calculate the record size if fields must start at a byte that is a **multiple of 8**.

Exercise 2: Record Header and Alignment Constraints

2. Assume the fields are as in **Exercise 1**, but the records also include a **header** consisting of:

- **Two 4-byte integers**
- **A 2-byte character field**

Questions:
Calculate the total record size, considering the three alignment constraints:
a) Fields can start at **any byte**.
b) Fields must start at a byte that is a **multiple of 4**.
c) Fields must start at a byte that is a **multiple of 8**.

Exercise 3: Block Storage and Record Packing

3. Assume records are as in **Exercise 2**, and we wish to pack as many records as possible into a **block of 8192 bytes**, using a **block header** consisting of **five 8-byte integers**.

Questions:
a) How many records can fit in the block if fields can start at **any byte**?
b) How many records can fit in the block if fields must start at a **multiple of 4**?
c) How many records can fit in the block if fields must start at a **multiple of 8**?
Assume unspanned storage for records.

Exercise 4: Variable-Length Record Organization

4. A **product inventory record** consists of the following:

- Fixed-length fields: **Product ID (10 bytes)**, **Category ID (10 bytes)**, and **Stock Quantity (4 bytes)**.
- Variable-length fields: **Product Name**, **Description**, and **Manufacturer Details**.

Additional Information:

- Each pointer within the record requires **4 bytes**.
- The record also includes a **4-byte field for the record length**.

Questions:
a) Calculate the total size of the record, excluding the variable-length fields.
b) Propose ways to **optimize internal storage** to reduce the record size.

Exercise 5: Variable-Length Field Size Estimation

5. Suppose records are as described in **Exercise 4**. The variable-length fields have the following size ranges:

- **Product Name**: 20–60 bytes
- **Description**: 50–300 bytes
- **Manufacturer Details**: 0–500 bytes

Questions:
a) Calculate the **average size** of each variable-length field.
b) Determine the **average size** of the total record, including both fixed and variable-length fields.

Learning Objectives:

1. Understand **field alignment** and its impact on memory usage.
2. Calculate **storage efficiency** in different scenarios (fixed vs. variable-length records).
3. Explore **record organization** techniques to optimize storage and reduce overhead.
4. Analyze the trade-offs between **alignment constraints** and **data packing** in database systems.

4a) **A fixed-length record design**:
A student record contains the following fixed-length fields: student ID (10 bytes), enrollment year (4 bytes), and major code (6 bytes). The record also includes pointers to two variable-length fields: name and email. Each pointer is 4 bytes long, and the record includes a 4-byte field for the record length.

- How many bytes are required for the record, excluding the variable-length fields?
- Could the record size be reduced by reorganizing its structure? If yes, explain how.

4b) **Optimization for variable-length fields**:
A company record contains the following fixed-length fields: company ID (12 bytes) and industry code (6 bytes). It also includes pointers to three variable-length fields: company name, headquarters address, and CEO name. Each pointer requires 4 bytes, and the record contains a 4-byte field for the record length.

- Calculate the size of the record without including the variable-length fields.
- Propose a design adjustment to reduce the record size further.

4c) **Variable-length fields and alignment**:
Consider a hospital record that contains the following fields: patient ID (10 bytes), doctor ID (10 bytes), and a field for record length (4 bytes). The record also has two pointers (each 4 bytes) to variable-length fields: diagnosis and prescriptions. If field alignment adds 2 bytes of padding wherever required,

- How many bytes are required for the record, including padding but excluding the variable-length fields?
- How can the record design be optimized to reduce padding while keeping the overall structure efficient?

These questions focus on storage calculations, optimization, and efficient record design in database systems.

Implementation of DBMS
Exercise Sheet 4, WS 2024 / 2025

1) Record Size and Field Alignment with Padding

Suppose a record has the following fields in this order:

- A fixed-length character string of **15 bytes**
- A floating-point number (4 bytes)
- A 2-byte integer
- An SQL date (10 bytes)

Question:
How many bytes does the record occupy under the following alignment conditions?

a) Fields can start at **any byte**.
b) Fields must start at a byte that is a **multiple of 4**.
c) Fields must start at a byte that is a **multiple of 8**.

