בקרת מקביליות

גישה לקבצים באמצעות תנועות

ניגשים לקבצים באמצעות פעולות בסיסיות
- קריאת גוש (בלוק) מהדיסק
- כתיבת גוש (בלוק) לדיסק

תנועה (transaction) היא פעולה לוגית המורכבת מכמה פעולות בסיסיות
- הכנסת רשומה לעץ B+

שכיח במסדי נתונים, למשל, במסד נתונים של בנק יש את התנועות הבאות לניהול חשבון עובר ושב
- **עמלה חודשית:** בכל סוף חודש, החשבון מחויב בסכום קבוע של 10₪.
- **ריבית שבועית:** בכל יום שישי, מזוכה החשבון בריבית של 0.1% על יתרת זכות, ולחילופין, החשבון מחויב בריבית של 1.0% על יתרת חובה.

ניהול תנועות

בחלק מהמערכות, כל תנועה מבוצעת על ידי **תהליך**

ביצוע התנועות מנוהל על ידי **מערכת בקרת גישה** אשר קובעת

- מתי כל תנועה תתחיל
- האם פעילותה של תנועה תושהה
- האם פעילותה של תנועה תופסק

התוצאה תלויה בסדר החישוב!

לדוגמה, כאשר החודש מסתיים ביום שישי, הבנק צריך לבצע שתי משימות: ריבית שבועית, ועמלה חודשית. נחשב את היתרה בחשבון הלקוח המכיל 1000₪.

- עמלה ואז ריבית: (1000-10)*1.001 = 990.99
- ריבית ואז עמלה: 1000*1.001-10 = 991

מבחינת מקביליות, שתי האפשרויות נכונות

ביצוע סדרתי (serial) של תנועות

- המערכת לא מאפשרת ביצוע תנועות במקביל
- כאשר המערכת מתחילה בביצוע של תנועה:
  - אם אין תנועה אחרת המתבצעת כעת, המערכת ממשיכה בביצוע
  - אם יש תנועה המתבצעת כעת, התנועה נכנסת לתור המתנה (המנוהל לפי מדיניות כלשהי)
- כאשר תנועה מסיימת:
  - אם יש תנועות נוספות בתור ההמתנה, התנועה שבראש התור מוצאת ממנו, והמערכת ממשיכה את ביצועה.

☹גם אם יש לנו כמה דיסקים, וכל תנועה ניגשת לדיסק נפרד, הן אינן מבוצעות במקביל.

☹ניצול גרוע של אמצעי המערכת — ספיקה (throughput) נמוכה.

☹לא ניתן לשפר ע"י תוספת ציוד.

אולי נבצע כמה תנועות בו-זמנית?

עמלה	ריבית	x	y	A
		---	---	1000
`x = READ balance`		1000		
	`y = READ balance`		1000	
`x = x - 10`		990		
	`y = y * 1.001`		1001	
`WRITE x TO balance`				990
	`WRITE y TO balance`			1001

נכונות כל תנועה בנפרד אינה מבטיחה נכונות ביצוע התנועות במקביל!

ניהול תנועות מקבילי (concurrent)

- מערכת בקרת הגישה מאפשרת למספר תנועות להתבצע בו-זמנית.
- **בקר מקביליות** (concurrency controller), המהווה חלק ממערכת בקרת הגישה, מוודא שלא נוצרות שגיאות הנובעות מהביצוע המקבילי.
  - תנועות שלא "מפריעות" זו לזו יוכלו להתבצע בו-זמנית.
  - אם שתי תנועות עשויות להפריע זו לזו, בקר המקביליות יכול לעכב את אחת התנועות, או אף להפסיק אותה.

ניהול סדרתי
- פשוט למימוש.
- מוריד באופן ניכר את תפוקת המערכת
- נמנע מבעיות של גישה מקבילית.

ניהול מקבילי
- מסובך יותר
- מבטיח מקביליות גבוהה ככל האפשר

נדרוש שלא תיווצרנה בעיות חדשות, הנובעות מהמקבול.
⇦ביצוע מקבילי הוא נכון אם "נראה כמו" ביצוע סדרתי כלשהו.

בקרת מקביליות: ראשי פרקים

☺ **הגדרת הרצוי (סדרתיות)**:

1. הגדרה של ביצוע "נכון"
2. מציאת קריטריון המאפשר ביצוע "נכון" ומניעת התרחשות בעיות הנובעות ממקביליות
3. מניעת בעיות כגון חבק (deadlock)

☺ **מימוש אלגוריתם זיהוי ומניעה**:

1. מנעולים
2. פרוטוקולים

☺ **התאוששות מביטולים ונפילות**

תכונות ACID: אטומיות (Atomicity)

☺ תנועה היא **אטומית** אם המערכת מבטיחה את ביצועה כמקשה אחת. כלומר, לא ייתכן שחלק מהפעולות שלה יצליחו וחלק יכשלו.

☺ החומרה מבטיחה שפעולות מסוימות הן אטומיות, לדוגמה:

1. פעולות אריתמטיות באוגרים
2. הוראת test and set או test and clear (למימוש מניעה הדדית)
3. קריאת גזרה (סקטור)
4. קריאת מערכת הפעלה (מבחינת תהליך המשתמש)

❖ מה בקשר לכתיבת גזרה? קריאת רשומה? כתיבת רשומה?

❖ בד"כ מניחים שפעולות מרמות אבסטרקציה נמוכות יותר הן אטומיות מפני שאותה רמה מספקת מנגנונים המבטיחים זאת

תכונות ACID: עקביות (Consistency)

תנועה היא **עקבית** אם ביצועה שומר על נכונות מבנה הנתונים ואינו מפר את האילוצים המוגדרים על הנתונים.

לדוגמה, במסד נתונים של בנק, כל חשבון משויך ללקוח. במחיקת רשומת לקוח יש לוודא מחיקת כל החשבונות המשויכים בלעדית ללקוח המבוטל.

אם $T_1, \ldots, T_n$ תנועות אטומיות עקביות, ו D הוא מבנה נתונים נכון,
אז אחרי ביצוע $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$, D נשאר נכון.

תנועה מורכבת הנכשלת לפני סופה, עלולה לפגום בעקביות מבנה הנתונים.

עקביות עם תנועות מורכבות

אם תנועה מופסקת לפני סיומה (ABORT) חייבים לבטל את כל פעולות הכתיבה (לשחזר את מה שהיה כתוב לפני ביצוע התנועה)

דוגמה: העברת 1000 ₪ מחשבון A לחשבון B:

```
x = READ A
WRITE x - 1000 TO A
y = READ B
WRITE y + 1000 TO B
```

צריך לשמור את הערך המקורי של A
ואם התנועה הופסקה, לשחזר את הערך המקורי של A
(אבל מה נעשה אם תנועה אחרת כבר קראה את הערך של A?)

