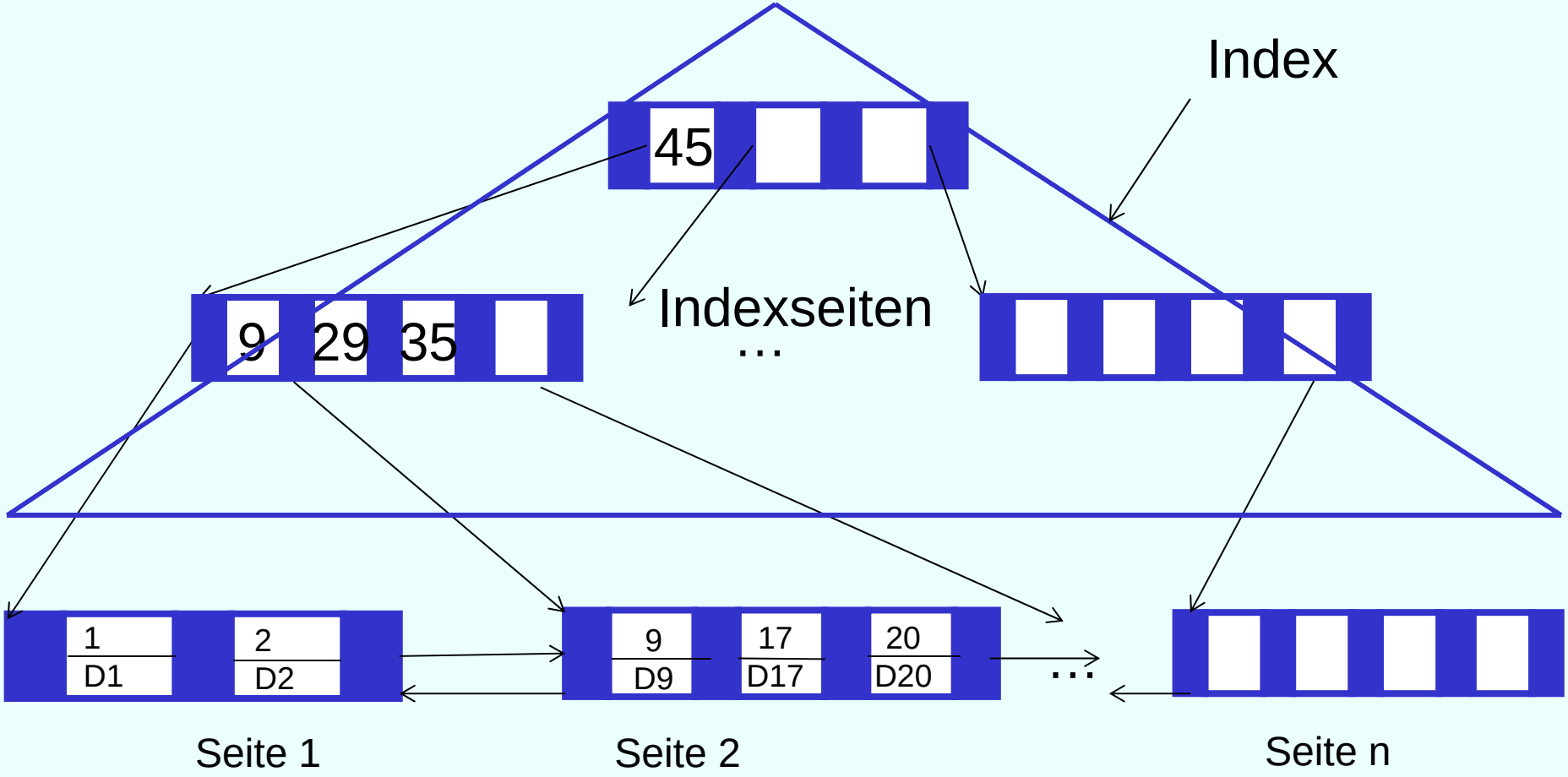
Kurze Wiederholung Indexe, dann Transaktionen

