Datenbanksysteme
SS 2025

Übungsblatt 4

Group:
Attila Haraszti (5602072),
Moritz Ruge (5600961),
Joshua Zahn (5600969).
John Jesus Llerena Kutscheid

Datum: Jun 16, 2025

Task 1: Indexing (25 %)

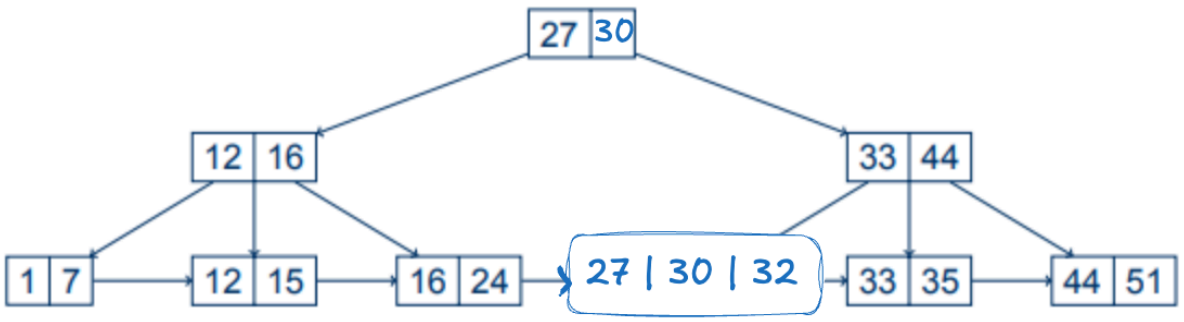
1 EN: Describe how a B+-Tree maintains balance, and why this is important in terms of performance.
DE: Beschreibe, wie ein B+-Baum das Gleichgewicht hält und warum dies für die Leistung wichtig ist.

A B^+-Tree is a special case of (a,b)-Trees with b = 2a - 1, which is a stricter requirement than the typical (a,b)-Tree with b ≥ 2a - 1. The parameters (a,b) define the least and most number of child nodes inner nodes (excluding the root) can have. B-Trees maintain perfect balance by keeping all leaves on the same height level, splitting and merging entries between sibling and parent nodes as needed, if overflow or underflow occurs. The $^+$ sign refers to the fact that only keys are stored at the inner nodes, with the leaf nodes forming an extra bottom layer containing the values, being linked together as a doubly linked list for in-order traversal.

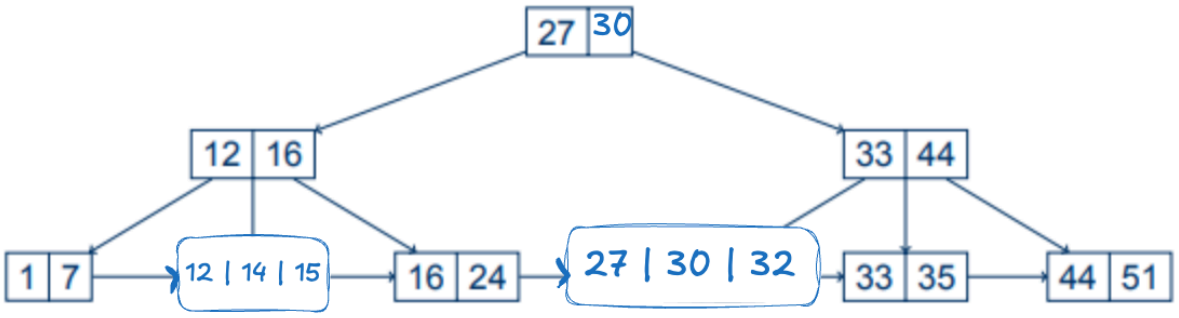
In database systems, B^+-trees indices are an alternative to indexed-sequential files.

“The idea behind a B-tree is to collapse several levels of a binary search tree into a single large node, so that we can make the equivalent of several search steps before another disk access is needed. B-trees can access enormous numbers of keys using only a few disk accesses." (Skiena, 2020, #443)

2 EN: Consider the B+-Tree, with n = 3 below. Insert value 30 into the B+-Tree. Draw the complete result tree.
DE: Betrachte den B+-Baum, mit n = 3 unten. Füge den Wert 30 in den B+-Baum ein. Zeichne den vollständigen Ergebnisbaum.



3 **EN:** Consider the resulting B+-Tree from subtask 2. Insert value 14 into the B+-Tree. After you have inserted the value, draw the complete result tree.
DE: Ausgehend vom resultierenden B+-Baum aus Teilaufgabe 2, füge den Wert 14 in den B+- Baum ein. Zeichne, nachdem der Wert eingefügt ist, den vollständigen Ergebnisbaum.