תכונות ACID: בידוד (Isolation)

הפעולה שמבצעת תנועה (המשמעות שלה) אינה תלויה בתנועות אחרות

- ישנן רמות בידוד שונות המגדירות אלו פעולות שמבצעת תנועה יכולה תנועה אחרת, הרצה במקביל, לראות.
- ככל שרמת הבידוד גבוהה יותר, תנועה פחות מושפעת מתנועות אחרות, אך רמת המקבילות יורדת, ועקב כך ביצועי המערכת עלולים להיפגע.

תכונות ACID: שרידות (Durability)

האפקט של תנועה שהסתיימה לא נעלם

דוגמה: העמלה נגבתה ומופיעה בקובץ

"דוגמה נגדית": כתיבה בזיכרון הראשי הנעלמת בהפסקת חשמל

ביטול תנועה: שחזור כל הקבצים במערכת למצב שהיה לפני ביצוע התנועה.
לא תמיד אפשרי (אם הדפסנו הודעה, או נתנו כסף מזומן).

התרת תנועות: לפעמים ניתן לבטל את האפקט של תנועה ע"י תנועה אחרת,
אבל אז מדובר בשתי תנועות שהתקיימו בפועל.

תזמון (schedule)

נקרא גם תרחיש (history)

סדרת הפעולות שהתבצעו במערכת, לפי סדר ביצוען

```
T1                    T2
x  =  READ  A
                      y  =  READ  A
x  =  x  -  100
                      y  =  y  -  200
WRITE  x  to  A
                      WRITE  y  TO  B
```


תזמון סדרתי

כל הפעולות של תנועה אחת מתבצעות לפני כל הפעולות של תנועה אחרת

```
T1                    T2
x  =  READ  A
x  =  x  -  100
WRITE  x  to  A
                      y  =  READ  A
                      y  =  y  -  200
                      WRITE  y  TO  B
```

T1 → T2

ובאופן כללי, ביצוע של T_1 ואז T_2... ואז T_n

$T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$

תזמון מקבילי

```
T1
x = READ A

WRITE x + 5 TO A

END
```

```
T2

y = READ A

WRITE y TO B

END
```

```
T3

z = READ C


WRITE z + 5 TO C
END
```

שקול ל- $T_3 \rightarrow T_2 \rightarrow T_1$
$T_2 \rightarrow T_3 \rightarrow T_1$
$T_2 \rightarrow T_1 \rightarrow T_3$

אבל לא ל- $T_1 \rightarrow T_2 \rightarrow T_3$

עדין לא מוגדר

בקר המקביליות יאפשר רק תזמונים **בני סידור** (השקולים לתזמון סדרתי)

בר סדרתיות (serializability)

בקר מקביליות הוא נכון אם כל התזמונים שהוא מאפשר הם **בני-סידור**:
נראה **כאילו** כל תנועה התבצעה לבד והתנועות התבצעו זו אחר זו.

נרצה לבנות פרוטוקול יעיל (מקבילי) אשר מבטיח נכונות (בר-סידור).
בשלב זה, נניח שאין נפילות או תקלות.

שקילות מצב סופי

תזמון S_1 **שקול מצב סופי** לתזמון S_2 אם מצב המערכת לאחר S_1 זהה למצב המערכת לאחר ביצוע S_2
(כלומר, מצב הדיסק זהה בסיום שני התזמונים)

תזמון הוא **בר-סידור מצב סופי** אם הוא שקול-מצב-סופי לתזמון סדרתי כלשהו.

```
T1                  T2
z = READ B
x = READ A

                    y = READ A
                    y = 2*y
                    WRITE y to A

x = x-z
WRITE x to A
```

מצב התחלתי
A = 10
B = 3

אינו שקול מצב סופי לשום ביצוע סדרתי

מצב התחלתי
A = 0
B = 0

שקילות מצב סופי

תזמון S_1 **שקול מצב סופי** לתזמון S_2 אם **מכל מצב התחלתי**, מצב המערכת לאחר S_1 זהה למצב המערכת לאחר ביצוע S_2

```
T1                  T2
x = READ A          x = READ A
WRITE x TO A        WRITE x TO A
```

כל תזמון הוא בר סידור!

איך בקר המקביליות יכול לדעת איזה פעולות מתבצעות על הנתונים?
צריך לאפשר רק תזמונים שהם בני-סידור, בלי תלות בסמנטיקה

תזמון S_1 **שקול מצב סופי** לתזמון S_2 אם מכל מצב התחלתי, מצב המערכת לאחר S_1 זהה למצב המערכת לאחר ביצוע S_2, ללא תלות בחישובים

בדיקת שקילות מצב סופי

תזמונים ללא ביטויי-תנאי או לולאות (straight-line)
לכל תזמון נרשום את המצב הסופי בצורה סימבולית.

A	B	C	D	
a	b	c	d	ערך לפני
f1(c)	f2(a,d)	f3(a,d,f1(c))	d	ערך אחרי

התזמונים שקולי מצב סופי אם ורק אם לכל משתנה יש אותו ערך סימבולי.

- הביטויים גדלים מאוד (אקפוננציאלי באורך התנועות)
- התנועות מכילות ביטויי-תנאי ולולאות

בעיית ההחלטה אם תזמון הוא בר-סידור מצב סופי היא NP-שלמה

תיקונים בדיעבד?

```
T1
z = READ B
x = READ A


x = x-z
WRITE x to A
```

```
T2


y = READ A
y = 2*y
WRITE y to A
```

```
T3


WRITE 2000 to A
```

מצב התחלתי
A = 1000

כתיבה עיוורת

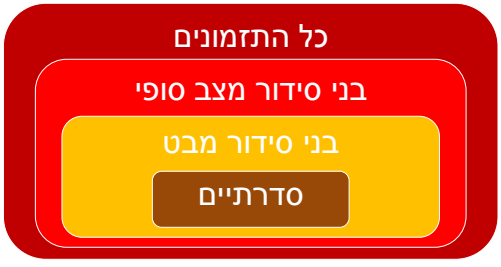
- האם התזמון הכולל רק את T_1 ו-T_2 בר סידור מצב סופי?
- האם התזמון הכולל את T_1, T_2 ו-T_3 הוא בר סידור מצב סופי?
- האם אפשר לסמוך על כך ש T_3 תופיע ו"תתקן" את המצב?
- ומה אם בונים על התנהגות נכונה תוך כדי ביצוע התנועה?

