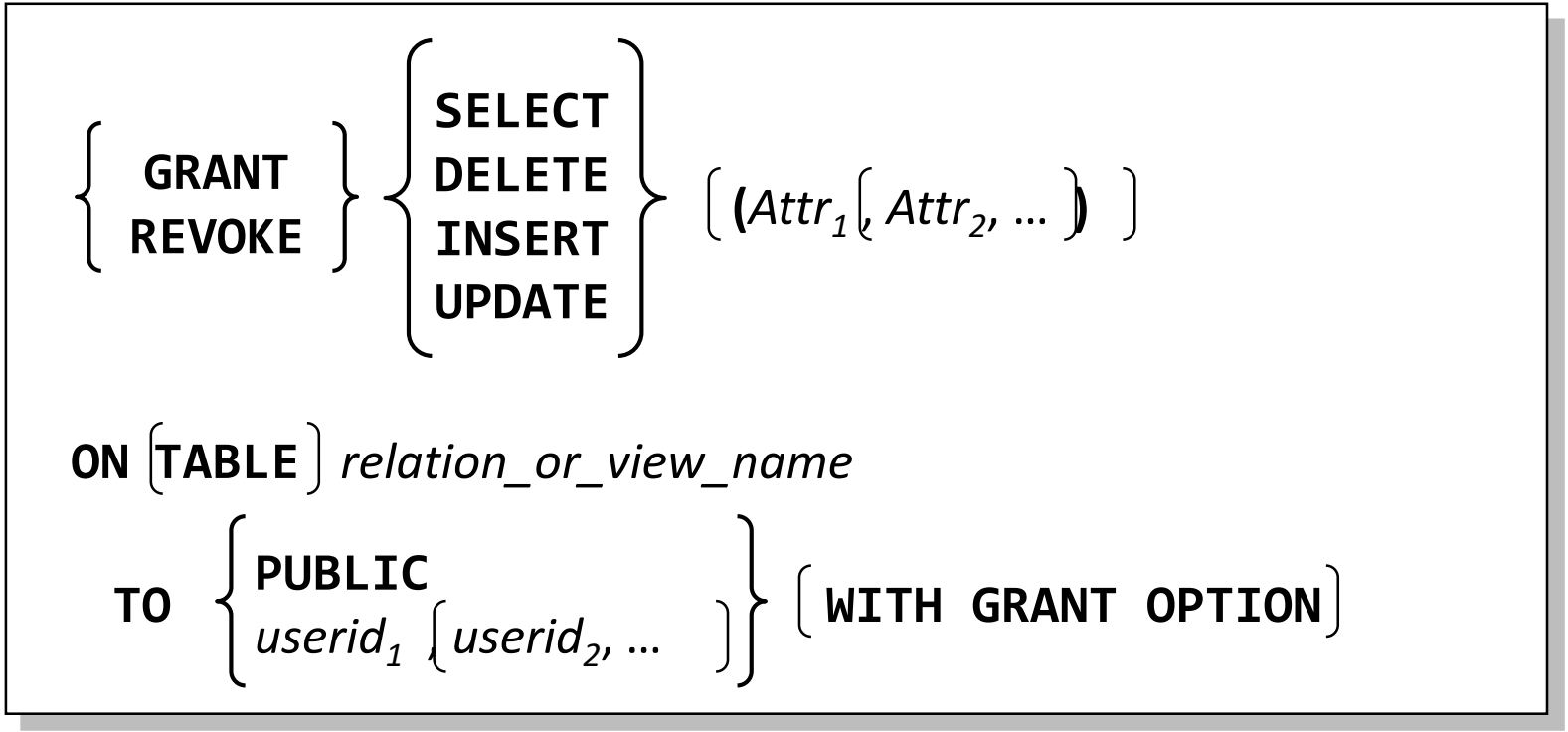
Programmierung von Systemen – 15 – SQL 4

