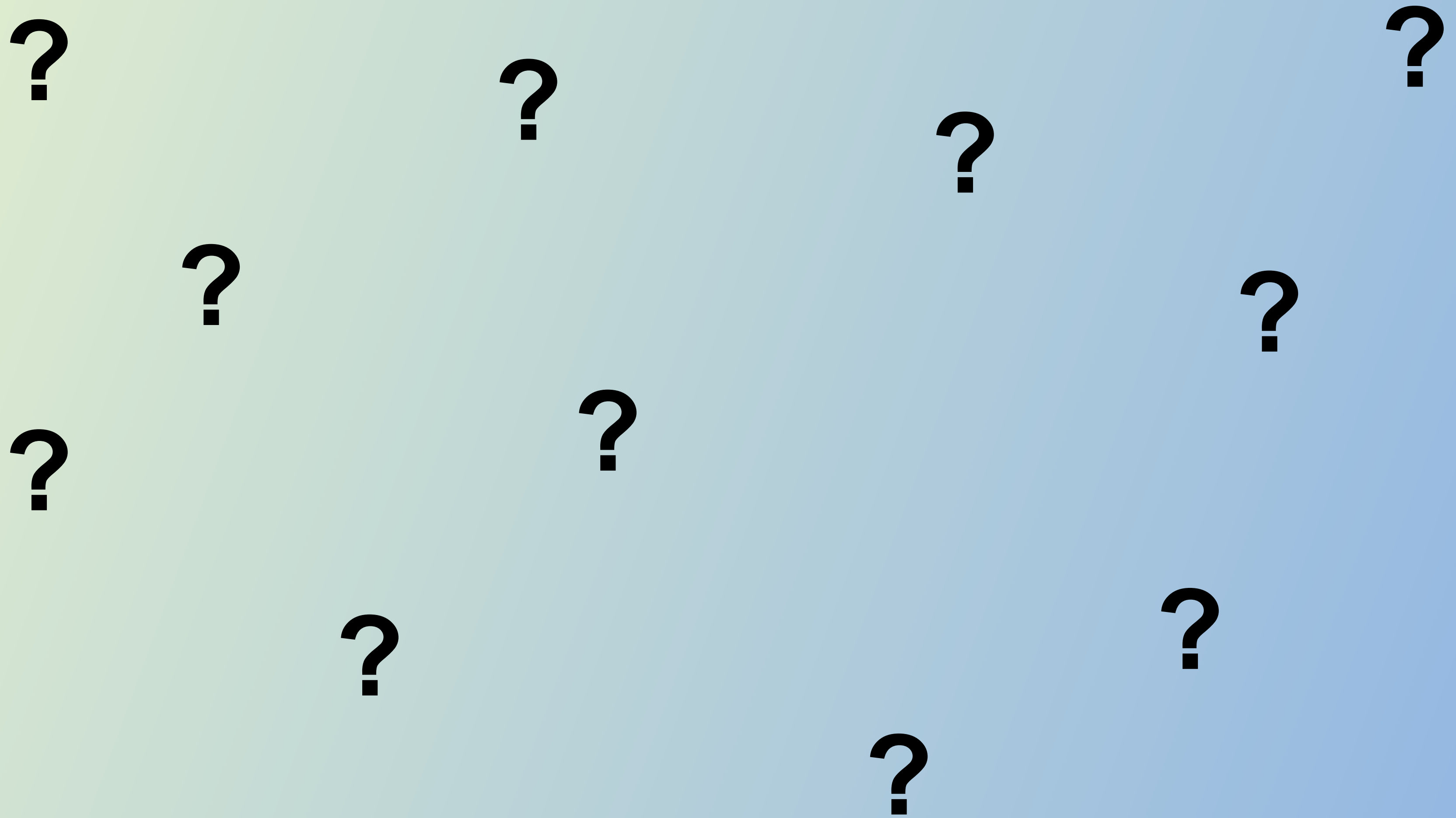
בסיסי נתונים - קורס מתקדם

2024 – תשפ"ד (סמסטר קיץ)

מרצה: רואי זרחיה

שיעור 4

האם יש שאלות משיעור קודם ?