בקר מקביליות מקוון

בר סדרתיות מבט

תזמון S_1 הוא שקול מבט (view equivalent) לתזמון S_2 אם:

- אם ב S_1 תנועה T קוראת ערך v למשתנה x
  אז גם ב S_2, T קוראת ערך v למשתנה x.
- אם ב S_1 תנועה T_1 קוראת ל-x ערך שכתבה תנועה T_2
  אז גם ב S_2, T_1 קוראת ל-x ערך שכתבה T_2.
- אם ב S_1 תנועה T היא האחרונה שכותבת ערך ל-x
  אז גם ב S_2, T היא האחרונה שכותבת ערך ל-x.



תזמון הוא **בר-סידור מבט** אם
הוא שקול מבט לתזמון סדרתי.

בר-סידור מבט ⇦ בר-סידור מצב סופי

שני תזמונים S_1 ו- S_2

אם S_1 שקול מבט לתזמון S_2 ⇦ S_1 שקול מצב סופי ל S_2.

- כל תנועה T בתזמונים מבצעת אותן קריאות בשני התזמונים.
- בגלל תכונת הבידוד של התנועות, ובהנחה שהחישוב דטרמיניסטי,
  Tמחשבת אותם חישובים, מבצעת אותן פעולות, וכותבת אותם ערכים.
- מאחר ו:
  - השוויון בקלט, ועל כן בפלט, זהה לכל התנועות המופיעות בשני התזמונים
  - לכל פריט נתונים, התנועה האחרונה הכותבת אותו היא אותה תנועה בשני התזמונים

⇦ המצב הסופי של שני התזמונים זהה.

למה תזמון אינו בר-סידור?

1. עדכון אבוד (lost update):

```
T1:  READ A
T2: WRITE A
T1: WRITE A
```

עדכון של T1 תלוי בערך הישן של A.

2. קריאה מלוכלכת (dirty read):

```
T1: WRITE A
T2:  READ A
T1: WRITE A
```

T2 קרא ערך ביניים.

3. קריאה שלא ניתן לחזור עליה (unrepeatable read):

```
T1:  READ A
T2: WRITE A
T1:  READ A
```

למרות ש- T1 לא שינתה את A, ערכו השתנה.

תמיד יש פעולות לאותו פריט מידע, ולפחות אחת מהן כתיבה

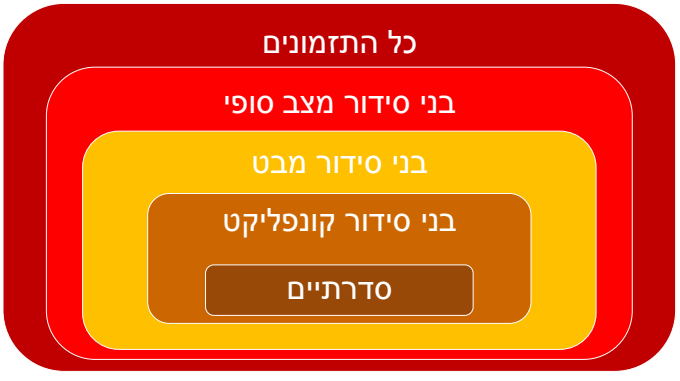
4. האם יש בעיה?

```
T1:  READ A
T2:  READ A
T1:  READ A
```


בר סדרתיות קונפליקט

תנאי **מספיק** לבר-סדרתיות שניתן לבדוק תוך כדי הביצוע,
ולהחליט אם לעכב או לעצור תנועות

שמרני (לא **הכרחי**)



הגדרה: בר-סדרתיות קונפליקט

שתי פעולות **מתנגשות** אם הן מתייחסות לאותו פריט נתונים ולפחות אחת מהן היא כתיבה
- read-write, write-read, write-write –

שני תזמונים הם **שקולי קונפליקט** אם הם
א. מוגדרים על אותה קבוצת תנועות.
ב. מסכימים על הסדר בין פעולות מתנגשות

כלומר, אם פעולה P_1 מתנגשת עם פעולה P_2 ומופיעה לפניה ב- S_1, אז P_1 מופיעה לפני P_2 גם ב- S_2.

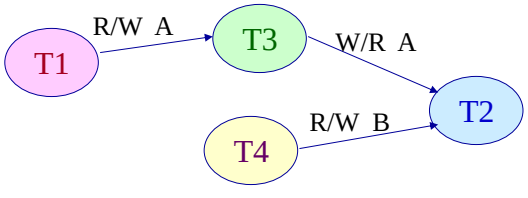
תזמון הוא **בר-סידור קונפליקט** (conflict serializable) אם הוא שקול קונפליקט לתזמון סדרתי (כלשהו).

בדיקה האם תזמון הוא בר-סידור קונפליקט

נתאים לתזמון S גרף תלויות מכוון G(S) (dependency graph):
- לכל תנועה יותאם צומת.
- יש קשת מכוונת מ-T1 ל-T2 אם קיים פריט נתונים ש-T1 כתבה עליו ואחר-כך T2 קראה אותו או כתבה עליו, או T1 קראה אותו ואחר-כך T2 כתבה עליו, זאת-אומרת:

WRITE → READ
READ → WRITE
WRITE → WRITE

תזמון S_1 שקול קונפליקט לתזמון S_2 אם ורק אם $G(S_1) = G(S_2)$



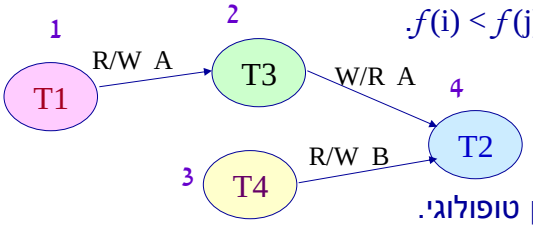
דוגמה:

T1: READ A
T2: READ B
T3: WRITE A
T4: READ B
T2: WRITE B
T2: READ A

תנאי הכרחי ומספיק לבר-סדרתיות קונפליקט

תזמון S הוא בר סידור קונפליקט אם ורק אם
אין מעגלים מכוונים ב G(S), גרף התלויות של S

מיון טופולוגי של גרף עם צמתים V וקשתות E הוא
מיפוי f מ V למספרים 1,…,|V|
כך שאם $i \to j \in E \Leftarrow f(i) < f(j)$.



בדוגמה, (T_1, T_3, T_4, T_2) הוא מיון טופולוגי.

