Chapter 4.6 Lock-Hierarchies

Database B partitioned into

$B = B_1 \cup B_2 \cup ... \cup B_k$
with $B_i \cap B_j = \varnothing \quad i \neq j$
$b_{i,e} \in B_i$, $y \in B_j$ ...

Object-Locks: r, a, x
r, a, c

Block-Locks:

R, X for complete partition classes (blocks)
R, A, X
R, A, C

Transaction t or DBS decide on "granularity"

Intention-Locks

IR (B_i) resp. (t, B_i, IR) : t has intention to set r-locks on objects of B_i

IX(B_i): ... to set x-locks or r-locks

RIX(B_i): wants R-lock on B_i to read all objects in B_i
t wants IX on B_i, to request x-lock for individual objects in B_i

Compatibility:

	R	X	IR	IX	RIX
R	+	-	+	-	-
X	-	-	-	-	-
IR	+	-	+	+	+
IX	-	-	+	+	-
RIX	-	-	+	-	-

Hierarchical Protocols

lock on Bi	meaningful lock on $b_{i,l}$
R	-
X	-
IR	r
IX	r,x
RIX	x

Wait - resp. dependence graph G_w, deadlock discovery ect. as before

General Case:

Sequence of partitions

$\Pi_1, \Pi_2, ..., \Pi_l$ with
$\Pi_1 = \{B_i\}_{i=1...k}$ and
Π_{i+1} is refinement of Π_i

Compatibility

VM	R	X	A	IR	IX	RIX	IA	RIA
A	+	-	-	+	-	-	-	-
IA	+	-	-	+	+	-	+	-
RIA	+	-	-	+	-	-	-	-

Example for VM [IA, RIX] = -
Assumption VM [IA, RIX] be +
two transactions s, t :
t reads state Z1 with RIA
t wants change to Z2 with a-lock on o_1

s changes to Z3 with a-lock on o_2, IA on block
IA ⇢ IX
a ⇢ x on o_2
commit s

t changes to Z2;
RIA ⇢ RIX
a ⇢ x on o1 => lost update of s
commit t

Lock-Protocol: meaningful and allowed locks in order of refinement steps

locks on blocks of partition	allowed locks next level	allowed locks on objects
R	-	
X	-	
A	-	
IR	R, IR	r
IX	all	r, x, a
RIX	X, A, IX, IA	x, a
IA	R, A, IR, IA, RIA	r, a
RIA	A, IA	a

Conversion at time of commit:

in hierarchy always from top to bottom:

A ⇢ X
IA ⇢ IX
RIA ⇢ RIX
a ⇢ x

Tradeoff for Lock-Hierarchies

minimal inhibition => small locks
simple lock management => few large locks
e.g. sorting, relation-scan

Question: Who determines lock-granularity?
Transaction or DBS?

Additional Synchronization-Variants

Predicate Locks:
lock * **from** R X **where** P(X)

Problem: $\{X|P_1(X)\} \cap \{X|P_2(X)\} = \varnothing$?

lock: R.A, S.B **from** R, S
where P (R, S)

with aggregat-functions?
i. a. $\bigwedge_i P_i(\vec{x}_i) = \varnothing$?

Interval-Locks: special predicate.Locks of the form

$$P(X) \equiv a \leq x \leq b$$

in combination with indexes

Time-stamp Method: see chapter 6

ESCROW-Techniques: see papers of Pat O'Neil

Chapter 4.7 Optimistic Synchronization

Opt. Concurrency Control OCC

Kung, Robinson: On optimistic methods for concurrency control.
ACM TODS 6 (2): 213-226, 1981.

Gawlick: Processing "hot spots" in high performance systems.
Proc. IEEE COMPCON Conference, S. 249-251, 1985.

Assumption: conflicts are rare, goal: avoid lock management and treatment of deadlocks.