2) Record with Header and Alignment Constraints

Assume the fields are as in **Exercise 1**, but now the records also include a **header** consisting of:

- Two 4-byte integers
- A single character field (1 byte)

Question:
Calculate the total length of the record, considering field alignment under the three alignment constraints:

a) Any byte
b) Multiple of 4
c) Multiple of 8

3) Packing Records into a Block

Suppose records are as in **Exercise 2**, and we want to pack as many records as possible into a **block of 8192 bytes**. The block includes a **header consisting of five 4-byte integers**.

Question:
How many records can fit into the block for each of the three field alignment scenarios? Assume **unspanned storage** of records.

4) Variable-Length Fields and Record Headers

A **student record** consists of the following **fixed-length fields**:

- Student ID (10 bytes)
- Enrollment number (10 bytes)
- Date of birth (10 bytes)

It also includes **variable-length fields**: Name, Address, Notes

Each **pointer** within the record requires **4 bytes**, and the record has a **4-byte integer** for its length.

Question:
How many bytes, excluding the variable-length fields, are required for the record? Assume no alignment requirements. Additionally, optimize the record layout to further reduce its size.

5) Average Record Size with Variable-Length Fields

Suppose the records are as described in **Exercise 4**, and the **variable-length fields** have the following properties:

- Name: Uniformly distributed length between **15 and 45 bytes**
- Address: Uniformly distributed length between **30 and 90 bytes**
- Notes: Uniformly distributed length between **0 and 2000 bytes**

Question:
What is the **average length** of a student record, including variable-length fields?

---

6) Estimating Records in a Block with Variable-Length Fields

For the **records described in Exercise 5**, assume:

- The **block size** is **4096 bytes**
- The block header contains **eight 4-byte integers**

Question:
How many records, on **average**, can fit into a single block? Assume that variable-length fields are stored **inside the block** and that records are stored **unspanned**.

record layout, field alignment, block packing, and space calculation in DBMS implementation. I'll teach the concepts first, then prepare **advanced exam-style questions**.

🧠 Key Concepts from Sheet 03

1️⃣ Field Alignment & Record Length

- **Why alignment matters:**
  Many CPUs require certain data types to start at specific memory addresses (multiples of 2, 4, or 8). This improves access speed.
- **Without alignment:**
  Fields start immediately after the previous field.
- **With alignment:**
  We may insert **padding bytes** so the field starts at the required multiple.

Example:

- Field = `int` (4 bytes)
- If current offset = 3, we insert 1 padding byte so offset becomes multiple of 4.

2️⃣ Record Header

- A record header stores metadata about the record:
  - Record length
  - Timestamps
  - Pointers to variable-length fields
  - Flags (deleted/active)
- Header size depends on the number and type of metadata fields.

3️⃣ Block Structure & Unspanned Records

- **Block:** Smallest unit of data transfer between disk and memory (e.g., 4096 bytes).
- **Block header:** Stores metadata for the block:
  - Free space info
  - Page ID
  - Transaction/LSN info
- **Unspanned storage:**
  A record must fit entirely inside one block (cannot cross block boundaries).

4️⃣ Variable-Length Fields

- Examples: Name, Address, History.
- Stored as:
  - **Pointer (offset + length)** to actual data (stored elsewhere)
  - Or **inline** if short enough
- **Pointer size** adds to fixed-length space.

5️⃣ Expected Value for Variable-Length Fields

If field length is uniformly distributed between `min` and `max`:

Average length=min+max2\text{Average length} = \frac{\text{min} + \text{max}}{2}

📊 Example Walkthrough (Exercise 1a)

Fields:

- Char(19) → 19 bytes
- Int(2) → 2 bytes
- Date (SQL) → 10 bytes
- Time (SQL) → 8 bytes

No alignment:
Total = 19+2+10+8=3919 + 2 + 10 + 8 = 39 bytes.

Alignment multiple of 4:

- Char(19) starts at byte 0 → offset after = 19 → pad 1 → next offset 20
- Int(2) starts at 20 → after = 22 → pad 2 → next offset 24
- Date(10) starts at 24 → after = 34 → pad 2 → next offset 36
- Time(8) starts at 36 → after = 44 → **final size = 44**

Alignment multiple of 8 would require even more padding.

