

# טרנזקציות

## תזכורת

- לכל טרנזקציה יש מקום משלה בזיכרון, ויש גם buffer שמשותף לכל הטרנזקציות.
- פעולת input (output) קוראת (כותבת) מידע מה DB לbuffer.
- פעולת read (write) קוראת (כותבת) מידע מהbuffer לזיכרון הפרטי של הטרנזקציה
- טרנזקציה תמיד מעבירה בסיס נתונים ממצב תקין למצב תקין - אם לפני הטרנזקציה מסד הנתונים היה במצב תקין, והטרנזקציה פועלת לבד, אז גם אחרי הטרנזקציה הוא צריך להיות במצב תקין.
- אם הטרנזקציה לא הצליחה להסתיים - היא צריכה לבטל את עצמה.

## דוגמה

$$F = \{A = B\}$$

$$T1 : r(A, t), t = t + 100, w(A, t), r(B, t), t = t + 100, w(B, t)$$

$$T2 : r(A, s), s = s * 2, w(A, s), r(B, s), s = s * 2, w(B, s)$$

נריץ את הטרנזקציות אחת אחרי השניה - קודם T1 ואחר כך T2:

T1	T2	A	B
		25	25
$r(A, t)$ $t = t + 100$ $w(A, t)$		125	
$r(B, t)$ $t = t + 100$ $w(B, t)$			125
	$r(A, s)$ $s = s * 2$ $w(A, s)$	250	
	$r(B, s)$ $s = s * 2$ $w(B, s)$		250

היינו גם יכולים להריץ קודם T2 ואחר כך T1, והיינו מקבלים תוצאות שונות - אבל עדיין נכונות:

$T1$	$T2$	$A$	$B$
		25	25
	$r(A, s)$ $s = s * 2$ $w(A, s)$	50	
	$r(B, s)$ $s = s * 2$ $w(B, s)$		50
$r(A, t)$ $t = t + 100$ $w(A, t)$		150	
$r(B, t)$ $t = t + 100$ $w(B, t)$			150

## Serial Schedule

זה schedule שבו כל טרנזקציה מתבצעת בצורה מלאה, אחת אחרי השנייה. אם יש לנו  $n$  טרנזקציות אזי יכולים להיות לנו  $n!$  schedule כאלה, שכל אחד אחד מהם מביא את בסיס הנתונים ממצב תקין למצב תקין.

## שקול סדרתי

$T1$	$T2$	$A$	$B$
		25	25
$r(A, t)$ $t = t + 100$ $w(A, t)$		125	
	$r(A, s)$ $s = s * 2$ $w(A, s)$	250	
$r(B, t)$ $t = t + 100$ $w(B, t)$			125
	$r(B, s)$ $s = s * 2$ $w(B, s)$		250

למרות שביצענו את הטרנזקציות במקביל, קיבלנו תוצאה תקינה - אותה תוצאה של ה-schedule הראשון שהראנו. הרצה שנותנת אותן תוצאות כמו serial schedule נקראת "שקול סדרתי", או serializable.

יש גם הרצות שאינן שקולות סדרתית. לדוגמה:

<i>T1</i>	<i>T2</i>	<i>A</i>	<i>B</i>
		25	25
$r(A, t)$ $t = t + 100$ $w(A, t)$			
	$r(A, s)$ $s = s * 2$ $w(A, s)$	250	
	$r(B, s)$ $s = s * 2$ $w(B, s)$		50
$r(B, t)$ $t = t + 100$ $w(B, t)$			150

קיבלנו  $A = 250 \neq 150 = B$  - התוצאות לא נכונות, ולא שקולות לאף serial schedule.  
הרצה כזאת נקראת non-serializable.

<i>T1</i>	<i>T2</i>	<i>A</i>	<i>B</i>	<i>T1</i>	<i>T2</i>	<i>A</i>	<i>B</i>
		25	25			25	25
$r(A, t)$ $t = t + 100$ $w(A, t)$				$r(A, t)$ $t = t + 100$ $w(A, t)$			
	$r(A, s)$ $s = s + 200$ $w(A, s)$	325		$r(B, t)$ $t = t + 100$ $w(B, t)$			125
	$r(B, s)$ $s = s + 200$ $w(B, s)$		225		$r(A, s)$ $s = s + 200$ $w(A, s)$	325	
$r(B, t)$ $t = t + 100$ $w(B, t)$			325		$r(B, s)$ $s = s + 200$ $w(B, s)$		325

כאן יש לנו סדר שונה, אבל אותה תוצאה. אבל זה רק במקרה!(בגלל הקומוטטיביות והסימטריות של החיבור).

## Conflict Serializability

נסתכל רק על הפעולות העיקריות - קריאה, כתיבה, קלט ופלט.