Matthias Tichy & Stefan Götz | SoSe 2020

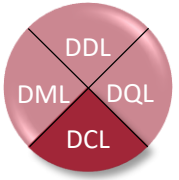
Ziele

- Zugriffskontrolle von DBMS kennen
- Probleme bei parallelem Zugriff erkennen
- Serialisierbarkeitsprinzip verstehen

Zugriffskontrolle

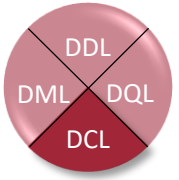


Erläuterungen



- Die Art des Zugriffsrechts kann beschränkt werden (z.B. auf lesenden Zugriff)
- Erteilung der Erlaubnis zur Weitergabe („Vererbung“) von Zugriffsrechten (➔ WITH GRANT OPTION)
- Wird dem „Besitzer“ einer GRANT-Option das Zugriffsrecht entzogen, so erlöschen auch alle von ihm an andere Benutzer weitergegebenen Zugriffsrechte

Erläuterungen



- SQL standardisiert die Rechtevergabe aber nicht die Benutzerverwaltung
- Bei jedem DBMS unterschiedlich ➔ Doku

- Bisher weggelassen, tatsächlich in der Praxis wichtig, insbesondere im Zusammenhang mit Datenbanken, die über das Netzwerk erreichbar sind