🎯 Advanced Exam-Level Questions

Q1 — Complex Alignment

You have a record with:

- Header: 2-byte short, 1-byte char, 8-byte timestamp
- Fields: Char(15), Double(8), Int(4), Char(3)
- Compute record length with:
    1. No alignment
    2. Alignment multiple of 4
    3. Alignment multiple of 8

Q2 — Block Packing with Mixed Records

A block is 8192 bytes with a 64-byte header.
You store:

- 60% records of 120 bytes
- 40% records of 200 bytes
  Alignment = multiple of 8.
  How many total records fit per block? How much free space remains?

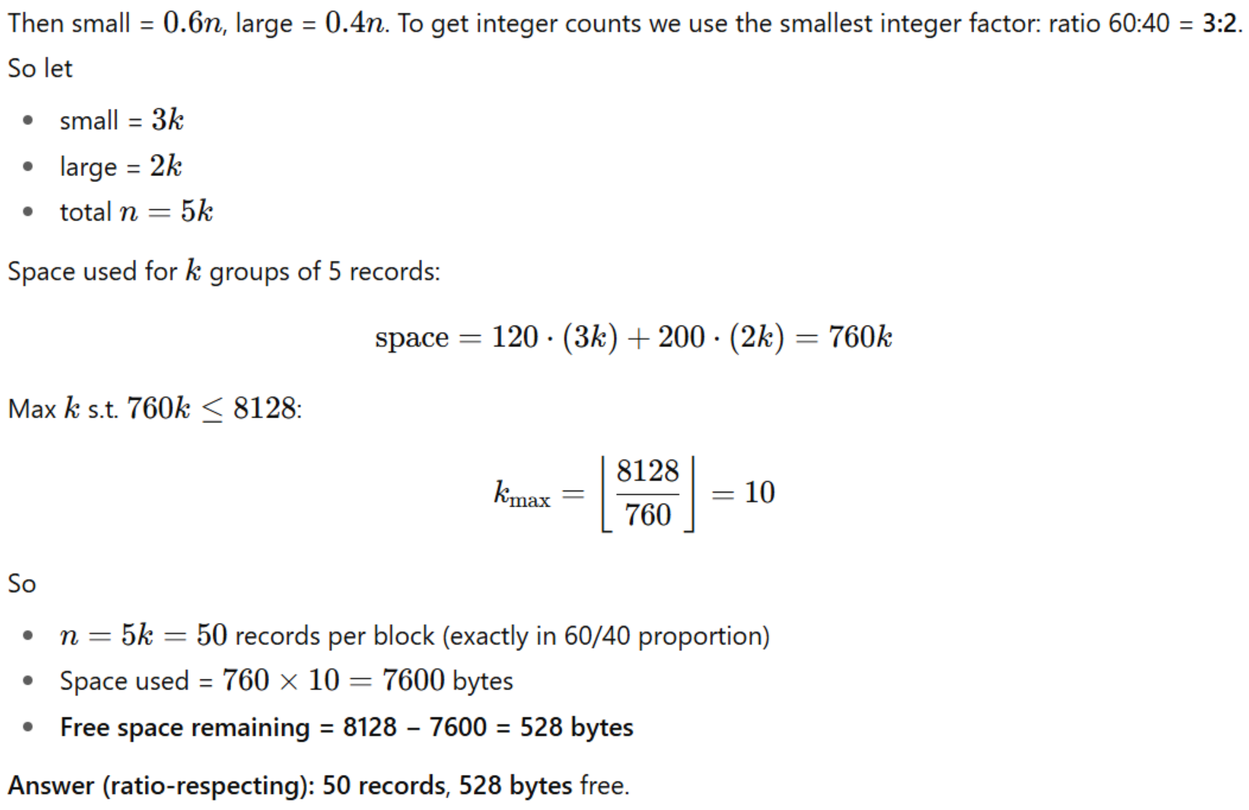
Nice — let's solve Q2 carefully and show a couple of useful variants so you fully understand the packing trade-offs.

Given

- Block size = **8192 bytes**
- Block header = **64 bytes** → usable payload per block = **8192 − 64 = 8128 bytes**
- Two record types:
  - **Small** = 120 bytes (60% of records)
  - **Large** = 200 bytes (40% of records)
- Alignment = **multiple of 8 bytes**
  → 120 and 200 are already multiples of 8, so **no padding** is required.

A — If you must keep the 60% / 40% ratio exactly (i.e. pack records in the same proportion)

Let the total number of records in the block be n.



Then small = $0.6n$, large = $0.4n$. To get integer counts we use the smallest integer factor: ratio 60:40 = **3:2**.