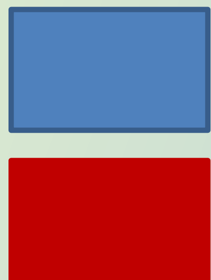
בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים

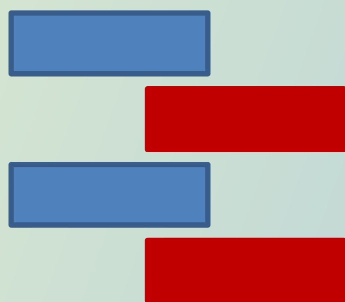
תזמונים

תזמון (schedule) של קבוצת טרנזקציות הוא רצף של פעולות, הכולל את כל הפעולות השייכות לקבוצת הטרנזקציות כך שהתזמון שומר על הסדר שבו מופיעות הפעולות בכל אחת מהטרנזקציות.

תזמון סדרתי (serial schedule) תזמון בו הפעולות השייכות לכל אחת מהטרנזקציות מופיעות **ברצף**, כלומר לא משולבות ביניהן פעולות של טרנזקציות אחרות (כל טרנזקציה חדשה מתחילה רק לאחר שקודמתה הסתיימה).



תזמון לא סדרתי (non-serial schedule) תזמון בו הפעולות השייכות לקבוצת טרנזקציות **משולבות זו בזו**, מכונה גם כ-**תזמון בו-זמני (מקבילי)**.



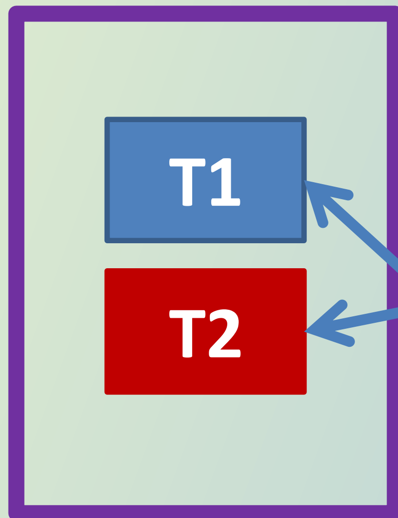
בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים סדרתיים

בשקפים הבאים נראה מספר תזמונים סדרתיים עבור הדוגמא הבאה.

דוגמא: נתון תזמון T_1 ותזמון T_2 שמבצעים העברת כספים במערכת הבנקאית בין חשבון A לחשבון B.

- **טרנזקציה T_1 מעבירה $50\$$ מחשבון A לחשבון B.**
- **טרנזקציה T_2 מעבירה 10% מהיתרה הקיימת בחשבון A לחשבון B.**



*** עבור כל n טרנזקציות, קיימים !n תזמונים סדרתיים שונים.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמון סדרתי – דוגמא 1

T_1 – העבר 50$ מחשבון A ל-B,
T_2 – העבר 10% מחשבון A ל-B.

נסתכל על התזמון הסדרתי הבא <T_1,T_2>, סימון זה מבטא את סדר הרצת הטרנזקציות בתוך התזמון - קודם T_1 ורק לאחר מכן T_2.

ערכי בסיס: B = 2000 ,A = 1000 ; $\sum = 3000$

ניתן לראות שהסכום של שני המשתנים נשמר גם לאחר הרצת הטרנזקציות בצורה סדרתית:

ערכי סיום: B = 2145 ,A = 855 ; $\sum = 3000$

תזמון סדרתי - S_1

T_1	T_2
read(A)	
A = A-50	
write(A)	
read(B)	
B = B+50	
write(B)	
	read(A)
	temp = A*0.1
	A = A-temp
	write(A)
	read(B)
	B = B + temp
	write(B)