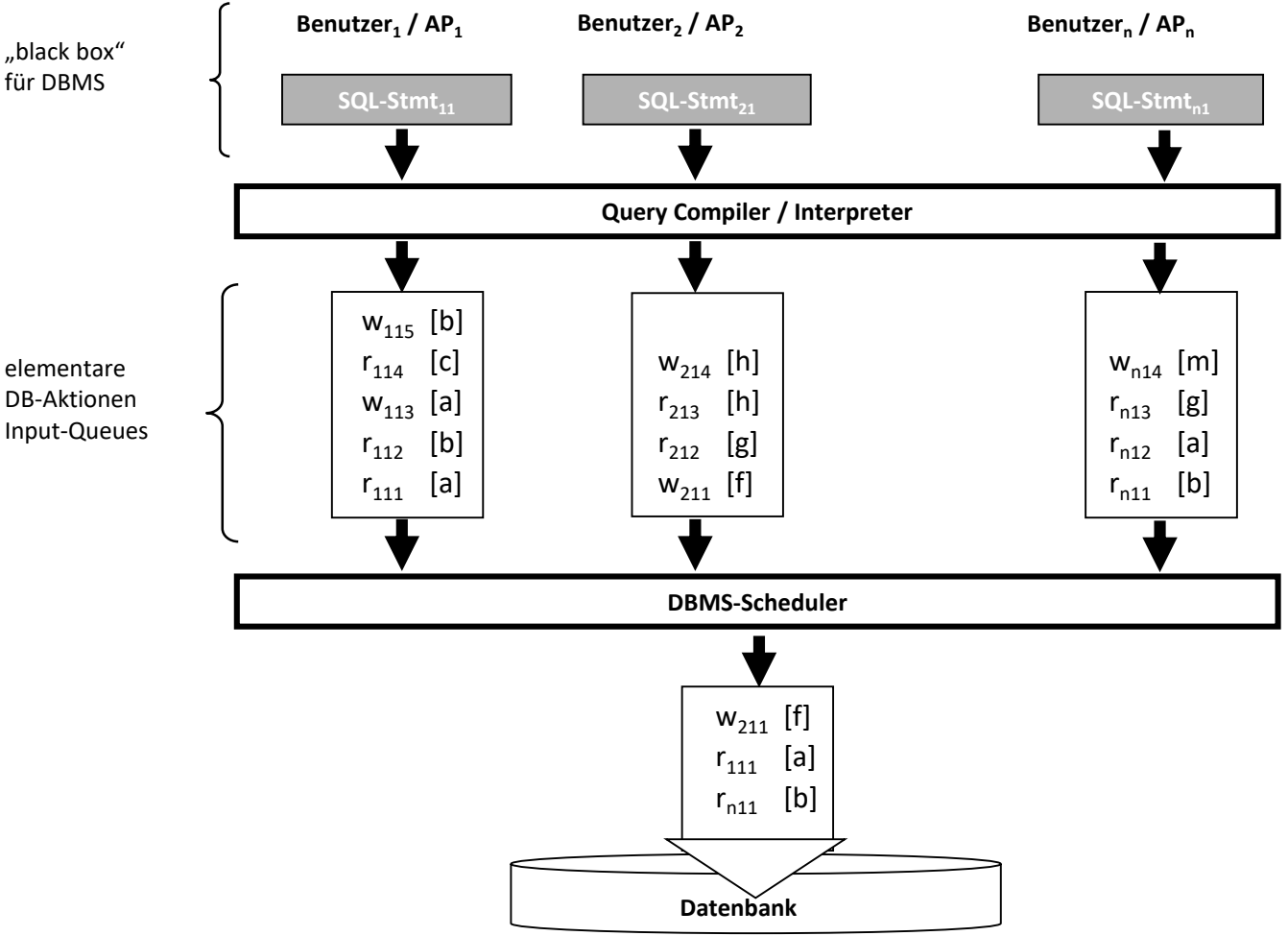
Nebenläufigkeit

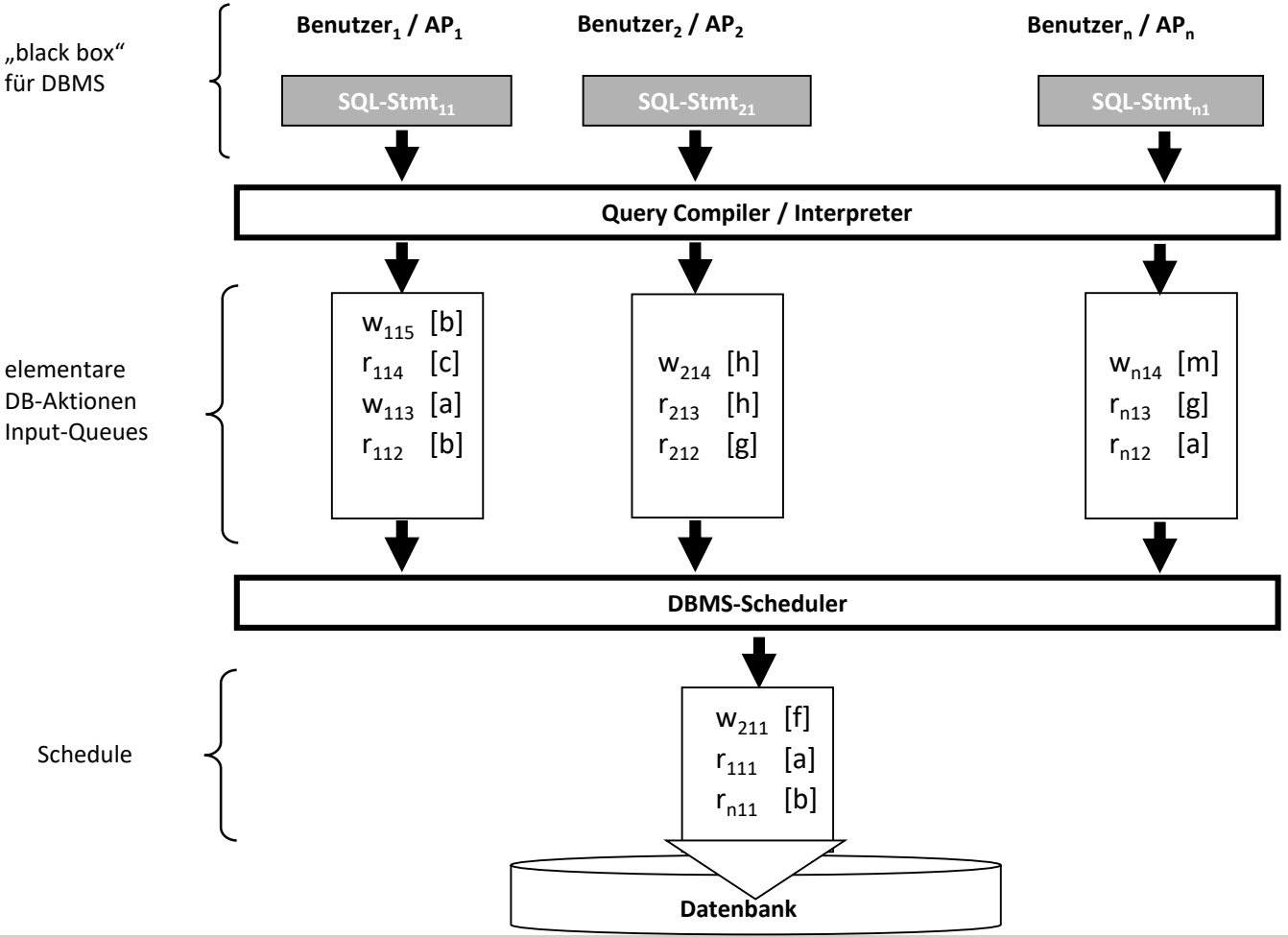
- Nebenläufigkeit immer wichtigeres Thema
  - Multi-Core-Systeme
  - Verteilung von Systemen über Netzwerke
  - reaktive Systeme
- menschliches Denken ist eher sequentiell
  ➔ Probleme, verteilte, nebenläufige Systeme korrekt zu bauen