לגרף יש מיון טופולוגי אם ורק אם אין בו מעגלים מכוונים.

תנאי לבר-סדרתיות קונפליקט: הוכחה

⇦ (אם S בר-סידור קונפליקט אזי ב-G(S) אין מעגלים):
נתון S שקול קונפליקט לתזמון סדרתי S'.
G(S) = G(S')
S' קובע מיון טופולוגי ב-G(S') ⇐ ב-G(S) אין מעגלים.

⇨ (אם אין מעגלים ב-G(S) אזי S בר-סידור קונפליקט):
נתון G(S) חסר מעגלים.
מצא מיון טופולוגי,
סדר את התנועות לפי המיון הטופולוגי וקבל תזמון סדרתי S'.
אפשר לראות ש- S ו-S' שקולות קונפליקט: מבצעות אותן הפעולות
ומסכימות על סדר הביצוע על פעולות מתנגשות.

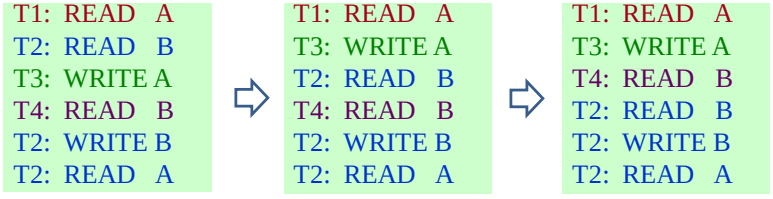
בר-סידור קונפליקט ⇦ בר-סידור מבט

בר-סידור קונפליקט ⇦ בר-סידור מבט

- טענה 1: אם תזמון S' מתקבל מתזמון S על ידי החלפה של שתי פעולות סמוכות שאינן מתנגשות אזי S ו-S' שקולי מבט (ושקולי קונפליקט).
  - קל לבדוק את המקרים השונים
- טענה 2: אם תזמון S' מתקבל מתזמון S על ידי סדרה סופית של החלפות של זוג פעולות סמוכות שאינן מתנגשות אזי S ו-S' שקולי מבט.
  - הוכחה באינדוקציה על מספר ההחלפות, בהסתמך על טרנזיטיביות של שקילות
- טענה 3: אם תזמון S' ותזמון S שקולי קונפליקט אזי ניתן להגיע ל-S מ-S' על ידי סדרה סופית של החלפות של זוגות פעולות סמוכות שאינן מתנגשות.
  - הוכחה בהמשך

בר-סידור קונפליקט ⇦ בר-סידור מבט

<u>טענה 3</u>: אם תזמון S' ותזמון S שקולי קונפליקט אזי ניתן להגיע ל-S מ-S' על ידי סדרה סופית של החלפות של זוגות פעולות סמוכות שאינן מתנגשות.



כל עוד יש זוג פעולות סמוכות שהסדר שלו שונה מסדרו ב-S', החלף

בר-סידור קונפליקט ⇦ בר-סידור מבט

<u>טענה 3</u>: אם תזמון S' ותזמון S שקולי קונפליקט אזי ניתן להגיע ל-S מ-S' על ידי סדרה סופית של החלפות של זוגות פעולות סמוכות שאינן מתנגשות.

כל עוד יש זוג פעולות סמוכות שהסדר שלו שונה מסדרו ב-S', החלף

- כל תזמון ביניים הוא שקול קונפליקט ל-S' ולכן זוג פעולות סמוכות שסדרו בתזמון הביניים שונה מסדרו ב-S' אינו פעולות מתנגשות. (שקילות קונפליקט זו הסכמה על סדר הפעולות המתנגשות). לכן, כל עוד לא הגענו ל-S', נוכל למצוא זוג פעולות סמוכות שיש להחליף.
- כל החלפה יוצרת סדרת ביניים קרובה יותר ל-S' (קרבה כפי שהוגדרה בפרק על מיון), לכן נעצור לאחר מספר סופי של החלפות.

בר-סידור קונפליקט ⇦ בר-סידור מבט

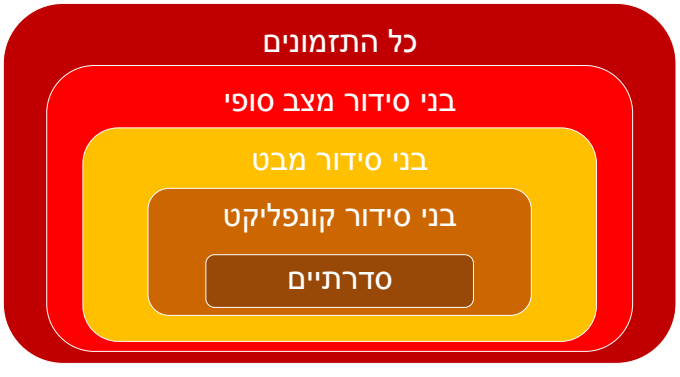
<u>סיכום:</u>

- אם תזמון S בר סידור קונפליקט אזי ישנו תזמון סדרתי 'S שקול קונפליקט ל-S.
- על פי טענה 3, ישנה סדרת החלפות סופית, של פעולות סמוכות שאינן מתנגשות, שמובילה מ-S ל-'S.
- על פי טענה 2, S שקול מבט לתזמון הסדרתי 'S.

בר-סידור קונפליקט ⇦ בר-סידור מבט

ומכיוון שבר-סידור מבט ⇦ בר-סידור מצב סופי

נובע שבר-סידור קונפליקט ⇦ בר-סידור מצב סופי



בר-סידור מבט ⇏ בר-סידור קונפליקט

דוגמה 1: פעולות מיותרות

T1	T2
WRITE x	
	WRITE x
WRITE x	

דוגמה 2: כתיבות עיוורות

T1	T2	T3
WRITE A		
	WRITE A WRITE B	
WRITE B		
		WRITE A WRITE B

אפשר להוכיח שאם S תזמון בר-סידור מבט שאינו מכיל פעולות מיותרות או כתיבות עיוורות אז S בר-סידור קונפליקט

סיכום ביניים

תזמון	מקביליות	נכונות על פי ACID	אכיפה
מקבילי	כן	לא	---
בר-סידור מבט	כן	כן	קשה
בר-סידור קונפליקט	כן	כן	קלה (בעזרת מנעולים)
סדרתי	לא	כן	קלה

נבטיח בר-סדרתיות על-ידי בר-סדרתיות קונפליקט

טיפול בקונפליקטים

נביט בתזמון חלקי, וכיצד הוא עשוי להתפתח:

T1	T2
BEGIN	BEGIN
READ A	
	READ A
	READ B
	READ C
	WRITE TO C
WRITE TO A	
END	
	WRITE TO A

T1	T2
BEGIN	BEGIN
READ A	
	READ A
	READ B
	READ C
	WRITE TO C
WRITE TO A	
END	
	WRITE TO B

איתור קונפליקטים מתנגשים

T1	T2
BEGIN	BEGIN
READ A	
	READ A
	READ B
	READ C
	WRITE TO C
WRITE TO A	
END	
	WRITE TO A

הבקר יאשר את ביצועה של תנועה T1 עד לסיומה המוצלח.