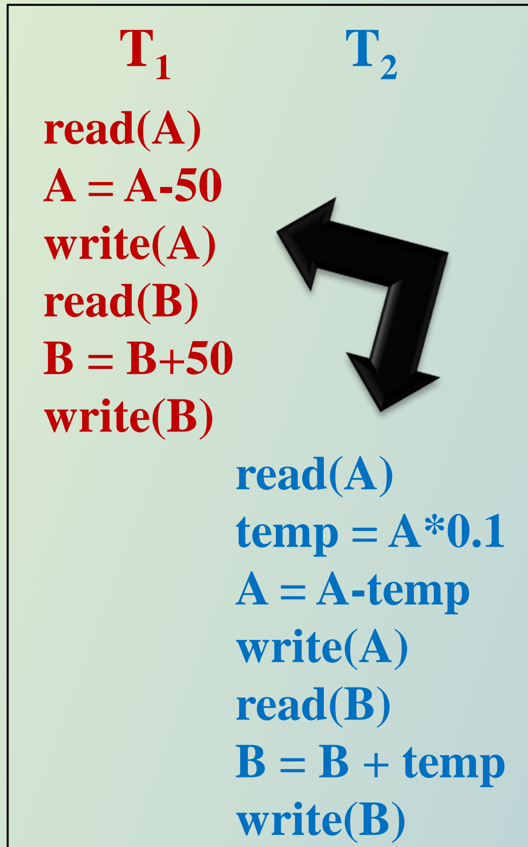
$A_1 = 855 \quad B_1 = 2145$

$\sum = 3000$

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמון סדרתי – דוגמא 1

T_1 – העבר 50$ מחשבון A ל-B,
T_2 – העבר 10% מחשבון A ל-B.



האם גם הרצת הטרנזקציות בתזמון <T2,T1> תשמר את הסכום 3000= ∑ ?

נריץ כעת את תזמון S2 שהוא תזמון הפוך ל S1
ובשקף הבא נראה את התשובה...

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמון סדרתי – דוגמא 2

T_2 – העבר 10% מחשבון A ל-B.
T_1 – העבר 50$ מחשבון A ל-B.

ערכי בסיס: $B = 2000$, $A = 1000$; $\sum = 3000$

כעת, נסתכל על התזמון הסדרתי הבא $<T_2,T_1>$

ניתן לראות שהסכום של שני המשתנים נשמר גם לאחר הרצת הטרנזקציות בתזמון $<T_2,T_1>$ המורץ בצורה סדרתית = (שמירה על עקביות).

ערכי סיום: $B = 2150$, $A = 850$; $\sum = 3000$

S_2 - תזמון סדרתי

T_1	T_2
	read(A)
	temp = A*0.1
	A = A-temp
	write(A)
	read(B)
	B = B + temp
	write(B)
read(A)	
A = A-50	
write(A)	
read(B)	
B = B+50	
write(B)	

$A_1 = 850 \quad B_1 = 2150$

$\sum = 3000$

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים בו-זמניים

למרות שהפיתרון הקל ביותר לאבטחת עקביות הינו לבצע תזמון בצורה סדרתית, ישנם מספר יתרונות לביצוע תזמונים בו-זמניים (**מקביליים**):

- **שיפור ביצועים** והגדלת תפוקת המערכת ע"י הגדלת מספר הטרנזקציות היכולות לרוץ במקביל בכל רגע נתון.
- **ניצול טוב יותר של המעבד והדיסק** שכן בצורה זו הם נמצאים פחות זמן במצב "המתנה" (ממתינים לקבלת משימות לביצוע).
- **מניעת עיכובים מיותרים -** נרצה למנוע מצב שטרנזקציה ארוכה מונעת מטרנזקציה קצרה לרוץ.

** בהרצת מספר טרנזקציות במקביל, עקביות מסד הנתונים יכולה להיפגע.

בדוגמאות הבאות נראה מספר תזמונים במטרה לאתר את הבעיות שיכולות להיגרם בהרצה במקביל.

תזמון בו-זמני (מקבילי) – דוגמא 1

תזמון בו-זמני - C_1

T_1	T_2
read(A)	
A = A-50	
write(A)	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
read(B)	
B = B+50	
write(B)	
	read(B)
	B = B + temp
	write(B)

T_1 – העבר 50$ מחשבון A ל-B,
T_2 – העבר 10% מחשבון A ל-B.