Allgemeines

Szenario:

- Mehrbenutzerbetrieb
- Verschiedene Benutzer/Anwendungen greifen (lesend und schreibend) auf denselben Datenbestand zu
- Anwendungen werden – aus Performanzgründen – überlappend ausgeführt
- d.h. Ausführung eines Operations-Mixes in der Datenbank

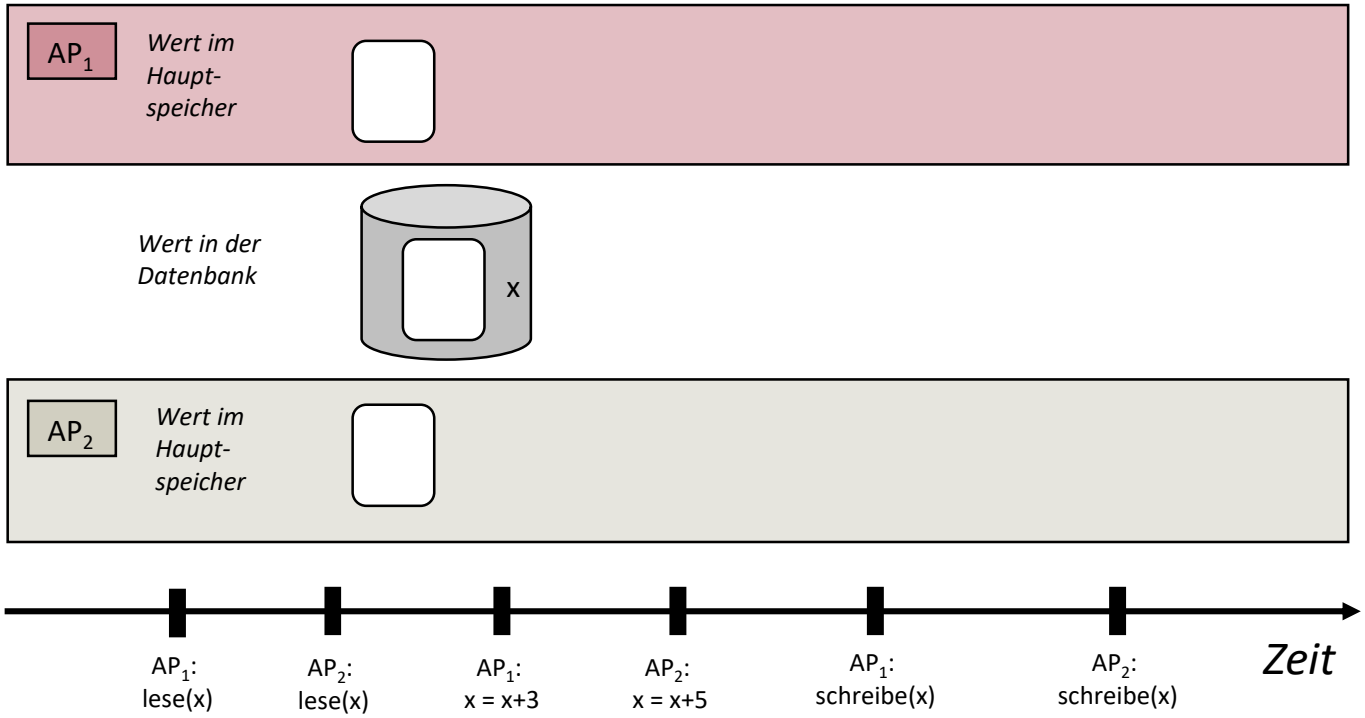

„black box“
für DBMS
Benutzer1 / AP1
Benutzer2 / AP2
Benutzern / APn
SQL-Stmt11
SQL-Stmt21
SQL-Stmtn1
Query Compiler / Interpreter
elementare
DB-Aktionen
Input-Queues
w115 [b]
r114 [c]
w113 [a]
r112 [b]
r111 [a]
w214 [h]
r213 [h]
r212 [g]
w211 [f]
wn14 [m]
rn13 [g]
rn12 [a]
rn11 [b]
DBMS-Scheduler
w211 [f]
r111 [a]
rn11 [b]
Datenbank


„black box“
für DBMS
Benutzer1 / AP1
Benutzer2 / AP2
Benutzern / APn
SQL-Stmt11
SQL-Stmt21
SQL-Stmtn1
Query Compiler / Interpreter
elementare
DB-Aktionen
Input-Queues
w115 [b]
r114 [c]
w113 [a]
r112 [b]
w214 [h]
r213 [h]
r212 [g]
wn14 [m]
rn13 [g]
rn12 [a]
DBMS-Scheduler
Schedule
w211 [f]
r111 [a]
rn11 [b]
Datenbank