$T1 : r(A), w(A), r(B), w(B)$

$T2 : r(A), w(A), r(B), w(B)$

איזה סדר יוצר קונפליקט, ואיזה לא?

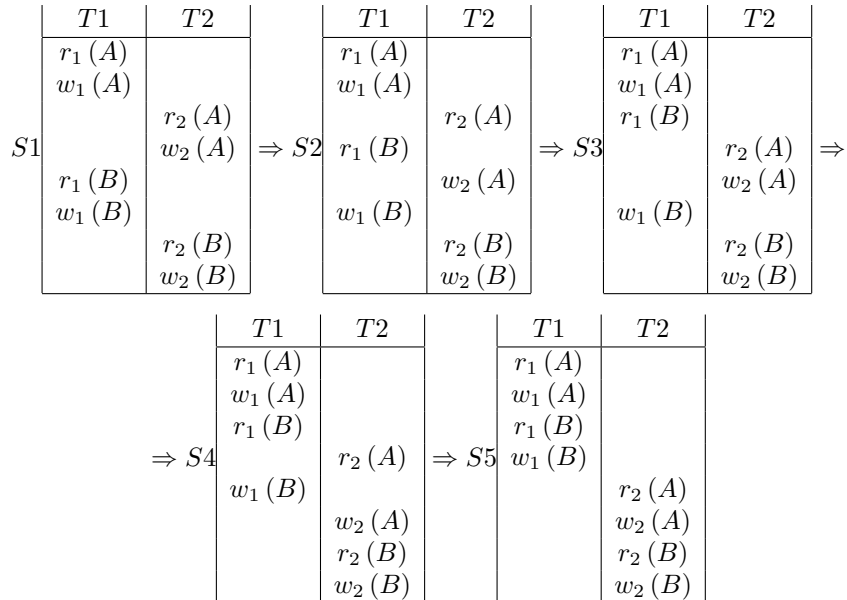
non-conflict	conflict
$r_i(X), r_j(Y)$	$A_i(X), A_j(Y)$
$r_i(X), w_j(Y), X \neq Y$	$r_i(X), w_j(X)$
$w_i(X), w_j(Y), X \neq Y$	$w_i(X), r_j(X)$
$w_i(X), w_j(Y), X \neq Y$	$w_i(X), w_j(X)$

כלומר פעולות קריאה בטרנזקציות שונות לא יוצרות קונפליקט אחת עם השנייה, וגם אם יש פעולות כתיבה, אבל של נתונים אחרים, אז אין בעיה. אבל אם יש שני פעולות של אותה טרנזקציה אז זה יוצר קונפליקט (כדי scheduler לא יכול לשנות את הסדר הפנימי של הטרנזקציה), וגם פעולת כתיבה ועוד פעולה על אותם נתונים יוצרת קונפליקט.

## הגדרות

- אם אפשר להעביר schedule אחד לאחר באמצעות סדרת החלפות (בזמן) non-conflict של פעולות (רק של פעולות צמודות) - אז הם נקראים "conflict-equivalent schedules"
- אם ניתן להגיע מהschedule לserial schedule, באמצעות החלפות כנ"ל, אז הוא נקרא conflict-serializable schedule.

## דוגמה



בכל שלב ביצענו רק החלפה אחת, חוקית, בין שני שורות סמוכות. לכן S1 וS5 הם conflict-equivalent, ולכן S1 הוא conflict serializable.

# Precedence graph

זה קשה לבנות סדרת החלפות, ולכן, כדי להקל על הבדיקה, בונים Precedence graph. עבור  $S$  schedulen, מסמנים  $T_1 <_S T_2$  אם קיימות שתי פעולות  $T_1 : A_1$  כך  $T_2 : A_2$  שמתקיים:

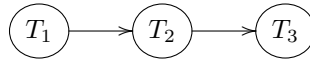
1.  $A_1$  לפני  $A_2$  ב  $S$
2. שתיהן עובדות על אותו נתון  $X$
3. לפחות אחת מהם היא פעולת כתיבה

## דוגמה

$S : r_2(A), r_1(B), w_2(A), r_3(A), w_1(B), w_3(A), r_2(B), w_2(B)$

- $T_2 < T_3$  - נסמן  $T_3$  לפני  $T_2$
- $T_1 < T_2$  - נסמן  $T_2$  לפני  $T_1$

לכן נצייר את הגרף

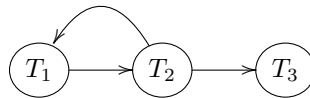


זהו DAG (Directed Acyclic Graph). אם Precedence Graph הוא DAG, אז schedulen הוא conflict serializable.

## דוגמה אחרת

$S' : r_2(A), r_1(B), w_2(A), r_2(B), r_3(A), w_1(B), w_3(A), w_2(B)$

הגרף של זה הוא



## הערה

conflict serializable הוא תנאי מספיק, אבל לא הכרחי, כדי להיות serializable