3-Phases of a Transaction:

1. **Read-Phase:** t works only in private storage (work area)
   Read Set (RS_t)
   Write Set (WS_t)

2. **Validation-Phase:** Violation of consistency between t_{val} and other t

3. **Write-Phase:** Changes -> DB

Essential Properties:
- no central lock manager, t manages own RS, WS
- phases 2 and 3 atomic
- no "blind writes": $WS_t \subseteq RS_t$
- sequence of validation = order of serialization
  (Transaction Number Counter TNC)

2 Validation Variants:

BoC: Backward oriented validation
FoC: Forward oriented validation

BoC: "Did t_{val} read old data?"
ValSet (t_{val}) = {t|TNC (t) < TNC (t_{val}) $\wedge$ t has validated after time (BOT(t_{val}))}
Violation: $\exists t \in$ ValSet (t_{val}) : $RS_{tval} \cap WS_t \neq \varnothing$
resp. serializable: $\forall t \in$ ... : $RS_{tval} \cap WS_t = \varnothing$

Example: Abb. 6

Advantes of BoC:
- ValSet (t_{val}) is completed, i.e. no active transactions disturbed or stopped.

Disadvantages of BoC:
- conflicts are discovered very late, at EOT, then (t_{val}) is **aborted** and started again.
- Problem of starvation
- unreal conflicts (Abb.7)

FoC: Forward-Validation
"who does not see new data of t_{val}"
ValSet (t_{val}) = {t|Zeit(BOT(t_{val}) < Zeit (EOT(t))}

Conflict if:
$\exists t \in$ ValSet (t_{val}) : $WS_{tval} \cap RS_t \neq \varnothing$

see. Abb.8

Disadvantages of FoC:
For validation- and write-phase of t_{val} all active transactions must
be stopped, RS_t may not change.
Read from DB, optimization by Caching of WS_{tval}

Advantages:
- conflicts recognized early
- WS need ot be saved unnecessarily long
- avoidance of unreal conflicts
- no validation of Read-Only Transaction

Flexibility of solving conflicts

- Kill t at conflict of t with t_{val}
- Abort t_{val}

Original Idea: D. Gawlick, IMS Fastpath von IBM

4.6.8

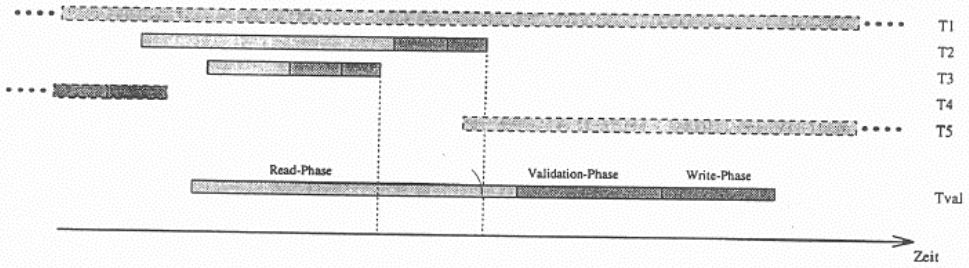

Abbildung 6: *Rückwärtsvalidierung (BoC)*

4.6.9

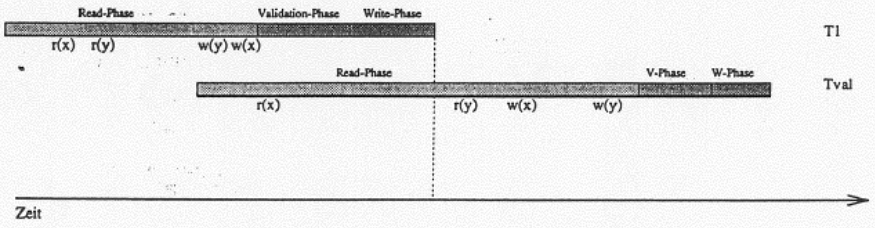

Abbildung 7: *Unechte Konflikte*

4.6.10

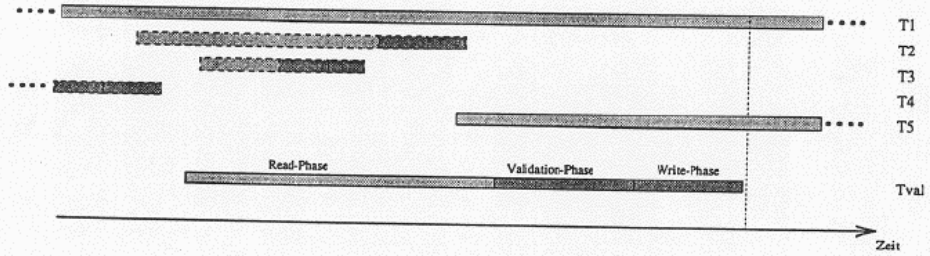

Abbildung 8: *Vorwärtsvalidierung (FoC)*