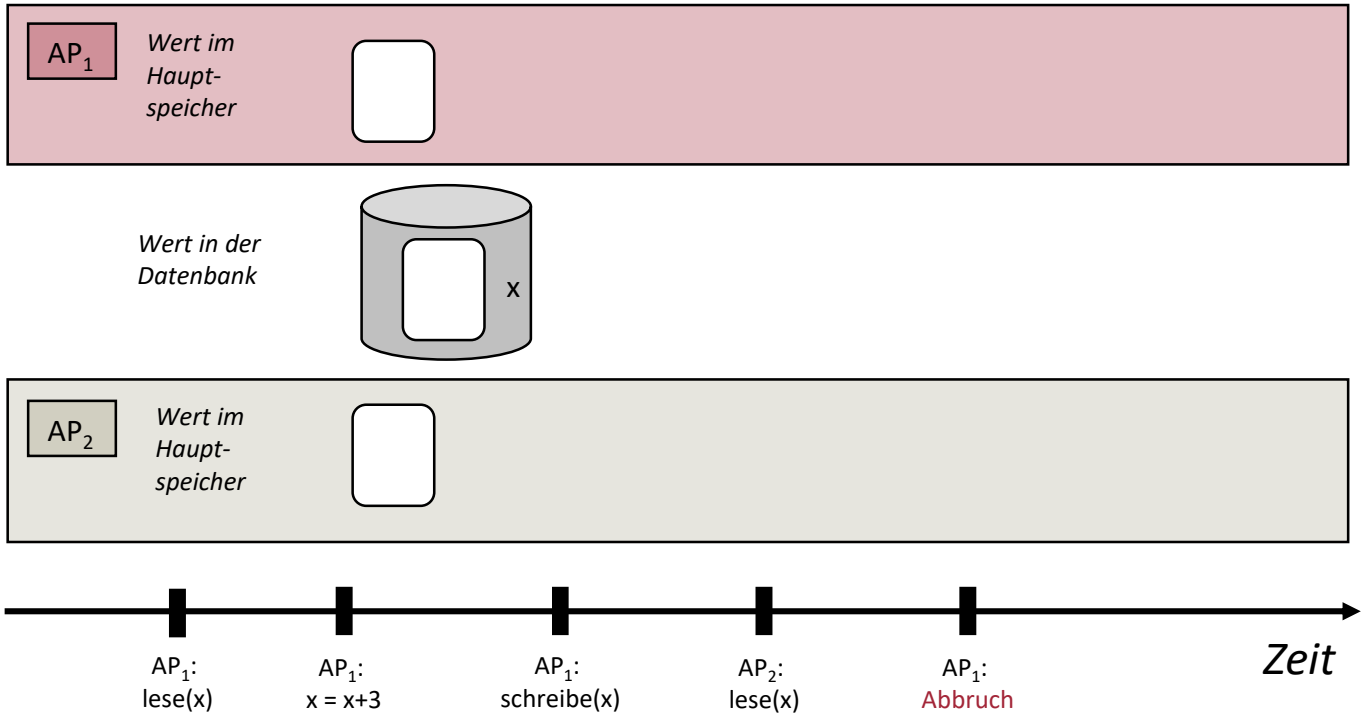
Probleme / Fragestellungen

- Welche aller möglichen Ausführungsreihenfolgen führen zu einem korrekten Resultat?
- Was heißt überhaupt „korrektes Resultat“ bei konkurrierenden Änderungen?
- Wie erkennt und vermeidet man unerwünschte (d.h. nicht korrekte) Ausführungsreihenfolgen?
- Wie geht man mit unvollständigen Änderungen (z.B. wegen Programmabbruch oder Systemcrash) um?

Lost updates



Dirty read


AP1
Wert im Haupt-speicher
Wert in der Datenbank
x
AP2
Wert im Haupt-speicher
AP1: lese(x)
AP1: x = x+3
AP1: schreibe(x)
AP2: lese(x)
AP1: Abbruch
Zeit

Transaktionskonzept

- zusammengehörige Operationen „klammern“ ➔ Transaktion

- Anforderungen:
  - „Alles-oder-nichts“-Prinzip
  - Transaktion überführt DB von einem konsistenten Zustand in einen neuen konsistenten Zustand.
  - Parallele Ausführung von Transaktionen so isolieren, wie wenn die Transaktionen nacheinander ausgeführt worden wären
  - Änderungen abgeschlossener Transaktionen auch durch Systemcrash nicht verloren

Transaktionskonzept

- Das ACID-Paradigma für DB-Transaktionen
  - **A**tomicity .. Realisierung: Synchronisation + Logging/Recovery
  - **C**onsistency .. eine Prämisse
  - **I**solation .. Realisierung: Synchronisation
  - **D**urability .. Realisierung: Logging/Recovery

- Eine Transaktion wird mittels
  - **`START TRANSACTION`** gestartet
  - **`COMMIT`** abgeschlossen
  - **`ROLLBACK [ WORK ]`** abgebrochen (zurückgesetzt)

Serialisierbarkeitsprinzip

- Überlappende Ausführung kein Problem, wenn es für jede Transaktion so aussieht als ob die anderen nicht da wären (Isolierung)
- Beispiel:

  Aktionen T_1: $r_1[a]$ $r_1[b]$ $r_1[c]$ $w_1[c]$

  Aktionen T_2: $r_2[a]$ $r_2[b]$ $r_2[c]$ $w_2[c]$ $w_2[d]$

  Mögliche Schedule (Ausführungsreihenfolge)

  $r_2[a]$ $r_1[a]$ $r_2[b]$ $r_1[b]$ $r_1[c]$ $w_1[c]$ $r_2[c]$ $w_2[c]$ $w_2[d]$

- Frage: Welche der vielen möglichen überlappenden Ausführungen führen zum gleichen Ergebnis wie eine sequentielle Ausführung der beiden Transaktionen?