כמו כן, הבקר יאשר את תחילת ביצועה של תנועה T2 עד לשלב האחרון שלה

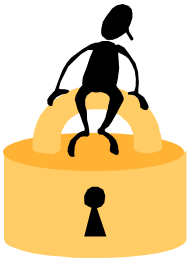
- ויפסול אותה רק אם נוצר מעגל בגרף הקונפליקטים.

במקרה כזה, צריך לעשות ABORT ל T2, ולשחזר את מצב הנתונים לפני הכתיבות שלה.

- פעולת השחזור יקרה, ורוצים להקטין את הסיכויים לעשות ABORT.

מנעולים

- מבני נתונים ופרוטוקול שנועדו למנוע מראש יצירת קונפליקטים בין תנועות.
  - הצהרת כוונות לפני גישה לאובייקט תאפשר למנוע היווצרות מעגלים בגרף.
- כמו כן, נעילה נכונה יכולה לאפשר גם ביצוע פעולות אטומיות בו-זמנית.
  - אין מניעה ששתי תנועות תקראנה אותו אובייקט, אך אסור לתת לתנועה לקרוא או לכתוב אובייקט בזמן שהאובייקט נכתב ע"י תנועה אחרת.



- את המניעה אפשר לעשות ברמת גרעיניות (granularity) שונה:
  - סקטור, בלוק
  - רשומה
  - צומת
  - קובץ (עץ B)
  - דיסק

fine
עדינה

coarse
גסה

מנעולים (המשך)

- נבנה טבלה שבה לכל אובייקט נציין אם תנועה נעלה אותו לקריאה או לכתיבה.
- אם האובייקט נעול ע"י תנועה T (הסיבית המתאימה "דלוקה"), נאמר ש-T **שמה מנעול** על האובייקט.

שני סוגי מנעולים:

- SLOCK← – מנעול קריאה (Shared lock)
- XLOCK← – מנעול כתיבה (eXclusive lock)

כיוון שכמה תנועות יכולות לקרוא בו-זמנית,
אפשר לשים על אובייקט יותר ממנעול קריאה אחד.

כללי שימוש במנעולים

תנאי גישה:

- מותר לתנועה לכתוב אובייקט רק לאחר שהיא שמה עליו מנעול כתיבה.
- מותר לתנועה לקרוא אובייקט רק לאחר שהיא שמה עליו מנעול קריאה או כתיבה.

תנאי נעילה:

- לתנועה אסור לנעול (במנעול נוסף) אובייקט שעליו יש מנעול כתיבה.
- לתנועה אסור לשים מנעול כתיבה על אובייקט שיש עליו מנעול (מכל סוג).

✓אבל, על אותו אובייקט אפשר לשים מספר כלשהו של מנעולי קריאה.

- כל תנועה תשים מנעול לפני קריאה/כתיבה ותשחרר אותו לאחר ביצוע הפעולה.
- התנועה חייבת לשחרר את כל המנעולים שלה לפני סיומה.

הפרוטוקול:

✓משתמש במנעולים כדי למנוע קונפליקטים (אין מעגלים בגרף)

✓לפעמים מונע לחלוטין גישה של שתי תנועות לאותו אובייקט (אין קשתות בגרף)

מימוש מנעולים

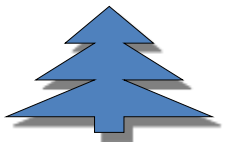
תלוי ברמת הגרעיניות

❖אובייקטים גדולים (קבצים):

- נבנה טבלת ערבול של כל האובייקטים
- עבור כל אובייקט נכתוב את ה-id של התנועות שנעלו אותו וסוגי המנעולים
- הכניסות בטבלה הן בעצמן אזורים קריטיים, המוגנים ע"י סמפורים

❖אובייקטים קטנים (צמתים, רשומות):

- נשמור את אינפורמציית הנעילה של הצומת בצומת עצמו,
- או בתמונה שלו,
- או במקרה של עץ –באביו



נעילה דו-שלבית (2 Phase Locking)

פרוטוקול המבטיח שכל התזמונים יהיו בני-סידור קונפליקט

```
slock x
xlock y
...
slock w
                 } פאזת הנעילות
...
unlock x
unlock w
unlock y
                 } פאזת שחרורים
```

תנועה תעמוד בדרישות 2PL אם ורק אם

- מתקיימים תנאי הנעילה והגישה
- כל הנעילות מופיעות לפני כל השחרורים

כלומר, בפאזה הראשונה רק נועלים,
ובפאזה השנייה רק משחררים.

משפט: אם כל התנועות בתזמון מקיימות את דרישת 2PL,
אז התזמון הוא בר-סידור קונפליקט.

הוכחת משפט הנכונות של 2PL

	T	T'
1:	xlock x	
2:	write x	
3:		xlock y
4:	unlock x	
5:		unlock y

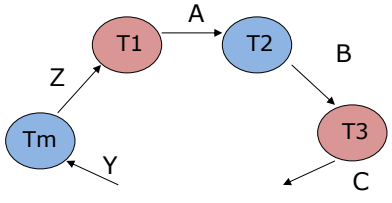
נוכיח כי בגרף הקונפליקט אין מעגלים.

לתנועה T בתזמון S, U(T) הוא המספר הסידורי של פעולת ה-unlock הראשונה ב T

למה: אם $T \rightarrow T'$ אז $U(T) < U(T')$

<u>הוכחת המשפט מהלמה</u> בדרך השלילה:

אם יש מעגל $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_m \rightarrow T_1$



נובע מהלמה ש

$U(T_1) < U(T_2) < \ldots < U(T_m) < U(T_1)$

זאת אומרת $U(T_1) < U(T_1)$

סתירה!