Task 2: Transactions (35 %)

$S_1 = \langle r_4(x), w_2(y), w_3(z), w_1(z), r_1(x), w_4(x), w_3(x), r_1(x), w_2(z), w_4(z), w_3(y) \rangle$

$S_2 = \langle w_1(a), w_2(b), w_3(c), w_4(d), r_4(d), r_2(c), r_3(a), r_4(c), w_2(d), r_2(a), r_2(d) \rangle$

1 EN: Explain what the ACID properties mean in the context of transactions.
DE: Erklären Sie was die ACID Eigenschaften im Kontext von Transaktionen bedeuten.

Die ACID Eigenschaften, die dafür sorgen sollen, dass die Datenintegrität einer Datenbank nach einer Transaktion immer noch gewährleistet wird.

Atomicity: Es werden entweder alle oder gar keine Operationen einer Transaktion durchgeführt.

Consistency: Wenn bestimmte Regeln vor der Ausführung eingehalten wurden, müssen diese auch nach Ausführung der Transaktion gelten.

Isolation: Wenn zwei Transaktionen gleichzeitig ausgeführt werden, beeinflussen sich die beiden Transaktionen nicht gegenseitig durch zum Beispiel Zwischenergebnisse.

Durability: Eine Transaktion erfolgreich ausgeführt, müssen die Änderungen, die diese vorgenommen haben, für immer gelten.

2 EN: Determine all conflict pairs of S_1 and S_2.
DE: Bestimmen Sie jeweils alle Konfliktpaare von S_1 und S_2.

Konfliktpaare S1:

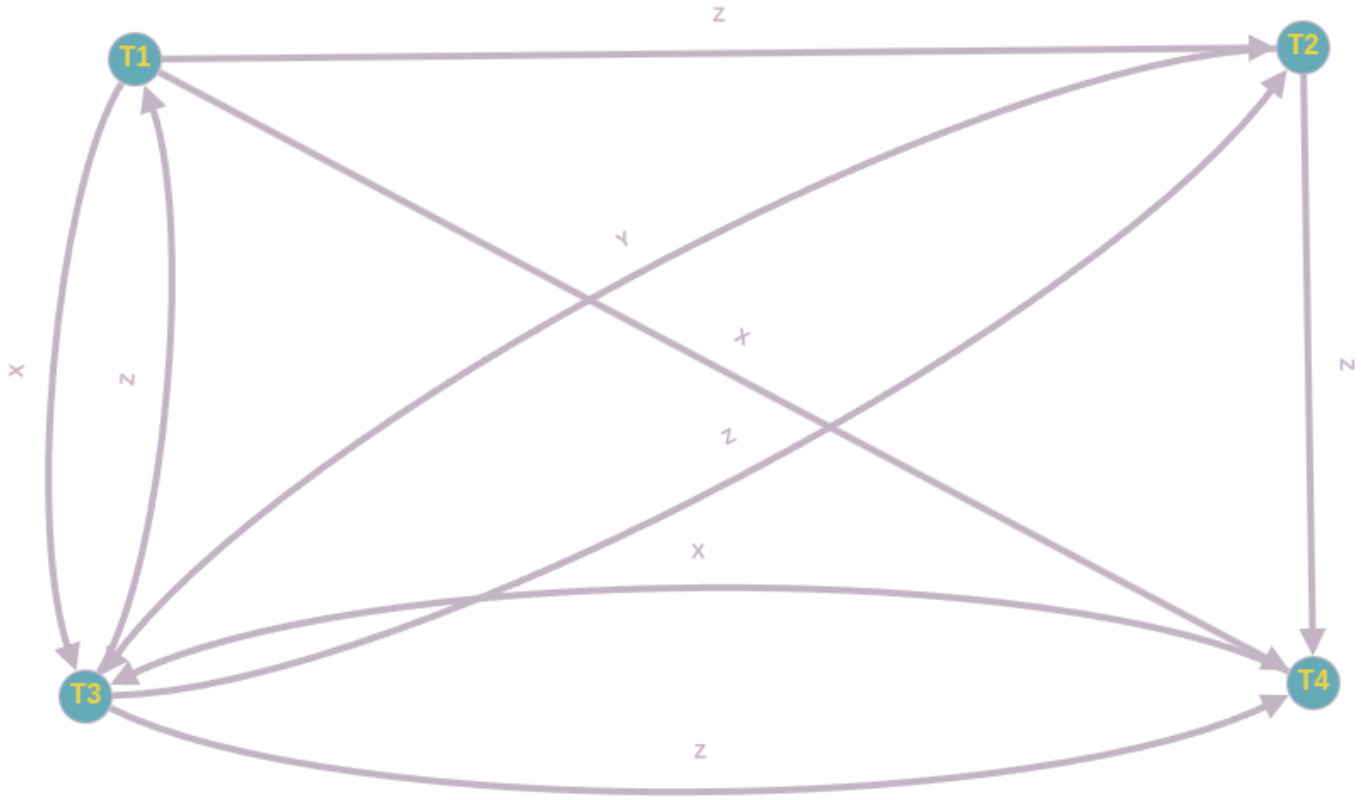
- (r4(x), w3(x))
- (r1(x), w4(x))
- (r1(x), w3(x))
- (w3(x), w4(x))
- (w2(y), w3(y))
- (w3(z), w1(z))
- (w3(z), w2(z))
- (w3(z), w4(z))
- (w1(z), w2(z))
- (w1(z), w4(z))
- (w2(z), w4(z))