- B+-Baum
- Sekundärindexe

B+-Bäume

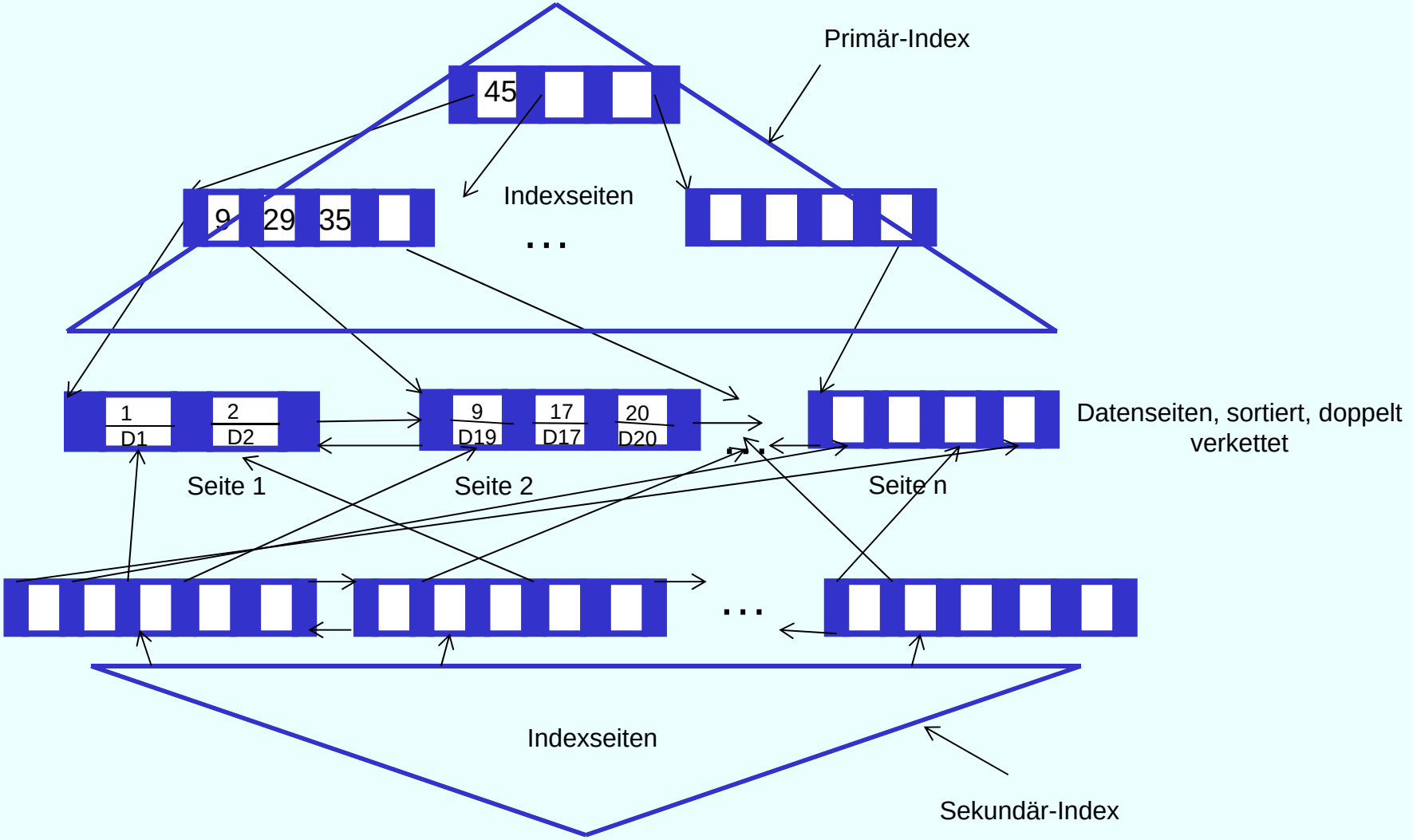
- Performanz eines B-Baums ist stark abhängig von der Höhe des Baumes ab → möglichst hoher Verzweigungsgrad der inneren Knoten
- Abspeichern von Daten in inneren Knoten reduziert den Verzweigungsgrad
- B+-Bäume speichern nur Referenzschlüssel in inneren Knoten, die Daten selbst werden in Blattknoten gespeichert
- Meistens sind die Blattknoten noch verkettet, um schnelle sequentielle Suche zu ermöglichen