משפט הנכונות של 2PL: הוכחת הלמה

למה: אם $T \rightarrow T'$ אז $U(T) < U(T')$

נניח שיש קונפליקט write-write (ההוכחה לסוגי קונפליקט אחרים זהה),
אז יש משתנה x שנכתב על-ידי T ואחר-כך על ידי T':

T	T'
XLOCK x	
WRITE x	
UNLOCK y	
UNLOCK x	
	XLOCK x
	WRITE x
	UNLOCK z
	UNLOCK x

- לפני ש- T כתבה ל- x היא שמה XLOCK על x.
- לפני ש- T' כתבה ל- x היא שמה XLOCK על x.
- כיוון ש- T כתבה לפני T', T שחררה את המנעול לפני ש- T' שמה את המנעול שלה.
- כיוון ש- T' מקיימת 2PL, משחררת אחרי נעילת x.

לכן $U(T) < U(T')$.

בעיה עם מנעולים: חבק (Deadlock)

דוגמה: נסתכל על שתי התנועות:

T1	T2
…	…
XLOCK B	XLOCK A
XLOCK A	XLOCK B
…	…

אם נריץ את התזמון הבא:

T1	T2
…	…
XLOCK B	
	XLOCK A
	XLOCK B
XLOCK A	
…	…

ניתקע!

פתרונות לבעיית החבק

1. <u>מניעה א:</u>
נגדיר סדר בין המנעולים וכל התנועות תבקשנה את המנעולים לפי סדר זה.
(אם המנעול תפוס, נחכה עד שהאובייקט ישוחרר)

2. <u>מניעה ב:</u>
נדאג לקבל את כל המנעולים בתחילת התנועה,
ואם אי אפשר – התנועה תיעצר (wait) או תבוטל (abort).
Conservative 2PL: כל בקשות המנעולים וקבלתם נעשות בתחילת ריצת התנועה לפני כל פעולות הקריאה והכתיבה.

3. <u>גילוי</u> (detection):
מדי פעם נבדוק אם יש חבק.
אם כן, נבטל תנועות ונשחרר את המנעולים שתפסו.

שאלה: כמה ואלו תנועות נבטל?

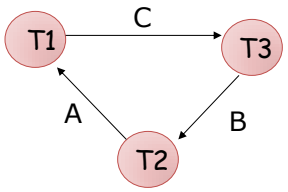
גילוי חבק

נבנה גרף מכוון, שצמתיו מתאימים לתנועות,
ונשים קשת בין T1 ל-T2 אם T1 מחכה למנעול ש-T2 מחזיק.

יש חבק אם ורק אם בגרף יש מעגל מכוון.

<u>דוגמה:</u>

T1	T2	T3
XLOCK A		
	XLOCK B	
		XLOCK C
SLOCK C		
	XLOCK A	
		SLOCK B



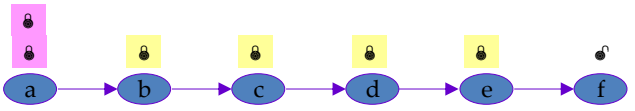
מתי מבצעים את הבדיקה?

מנעולים מגבילים את המקביליות

נעילת אובייקט גדול (עץ B או רשימה מקושרת) עלולה לעכב ללא צורך מספר רב של תנועות, וכך לפגוע בביצועי המערכת.

בעיות אלה נפתרות בעזרת פרוטוקולים סמנטיים, אשר מותאמים באופן ספציפי למבנה הנתונים ולפעולות עליו.

דוגמה על עץ מנוון (רשימה מקושרת)



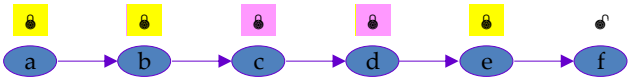
רק תנועה אחת יכולה להתקדם על הרשימה

בקרת מקביליות על עצים

פרוטוקול העץ (עם מנעולי כתיבה)

- חוץ מהצומת הראשון שנועלים, ניתן לנעול צומת רק אם מחזיקים מנעול על אביו.
- תנועה לא נועלת אותו צומת פעמיים.

דוגמה על עץ מנוון (רשימה מקושרת)

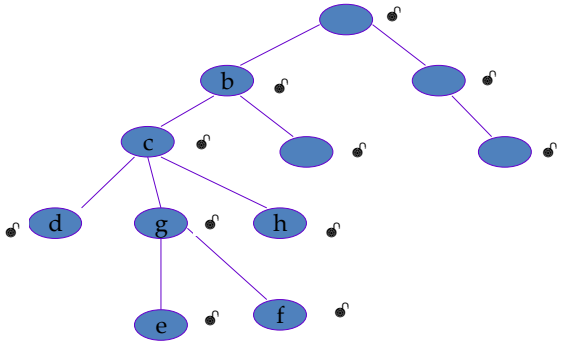


יד-אחר-יד (hand-over-hand) משמש ב java.util.List

הערה: לא מקיים 2PL.

דוגמא לפרוטוקול העץ (על עץ)

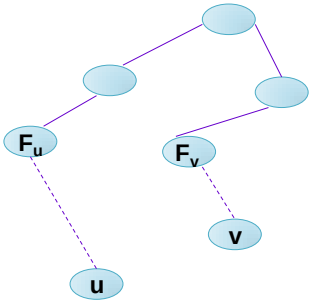
```
LOCK b
LOCK c
LOCK d
UNLOCK d
LOCK g
UNLOCK b
LOCK e
LOCK f
UNLOCK g
UNLOCK f
UNLOCK e
UNLOCK c
```



הערה: התנועות מתקדמות על העץ בגלים.

פרוטוקול העץ מבטיח בר-סדרתיות קונפליקט

יהי H תזמון עם תנועות $T_{n-1}, \ldots, T_1, T_0$.



למה 1: אם שני צמתים u,v נגעלו ע"י תנועה , אזי T נעלה (מתישהו) את כל הצמתים על המסלול שביניהם.

<u>הוכחה</u>: אחרת, יש אב קדמון F_v של v ש- T נעלה אך לא נעלה את אביו,
ואב קדמון F_u של u ש- T נעלה אך לא נעלה את אביו,
ו- $F_u \neq F_v$ (ז"א אחד מהם אינו הצומת הראשון)
סתירה לכך שרק בצומת הראשון ש-T נעלה היה האב לא נעול.

פרוטוקול העץ מבטיח בר-סדרתיות קונפליקט

נסמן ב- $LOCK(T_i)$ את קבוצת הצמתים שתנועה T_i נעלה (אי פעם).

נגדיר יחס ➔ בין התנועות:

T_i ➔ T_j אם קיים צומת שנעל ע"י שתיהן: $v \in LOCK(T_i) \cap LOCK(T_j)$
כך שתנועה T_j נעלה את הצומת v לפני T_i.