So let

- small = $3k$
- large = $2k$
- total $n = 5k$

Space used for k groups of 5 records:

$$\text{space} = 120 \cdot (3k) + 200 \cdot (2k) = 760k$$

Max k s.t. $760k \le 8128$:

$$k_{\text{max}} = \left\lfloor \frac{8128}{760} \right\rfloor = 10$$

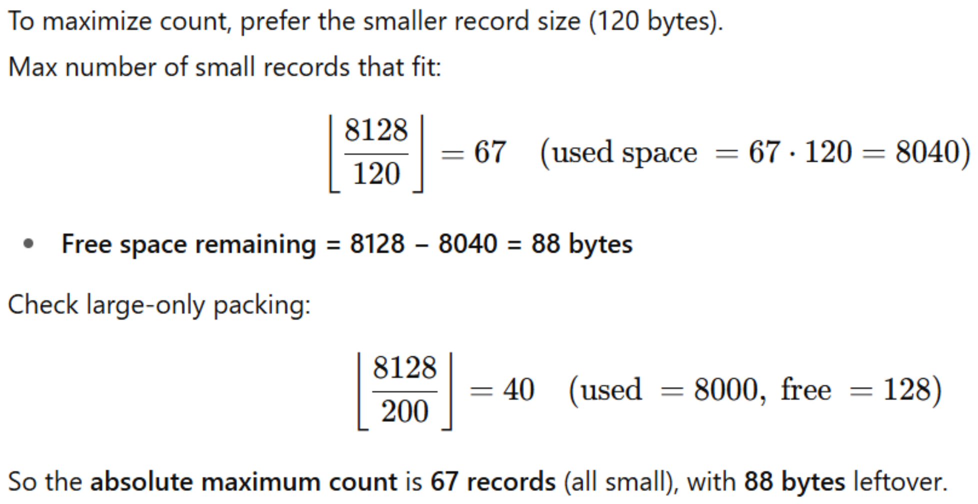
So

- $n = 5k = 50$ records per block (exactly in 60/40 proportion)
- Space used = $760 \times 10 = 7600$ bytes
- **Free space remaining = 8128 − 7600 = 528 bytes**

Answer (ratio-respecting): 50 records, 528 bytes free.

B — If you only care about maximizing the total number of records (no ratio constraint)

To maximize count, prefer the smaller record size (120 bytes).
Max number of small records that fit:



To maximize count, prefer the smaller record size (120 bytes).

Max number of small records that fit:

$$\left\lfloor \frac{8128}{120} \right\rfloor = 67 \quad (\text{used space } = 67 \cdot 120 = 8040)$$

- **Free space remaining = 8128 − 8040 = 88 bytes**

Check large-only packing:

$$\left\lfloor \frac{8128}{200} \right\rfloor = 40 \quad (\text{used } = 8000, \text{ free } = 128)$$

So the **absolute maximum count** is **67 records** (all small), with **88 bytes** leftover.

C — If you want to maximize used space (minimize slack) while keeping 60/40 roughly

We already have the exact ratio solution (50 recs, 528 free). If you allow small deviations from 60/40 you might pack 51 or 52 records with different mixes, but usually the exact-ratio solution is simpler and predictable.

Quick summary

- Respect 60%/40% exactly → **50 records**, **528 bytes** free.
- Maximize count (ignore ratio) → **67 records** (all small), **88 bytes** free.
- Large-only → **40 records**, **128 bytes** free.

Q3 — Variable-Length Average with Alignment

A record has:

- Fixed fields: ID(4 bytes), Status(1 byte), Flags(2 bytes)
- Variable-length fields: Bio(50–150 bytes), Notes(0–300 bytes)
  Header stores record length (4 bytes) and 2 pointers (4 bytes each).
  Assume alignment to multiple of 4.
  Find the **average record size**.

Q4 — Storage Utilization Optimization

A 4096-byte block stores unspanned records with 20-byte headers and variable-length fields stored inline.
Average record size = 130 bytes, but distribution is skewed (10% are 400 bytes).
Devise an **improved block organization** to maximize utilization, and calculate the new utilization %.

Q5 — Trade-Off Question

Explain how **alignment** improves CPU access but can reduce **storage utilization**, and give **three DBMS design strategies** to minimize wasted space while keeping performance.