ערכי בסיס: B = 2000 ,A = 1000 ; $\sum = 3000$

האם קיים שוני בין ההרצה של התזמון הבו-זמני (מקבילי) להרצה הסדרתית ?

בשקף הבא נבדוק ונשווה את ערכי המשתנים A, B לאחר ההרצה הסדרתית ולאחר ההרצה הבו-זמנית.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

השוואת תזמונים - דוגמא

T_1 – העבר 50$ מחשבון A ל-B,
T_2 – העבר 10% מחשבון A ל-B.

תזמון סדרתי - S_1

T_1	T_2
read(A)	
A = A-50	
write(A)	
read(B)	
B = B+50	
write(B)	
	read(A)
	temp = A*0.1
	A = A-temp
	write(A)
	read(B)
	B = B + temp
	write(B)

$A_1 = 855 \quad B_1 = 2145$

$A = 1000, \; B = 2000$

העקביות נשמרת:

$A_1 + B_1 = A_2 + B_2$

תזמון בו-זמני - C_1

T_1	T_2
read(A)	
A = A-50	
write(A)	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
read(B)	
B = B+50	
write(B)	
	read(B)
	B = B + temp
	write(B)

$A_2 = 855 \quad B_2 = 2145$

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים בו-זמניים (מקביליים)

לאור העובדה שטרנזקציה היא **יחידה השומרת על עקביות הנתונים**, הרצה סדרתית של טרנזקציות מבטיחה שמירה על עקביות הנתונים, אך האם זהו המצב גם בתזמונים בו-זמניים ?

בדוגמא הקודמת, ניתן לראות שהסכום של שני המשתנים נשמר לאחר הרצת הטרנזקציות גם כאשר הן מורצות בצורה בו-זמנית (ניתן לראות שתוצאת הרצת **תזמון בו זמני C_1** שקולה לתוצאת הרצת **התזמון הסדרתי S_1**).

האם כל התזמונים שמורצים בצורה בו-זמנית* יסתיימו באותם הערכים במשתנים / ישמרו על עקביות ?

נבדוק כעת מספר תזמונים נוספים עבור אותה דוגמא במטרה לבדוק האם העקביות נשמרת מבחינת ערכי המשתנים הסופיים.

* תזכורת: עבור כל n טרנזקציות, קיימים יותר מ !n תזמונים סדרתיים שונים.

תזמון בו-זמני – דוגמא 2

תזמון בו-זמני - C_2

T_1	T_2
read(A)	
A = A-50	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B = B+50	
write(B)	
	B = B + temp
	write(B)

$A_2 = 950 \quad B_2 = 2100$

$\sum = 3050$

T1 (phase I)	T2 (phase I)
1000	
950	
	1000
	100
	900
	→ 900
	2000
→ 950	
2000	
2050	
→ 2050	
	2100
	→ 2100

חץ סגול מסמל כתיבה מהזיכרון ל DB.

ערכי בסיס: A = 1000
B = 2000
∑ = 3000

ניתן לראות שקיבלנו מצב שאינו עקבי וזאת לאור העובדה שקיבלנו שסכום משתני התזמון הבו-זמני גבוה ב-$50 מסכום משתני התזמון הסדרתי שכן תוצאת סכום המשתנים A, B הסופית היא $3050.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים - השוואה

T_1 – העבר 50$ מחשבון A ל-B,
T_2 – העבר 10% מחשבון A ל-B.

S₁ - תזמון סדרתי

T_1	T_2
read(A)	
A = A-50	
write(A)	
read(B)	
B = B+50	
write(B)	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
	read(B)
	B = B + temp
	write(B)

$A_1 = 855 \quad B_1 = 2145$

$\sum = 3000$

$A_i = 1000, \quad B_i = 2000$

העקביות הופרה:

$A_1 + B_1 \neq A_2 + B_2$

C₂ - תזמון בו-זמני לא שקול

T_1	T_2
read(A)	
A = A-50	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B = B+50	
write(B)	
	B = B + temp
	write(B)

$A_2 = 950 \quad B_2 = 2100$

$\sum = 3050$

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים בו-זמניים

הדוגמא האחרונה מראה מה קורה כשהבקרה לתזמונים הסדרתיים מטופלת ע"י **מערכת ההפעלה**, לאור כך, קיימות מספר אופציות הרצה לתזמונים, כולל הרצות בהן מסד הנתונים **מגיע לחוסר עקביות** (כמו בתזמון הבו-זמני C_2).