Struktur B+-Baum



Datenseiten, sortiert,
doppelt verkettet

Sekundärindexe



Transaktionsparadigma

Definition: Transaktion

‚ununterbrechbare' Folge von DML-/DDL-Befehlen

begin transaction --- *end transaction*

- *begin*: meist implizit mit ersten Datenbankzugriff
- *end*: *commit* (*work*) oder *abort* (*rollback*)

Überführung der Datenbank von einem logisch konsistenten Zustand in einen neuen logisch konsistenten Zustand

→Eigenschaften von TAs: ACID-Prinzip

ACID-Prinzip

Atomicity: "Alles-oder-Nichts"-Eigenschaft (Fehlerisolierung)

Consistency: Erhalt der DB-Konsistenz (definierte Integritätsbedingungen)

Isolation: Logischer Einbenutzerbetrieb (alle Aktionen innerhalb einer TA müssen vor parallel ablaufenden TAs verborgen werden)

Durability: Überleben aller Änderungen trotz beliebiger (erwarteter) Fehler garantiert (Persistenz)

Transaktionsverwaltung

Anforderungen:

- Synchronisation (concurrency control): mehrere TAs sollen gleichzeitig (nebenläufig) ablaufen
- Logging
- Recovery
- Commit-Behandlung
- Integritätssicherung

Anomalien (1)

Im Mehrbenutzerbetrieb kann es zu folgenden Anomalien kommen:

- Lost Update
- Dirty Read
- Non-Repeatable Read
- Phantom Reads

Anomalien (2)

Lost Update:

Schritt	T1	T2
1	read(A, a1)	
2	a1 = a1 – 300	
3		read(A, a2)
4		a2 = a2 *1,03
5		write(A, a2)
6	write(A, a1)	
7	read(B, b1)	
8	b1 = b1 + 300	
9	write(B, b1)	

T1 transferiert 300 € von Konto A nach B.

T2 schreibt Konto A 3% Zinsen gut.

Interessante Schritte: 5 und 6.

Änderung von TA 2 unkontrolliert überschrieben und somit verloren.

Anomalien (3)

Dirty Read

Schritt	T1	T2
1	read(A, a1)	
2	a1 = a1 – 300	
3	write(A, a1)	
4		read(A, a2)
5		a2 = a2 * 1,03
6		write(A, a2)
7	read(B, b1)	
8	...	
9	abort	

T1 transferiert 300 € von Konto A nach B.

T2 schreibt Konto A 3% Zinsen gut.

Interessante Schritte: 4 und 9.

T1 muss zurückgesetzt werden,
T2 hat aber in Schritt 5/6 die Zinsen auf "falschem" Wert berechnet.

Anomalien (4)

Non-Repeatabe Read

Schritt	T1	T2
1	select gehalt from pers where pnr = 2	
2		update pers set gehalt = gehalt + 1000 where pnr = 2
3		update pers set gehalt = gehalt + 2000 where pnr = 3
4	select gehalt from pers where pnr = 2	

T1 liest verschiedene Gehälter.

T2 vergibt Gehaltserhöhungen.

T1 liest zweimal ein Gehalt mit zwei unterschiedlichen Ergebnissen.

Anomalien (4)

Non-Repeatabe Read (komplexerer Fall)

Schritt	T1	T2
1	select gehalt into :gehalt from pers where pnr = 2	
2	s = s + gehalt	
3		update pers set gehalt = gehalt + 1000 where pnr = 2
4		update pers set gehalt = gehalt + 2000 where pnr = 3
5	select gehalt into :gehalt from pers where pnr = 3	
6	s = s + gehalt	

T1 addiert verschiedene Gehälter.

T2 vergibt Gehaltserhöhungen.

Wenn man T1 ein weiteres Mal ausführt, bekommt man eine andere Summe *s* als beim ersten Mal.

Anomalien (5)

Phantom Read

Schritt	T1	T2
1		select sum(KontoStand) from Konten;
2	insert into Konten values (C, 1000)	
3		select sum(KontoStand) from Konten;

T2 fragt zweimal die Summe aller Kontostände ab.

T1 erzeugt eine neues Konto mit 1000 € Guthaben.

T2 berechnet innerhalb derselben TA zwei unterschiedliche Werte.