בפרט, זה אומר ש $LOCK(T_i) \cap LOCK(T_j)$ אינו ריק.

נסמן ב- $FIRST(T)$ את הצומת הראשון שתנועה T נועלת.

למה 2: כל הצמתים שתנועה T נועלת הם צאצאים של $FIRST(T)$.

למה 3: אם T_i ➔ T_j אז $FIRST(T_i)$ צאצא של $FIRST(T_j)$, או להיפך.

פרוטוקול העץ מבטיח בר-סדרתיות קונפליקט

למה 4: היחס ➔ הוא אנטי-סימטרי (לא יתכן גם T_i ➔ T_j וגם T_j ➔ T_i).

<u>הוכחה:</u> בדרך השלילה.

נסתכל על $v_i, v_j \in LOCK(T_i) \cap LOCK(T_j)$

כך ש-T_i נועלת את v_i לפני T_j ו- T_j נועלת את v_j לפני T_i.

מלמה 1 נובע ששתי התנועות נעלו את כל המסלול בין v_i ל- v_j.

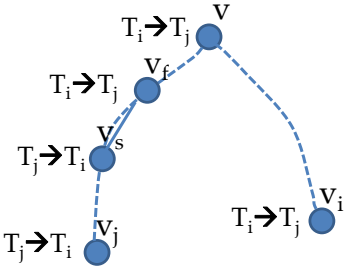
יהי v האב הקדמון המשותף הנמוך ביותר ל-v_i,v_j.

נניח, למשל, ש T_i נעלה את v לפני T_j.

על המסלול בין v ל- v_j קיים צומת v_s

כך ש- T_j נעלה אותו לפני T_i,

אך את אביו, v_f, T_i נעלה קודם.



פרוטוקול העץ מבטיח בר-סדרתיות קונפליקט

למה 4: היחס ➔ הוא אנטי-סימטרי (לא יתכן גם T_i ➔ T_j וגם T_j ➔ T_i).

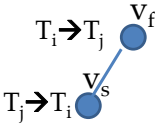
התזמון S מכיל את הפקודות הבאות (ונוספות) בסדר הבא:

```
LOCK_i  v_f
UNLOCK_i  v_f
LOCK_j  v_f
```

לפני פעולת $LOCK_j\ v_f$ יש פעולת $UNLOCK_i\ v_f$.

```
LOCK_j  v_s
LOCK_i  v_f
LOCK_i  v_s
```

מכאן, שיש עוד פעולת $LOCK_i\ v_f$ לפני $LOCK_i\ v_s$.



אבל אסור לנעול צומת פעמים

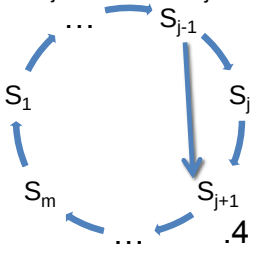
סתירה!

⇦ אם T_i ➔ T_j, אז לכל $v \in LOCK(T_i) \cap LOCK(T_j)$,
T_j נעלה את v לפני T_i

פרוטוקול העץ מבטיח בר-סדרתיות קונפליקט

למה 5: היחס ➔ הוא חסר מעגלים (לא ייתכן S_1 ➔ … ➔ S_m ➔ S_1)

<u>הוכחה:</u> נניח שיש מעגל באורך m. בגלל למה 4, $m > 2$.
תהי S_j תנועה כך ש $v_j = FIRST(S_j)$ הוא הנמוך ביותר בעץ.
לכן, v_j הוא צאצא גם של $FIRST(S_{j-1})$ וגם של $FIRST(S_{j+1})$.
כיון ש S_{j-1} ➔ S_j, קיים צומת $v \in LOCK(S_j) \cap LOCK(S_{j-1})$
כיון ש-S_{j-1} נעלה את כל המסלול בין $FIRST(S_{j-1})$ ל-v, S_{j-1} נעלה את v_j.
מאותו נימוק גם S_{j+1} נעלה את v_j.
שלוש התנועות S_{j+1}, S_j, S_{j-1} נעלו את v_j בסדר זה.
על-כן, S_{j+1} ➔ S_{j-1} ויש מעגל באורך m-1,
S_1 ➔ … ➔ S_m ➔ … ➔ S_{j+1} ➔ S_{j-1} ➔ … ➔ S_1,



באופן זה, אפשר להגיע למעגל באורך 2, סתירה ללמה 4.

נכונות פרוטוקול העץ: סיכום

משפט: כל התזמונים של תנועות המקיימות את פרוטוקול העץ הם בני-סידור קונפליקט.

נבנה גרף מכוון G, שצמתיו התנועות וקשתותיו הן ➔.
יהי C גרף הקונפליקט.

$E(C) \subseteq E(G)$

כי אם $T_1 \xrightarrow{WW/v} T_2 \in E(C)$

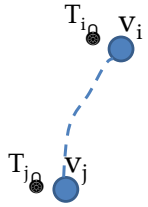
אזי T_1 נעלה את v לפני ש- T_2 נעלה אותו ועל כן, T_1 ➔ $T_2 \in E(G)$.

כיון שב- G אין מעגלים גם ב-C אין מעגלים.
לכן התזמון הוא בר-סידור קונפליקט.

אין חבק בפרוטוקול העץ

הוכחה: נניח בדרך השלילה שיש חבק. נסתכל רק על מעגל באורך 2:

יש צומת v_i שנעול ע"י T_i ותנועה T_j מנסה לנעול אותו,
וצומת v_j נעול ע"י T_j ש- T_i מנסה לנעול אותו.

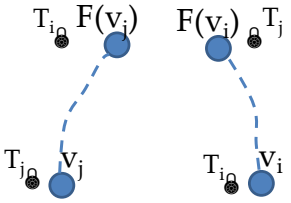


מקרה 1: v_j צאצא של v_i (או v_i צאצא של v_j):
T_j נעלה בעבר את v_i, סתירה לכך שתנועה נועלת צומת לכל היותר פעם אחת.

אין חבק בפרוטוקול העץ

הוכחה: נניח בדרך השלילה שיש חבק. נסתכל רק על מעגל באורך 2:

יש צומת v_i שנעול ע"י T_i ותנועה T_j מנסה לנעול אותו,
וצומת v_j נעול ע"י T_j ש- T_i מנסה לנעול אותו.



מקרה 2: v_j אינו צאצא של v_i או הפוך:

v_i אינו $FIRST(T_j)$, ולכן, T_j מחזיקה מנעול על (v_i) FATHER.