Serialisierbarkeitsprinzip

Definition: Serialisierbarkeitsprinzip

Eine überlappte Ausführung der Transaktionen T_1, T_2, ..., T_n ist korrekt genau dann, wenn es mindestens eine serielle Ausführungsreihenfolge („serielle Schedule“) der Transaktionen T_1, T_2, ..., T_n gibt, die, angewandt auf denselben Ausgangszustand, zum selben Ergebnis führt.

- Anmerkungen:
  - Das Serialisierbarkeitsprinzip ist die Grundlage von fast allen Synchronisationsverfahren.
  - Die verschiedenen Synchronisationsverfahren realisieren dieses Prinzip zum Teil auf sehr unterschiedliche Weise.
  - Beim Korrektheitsnachweis für ein Verfahren muss gezeigt werden, wie die äquivalente serielle Schedule bestimmt werden kann.

Serialisierbarkeitsprinzip

Serialisierbarkeit kann mit Hilfe eines "Transaktions-Abhängigkeitsgraphen" getestet werden:

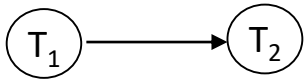
- Knoten = Transaktion
- gerichtete Kante = Vorgänger-Nachfolger-Beziehung zwischen zwei Transaktionen
- Zwei Operationen $op_{1i}[x]$ und $op_{2j}[x]$ von T_1 und T_2 in Konflikt, wenn mindestens eine von beiden schreibend auf x zugreift
- Für alle in Konflikt stehenden Operationen:
  Falls $op_{1i}[x]$ vor $op_{2j}[x]$ in S,
  dann Kante $T_1 \rightarrow T_2$ hinzu,
  sonst Kante $T_2 \rightarrow T_1$
- Ist der Graph azyklisch, kann mittels der topologischen Knotensortierung die äquivalente serielle Ausführungs-reihenfolge bestimmt werden

Serialisierbarkeitsprinzip

Serialisierbar? $r_2[a]\ r_1[a]\ r_2[b]\ r_1[b]\ r_1[c]\ w_1[c]\ r_2[c]\ w_2[c]\ w_2[d]$

Analyse:

$r_2[a]$: kein Konflikt
$r_1[a]$: kein Konflikt
$r_2[b]$: kein Konflikt
$r_1[b]$: kein Konflikt
$r_1[c]$: in Konflikt mit $w_2[c]$ ➔ $T_1 \rightarrow T_2$
$w_1[c]$: in Konflikt mit $r_2[c]$ ➔ $T_1 \rightarrow T_2$ (Kante bereits eingetragen)
$r_2[c]$: ab hier keine Konflikte mehr möglich, nur noch Operationen von T_2



Ergebnis:

- Die Schedule ist serialisierbar,
- äquivalente serielle Ausführungsreihenfolge: T1, T2

Serialisierbarkeitsprinzip

- Wenn Transaktionsabhängigkeitsgraph zyklenfrei, dann ist zugrundeliegende Schedule serialisierbar
- Der Transaktionsabhängigkeitsgraph ist allerdings nur für Theoriefragestellungen interessant (Warum?)
- Synchronisationsverfahren der DBMS arbeiten anders, sind aber Teil der Vorlesung „Informationssysteme“ bei DBIS

Ziele

- Zugriffskontrolle von DBMS kennen
- Probleme bei parallelem Zugriff erkennen
- Serialisierbarkeitsprinzip verstehen