Konfliktpaare S2:

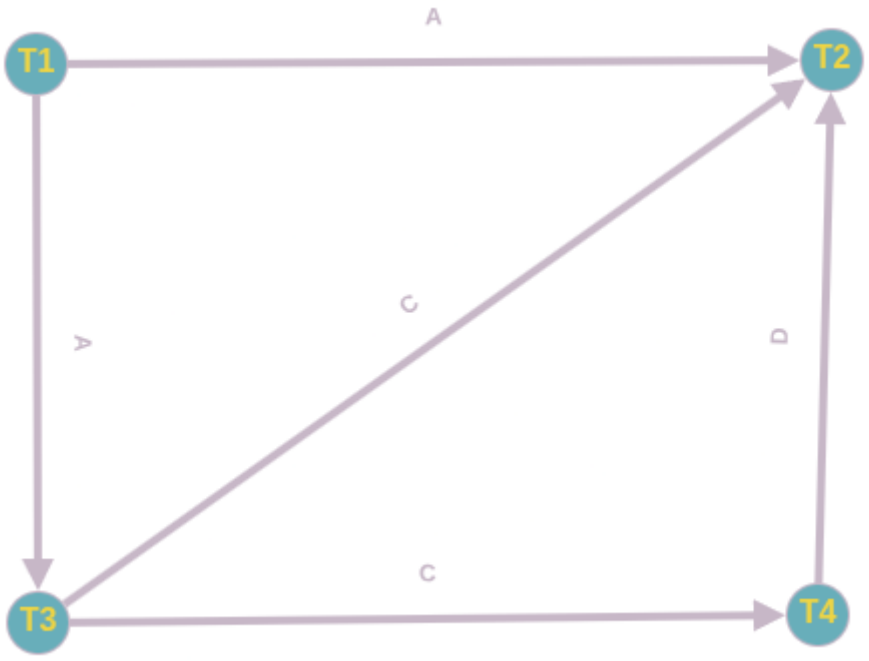
- (w1(a), r3(a))
- (w1(a), r2(a))
- (w3(c), r2(c))
- (w3(c), r4(c))
- (w4(d), w2(d))
- (w4(d), r2(d))
- (r4(d), w2(d))

3 EN: Draw the precedence graph of S_1 and S_2. Label each edge with the data object (resource) on which the conflict is based.
DE: Zeichnen Sie jeweils den Konfliktgraphen von S_1 und S_2. Beschriften Sie jede Kante mit dem Datenobjekt (Ressource), auf dem der Konflikt beruht.

S1:



S2:



4 **EN:** Is S_1 serializable? If yes, give a serial schedule equivalent to S_1. If not, why not? What if we just consider z?
DE: Ist S_1 serialisierbar? Falls ja, geben Sie eine mögliche Serialisierung von S_1 an. Falls nein, warum nicht? Was passiert, wenn nur z betrachtet wird?

S1 ist nicht serialisierbar, da es mehrere Kreise im Konfliktgraphen von S1 gibt (z.B.: T1 -> T3 -> T1). Wenn nur das Datenobjekt z betrachtet wird, ist S1 serialisierbar, da es dann keine Kreise mehr im Konfliktgraphen geben würde. Eine mögliche Serialisierung würde folgendermaßen aussehen: T3 -> T1 -> T2 -> T4.

5 **EN:** Is S_2 serializable? If yes, give a serial schedule equivalent to S_2.
DE: Ist S_2 serialisierbar? Falls ja, geben Sie eine mögliche Serialisierung von S_2 an.