מאותה סיבה, T_i מחזיקה מנעול על (v_j) FATHER.

מלמה 2, לפני ש- T_i נעלה את v_i היא נעלה את אביו. ולכן, T_j ➔ T_i

נימוק זהה לגבי v_j מוכיח ש- T_i ➔ T_j, בסתירה לכך שהיחס ➔ אנטי סימטרי.

פרוטוקול מבוסס חותמות זמן (Timestamps)

כל תנועה T מקבלת **חותמת זמן ייחודית**, TS(T).

תנועה T_2 **מאוחרת** מתנועה T_1 אם $TS(T_1) < TS(T_2)$
בגלל שהחותמות ייחודיות:
$TS(T_1) < TS(T_2)$ או $TS(T_1) > TS(T_2)$

תנועה **מבוטלת** אם היא:

- קוראת אובייקט שנכתב ע"י תנועה מאוחרת יותר.
- מנסה לכתוב לאובייקט שנקרא או נכתב ע"י תנועה מאוחרת יותר.
- קראה נתון שנכתב ע"י תנועה שהופסקה.

דוגמה:

T1	T2
BEGIN (8:00)	
	BEGIN (8:01)
WRITE A	
	READ A
	WRITE B
READ B	
ABORT	
ROLLBACK A	
	ABORT

תזמון של הפרוטוקול הזה הינו **שקול קונפליקט** לתזמון הסדרתי המסודר לפי חותמות הזמן.

60

נכונות כאשר תנועות מופסקות

אם יש פעולות שמופסקות (abort) ביוזמתן או בגלל בקר המקביליות

נבדוק אם התזמון ללא התנועות המופסקות הינו בר-סידור (מצב סופי, מבט, קונפליקט)

מימוש פרוטוקול Timestamps (1)

לכל אובייקט A, נשמור שני משתנים:

- LastW(A) – התנועה המאוחרת ביותר שכתבה ל-A.
- LastWR(A) – התנועה המאוחרת ביותר שכתבה/קראה מ-A.

מההגדרה נובע כי: $TS(LastW(A)) \leq TS(LastWR(A))$

לכל תנועה T נשמור את המשתנה:

- Pred(T) – קבוצת התנועות שכתבו נתונים שתנועה T קראה.

מימוש פרוטוקול Timestamps (2)

- קריאה מ-A ע"י תנועה T:
  אם ((TS(T) < TS(LastW(A אז ABORT(T);
  אחרת — READ A
  הוסף את LastW(A) ל-Pred(T)
  אם T מאוחרת מ-LastWR(A) אזי LastWR(A) = T.
- כתיבה ל- A ע"י תנועה T :
  אם ((TS(T) < TS(LastWR(A אז ABORT(T)
  אחרת — WRITE A
  LastW(A) = LastWR(A) = T.
- סיום תנועה T :
  אם Pred(T) מכילה תנועות שהופסקו אז ABORT(T);
  אם Pred(T) מכילה תנועות שטרם הסתיימו, אזי T תחכה עד שכולן תסתיימנה בהצלחה.
  אם אחת מתנועות אלו מופסקת, גם T תופסק.

נכונות פרוטוקול Timestamps

משפט: יהי S תזמון המקיים את פרוטוקול ה-Timestamps.
אם נסלק מ-S את התנועות שהופסקו אז נקבל תזמון בר-סידור קונפליקט.

הוכחה: אם יש קשת T1 → T2 בגרף הקונפליקט, אז TS(T1) < TS(T2)

אלגוריתם ה-Timestamps עלול לגרום למפולת של הפסקות (cascading aborts)

ניתן למנוע זאת על-ידי strict timestamps:

- בקריאה מ A, המתן לסיום התנועה שכתבה אחרונה ל A
- בכתיבה ל-A, המתן לסיום התנועה שכתבה אחרונה ל-A וכל התנועות שקראו מ A

Snapshot Isolation

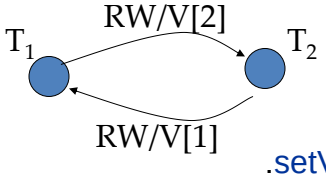
- כל תנועה מקבלת עותק פרטי של מבנה הנתונים (עם timestamp).
- כאשר תנועה מסתיימת היא בודקת עבור כל הנתונים שהיא שינתה אם קיים לגביהם קונפליקט write-write.
  אם כן, התנועה מופסקת (abort) ולא מעדכנת את מסד הנתונים.

האם הפרוטוקול בר-סידור?

דוגמה

- בבי"ח מותר לרופא לקחת חופשה אם מספר הרופאים בתורנות גדול מ k
- נניח שיש מערך Vacation[i] שערכו 1 אם רופא i בחופש.
- תנועה setVacation(doctor i)
  - קרא את כל המערך
  - אם מספר הרופאים שאינם בחופש ≤ k+1, הצב Vacation[i]=1

גרף הקונפליקט



נניח תזמון שבו יש 2 רופאים, k=1
ונריץ במקביל את setVacation(1) ו-setVacation(2).
כיון שאף נתון לא נכתב ע"י שתי התנועות, אף תנועה לא תופסק
והתוצאה היא שהמערך יתאפס וביה"ח יישאר ללא רופאים!

מאפשר תזמונים שאינם בני-סידור קונפליקט, ואפילו לא בני-סידור מבט

רמות בידוד

- השימוש ב-snapshot isolation בשילוב עם מנעולים מאפשר להגדיר רמות בידוד שונות (מהגבוהה לנמוכה):
  - Serializable
  - Repeatable Read
  - Read Committed
  - Read Uncommitted
- ככל שרמת הבידוד גבוהה יותר, פעולות המערכת בטוחות יותר בכך שנמנעות יותר בעיות הנובעות ממקביליות.
  מצד שני, עליה ברמת הבידוד מורידה את רמת המקביליות ואת תפוקת המערכת.

סיכום ביניים

- שימוש במנעולים מאפשר לבקר המקביליות להבטיח כי התזמון המתבצע בפועל הוא בר סידור קונפליקט ולכן בר סידור מבט
- המחיר של השימוש במנעולים:
  - עלות אחזקת מבני הנתונים המתאימים
  - אפשרות לחבק
  - הפסקת תנועות ולעיתים אף מפל של הפסקות
- חותמות זמן ושטחי עבודה פרטיים (snapshot) יכולים להחליף נעילות
- מערכות מודרניות רבות משלבות מנעולים, חותמות זמן ושטח עבודה פרטי כדי לאפשר למשתמש גמישות בהגדרת רמת המקביליות