Synchronisation (1)

Korrektheitskriterium (Ziel):

logischer Einbenutzerbetrieb, d.h. Vermeidung **aller** Mehrbenutzeranomalien

Formales Korrektheitskriterium:

Serialisierbarkeit: Parallele Ausführung einer Menge von Transaktionen ist serialisierbar, wenn es ***eine*** serielle Ausführung derselben TA-Menge gibt, die den gleichen DB-Zustand und die gleichen Ausgabewerte wie die ursprüngliche Ausführung erzielt.

Synchronisation (2)

Aber: Serialisierbarkeit behindert parallele Ausführung von Transaktionen

➔Inkaufnahme von Anomalien ermöglicht weniger Behinderung von TAs
sehr mit **Vorsicht** zu verwenden!!

Wie Gewährleistung der Serialisierbarkeit?

.. durch *Sperrverfahren*.

Sperrverfahren (1)

Beispiel: RX-Sperrverfahren (einfach)

zwei Sperrmodi:

Lese- oder Read (R)-Sperren

Schreib- oder exclusive (X)-Sperren

Kompatibiltitätsmatrix:

	keine	R	X
R	+	+	-
X	+	-	-

"+" bedeutet: Sperre wird gewährt

"-" bedeutet: Sperrkonflikt

Deadlock, Timeout

- Unverträglichkeit eines Sperrwunsches: TA muss warten
- Deadlock: in periodischen Abständen (einstellbar) wird nach Deadlocks gesucht (Zyklen-Erkennung), durch Zurücksetzen einzelner TA aufgelöst
- Timeout: maximale Wartezeit (einstellbar) nach der eine TA abgebrochen wird

Optimierungen

Reduzierung der Beeinträchtigungen:

- hierarchische Sperrverfahren
- reduzierte Konsistenzebene
- Zeitstempel
- Snapshopt Isolation (Oracle, PostgreSQL, ..)
  (Mehrversionen-Ansatz)
- Optimistische Synchronisation

Hierarchische Sperren

Sperrgranulate: Table Spaces, Tables, Rows

Anwartschaftssperren (Intentionssperren)

Sperren auf Tabellen (tables)

Sperren auf Zeilen (rows)

...

Konsistenzebenen in SQL

vier Konsistenzebenen (isolation levels)

bestimmt durch die Anomalien, die in Kauf genommen werden

Lost Update wird immer vermieden: Schreibsperren bis zum TA-Ende.

Default: Serializable

	Dirty Read	Non-Repeatable R.	Phantome
Read Uncomitted	+	+	+
Read Comitted	-	+	+
Repeatable Read	-	-	+
Serializable	-	-	-

Zeitstempelverfahren

- Zeitstempel:
  - jedes Objekt hat zwei Zeitstempel
    1. wann zuletzt gelesen
    2. wann zuletzt geschrieben
  - jede Transaktion hat einen Zeitstempel: wann begonnen
  - Bei Zugriffswunsch werden die Zeitstempel verglichen
    nur, wenn Transaktion jünger ist,
    - bei Lesewunsch jünger als der Schreibstempel
    - bei Schreibwunsch jünger als der Schreib- und der Lesestempel

    darf sie zugreifen und
    den Objektstempel entsprechend setzen

MVCC Multi Version Concurrency Control

Snapshot Isolation

TA sieht DB in dem Zustand, zu dem die TA startet

→ liest die zum Startzustand gültigen Werte

- nur Konflikte zwischen Schreiboperationen → Zurücksetzen der TA
- implementiert mit MVCC
- Vorteil:
  - kein Leser wartet auf einen Schreiber
  - kein Schreiber wartet auf einen Leser
- Nachteil:
  - Mehr Platzbedarf für neue Versionen
  - snapshot isolation garantiert noch keine Serialisierbarkeit
    → serializable snapshot isolation

Optimistische Synchronisation

Optimistische Synchronisation:

1. Lesephase:
   Arbeiten nur auf lokalen Variablen
2. Validierungsphase:
   Konflikt mit anderen Transaktionen?
3. Schreibphase:
   Bei überstandener Validierung: Schreiben in der Datenbank

Klassifikation der Synchronisationsverfahren