S2 ist serialisierbar, denn es gibt keine Kreise im konfliktgraphen von S2. Eine mögliche Serialisierung würde folgendermaßen aussehen: T1 -> T3 -> T4 -> T2

Task 3: Concurrency Control (30 %)

1 EN: Explain the concept of concurrency control in a database system. Give an example why it is important.
DE: Erklären Sie das Konzept der Concurrency Control (Nebenläufigkeitskontrolle) in einem Datenbanksystem. Beschreiben Sie anhand eines Beispiels, warum dies wichtig ist.

Concurrency control techniques are used to ensure the noninterference or isolation property of concurrently executing transactions. (Elmasri & Navathe, 2016, #781) Without concurrency control, various scenarios similar to memory-access hazards (read-after-write aka. dirty reads, write-after-read aka. inconsistent views or non-repeatable reads, write-after-write aka. lost updates) could happen.

An additional scenario is referred to as phantoms. Phantoms involve new or missing rows in a query that appear before the transaction is finalized. A record r is called a phantom record when it was not there when T_1 starts but is there when T_1 ends. (Elmasri & Navathe, 2016, #774)

2 EN: Explain what a lock table is.
DE: Erklären Sie, was ein Lock Table ist.

A system lock table tracks all locks that are held on database objects, its type (shared or exclusive), owner (transaction) and their state. The granularity (record, field, block, file, entire database) varies by implementation. Items not in the lock table are considered to be unlocked. It is used to detect deadlocks and handle scheduling queues.

3 EN: Explain how deadlocks occur in a database system and why they present a problem.
DE: Erklären Sie, wie Deadlocks in einem Datenbanksystem entstehen und warum sie ein Problem darstellen.

A deadlock occurs where a transaction sets a lock and never releases it, for example due to an error or due to an infinite wait for acquiring another lock that was not released by a different transaction.

4 EN: Give three techniques to control deadlocks and give a description for each. **DE:** Nennen Sie drei Techniken zur Kontrolle von Deadlocks und beschreiben Sie jede davon.

- **Avoidance**
  - A transaction T requesting a new lock is aborted if there is a possibility of deadlock. The transaction is rolled back and rescheduled.

- **Prevention**
  - The transaction must obtain all the locks it needs before it can be executed. This technique avoids rollbacks, but limits concurrency.

  - Some of the techniques use transaction timestamps, preferring older transactions (Elmasri & Navathe, 2016, #790):
    - Wait-die: *non-preemptive*, if an older transaction requests a lock held by a younger one, it *waits*. If a younger transaction requests a lock held by an older one, it aborts (*dies*) and restarts with its original timestamp after a random delay.
    - Wound-wait: *preemptive*, if an older transaction requests a lock held by a younger one, the younger is aborted and restarted with its original timestamp after a random delay. If a younger transaction requests a lock held by an older one, it waits.

  - Some of them do not require timestamps:
    - No waiting: A transaction immediately aborts if its requested lock is unavailable (it rolls back and restarts). This technique leads to increased transaction aborts.
    - Cautious waiting: Proposed as a solution to the increased aborts observed with no waiting. Here a transaction waits for a lock if no other transactions requesting relevant locks are queued afterwards, but aborts and restarts otherwise. Cautious waiting is deadlock-free, because no transaction will ever wait for another blocked transaction, there can be no cycles in the wait-for graph.

- **Detection**
  - The database management system periodically tests the database for deadlock conditions. If one is found, a “victim” transaction is selected and then aborted (rolled back + restarted). The other transactions continue.

Task 4: 2-Phase Locking (10 %)

1 EN: Can deadlocks occur with strict 2-phase locking? If yes, how, and give an example using the table below. If not, why not?
DE: Können Deadlocks bei strengem (strict) 2-Phase-Locking auftreten? Wenn ja, wie, und gebe ein Beispiel unter Verwendung der unten stehenden Tabelle. Wenn nicht, warum nicht?

Strict 2-Phase Locking guarantees serializability, but does not prevent deadlocks. As transactions hold locks until they commit, several transactions can each wait for a lock held by the other.

2 EN: Can deadlocks occur with conservative 2-phase locking? If yes, how, and give an example. If not, why not?
DE: Können Deadlocks bei konservativem 2-Phase-Locking auftreten? Wenn ja, wie, und gebe ein Beispiel an. Wenn nicht, warum nicht?

No, **Conservative 2-Phase Locking** is the only variation that guarantees deadlock-free transactions, because it does not allow a transaction to start before acquiring all the locks it needs. This requires predeclaration and results in a serial execution schedule without any benefits from concurrency.

References

Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems*. Pearson.

Skiena, S. S. (2020). *The Algorithm Design Manual*. Springer International Publishing.