על מנת להימנע מתרחישים כאלו, **נרצה לדאוג שהבקרה על התזמונים תתבצע ע"י מסד הנתונים ולא ע"י מערכת ההפעלה** שכן תפקיד מסד הנתונים הוא לדאוג לכך שכל התזמונים שרצים בו זמנית לא יגררו מצבים בהם מופרת העקביות.

נוכל למנוע מקרים של הפרת העקביות ע"י כך שנוודא שכל הרצה של תזמונים בו זמנית תחזיר את אותה התוצאה של התזמון הסדרתי וזאת ע"י שימוש בשיטה (מערכת) היודעת להבטיח **שקילות סדרתית** (serializable) – זאת אומרת **הבטחה ששקילות מקבילית תהיה שקולה לזו הסדרתית**.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות סדרתית - serializable

הגדרה: תהיינה T1,T2,...,Tn טרנזקציות. תזמון של טרנזקציות אלו ייקרא **תזמון שקול סדרתית** אם קיים תזמון סדרתי של n טרנזקציות שנותן תוצאה זהה לתזמון זה, אחרת התזמון יהיה **תזמון לא שקול סדרתית.**

בסיס הנתונים חייב לדאוג לבצע בקרה על תזמונים בו זמניים במטרה לשמור על עקביות, ולכן ראשית חשוב להבין אילו תזמונים יבטיחו עקביות ואילו לא:

לשם כך נתמקד בשתי פעולות ספציפיות המתרחשות בתזמונים השונים: פקודת קריאה (**read**) ופקודת כתיבה (**write**).

לאור העובדה שבין פעולת הקריאה לפעולת הכתיבה עבור משתנה מסוים, הטרנזקציה יכולה לבצע שילוב שרירותי כלשהוא של פעולות חישוביות על גבי משתנה זה, אזי מבחינת פרמטר העקביות, הפעולות המשמעותיות בטרנזקציה שנרצה לבדוק מבחינת התזמון הן **פעולות הקריאה והכתיבה**.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות סדרתית - serializable

T_0	T_1
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

בכדי לבדוק האם תזמון מקבילי (בו-זמני) שקול לתזמון סדרתי – ז"א שמתקיימת שקילות סדרתית, אזי **נתמקד רק בפקודות הקריאה והכתיבה** (ראו דוגמא) כיוון **ששאר פעולות החישוב אינן משפיעות** או רלוונטיות לתהליך המקבילי.

ולכן משלב זה ואילך, כך נציין מבנה של תזמון (רק פעולות של קריאה וכתיבה ללא פעולות ביניים).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

1) שקילות התנגשותית

<u>עקרון ההחלפה בין פעולות (swap):</u>

יהי תזמון S, ושתי פעולות עוקבות Ii ו Ij השייכות לטרנזקציות שונות Ti ו Tj .
אם Ii ו Ij **<u>מתייחסות לפריטי מידע שונים</u>** אזי **ניתן להחליף בין Ii ל Ij מבלי להשפיע על תוצאות הפקודות בתזמון**. אך אם Ii ו Ij מתייחסות לאותו פריט מידע אזי סדר ההרצה שלהם חשוב.

לאור העובדה שאנו מתייחסים בתזמון רק לפעולות הקריאה והכתיבה אזי קיימות 4 אפשרויות בהן נוכל להיתקל בטרנזקציות השונות:

1) Ii = read(Q) , Ij = read(Q)

2) Ii = read(Q) , Ij = write(Q)

3) Ii = write(Q) , Ij = read(Q)

4) Ii = write(Q) , Ij = write(Q)

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות התנגשותית

(1 Ii = read(Q) , Ij = read(Q)

ניתן לראות ששתי הפעולות קוראות את המשתנה Q אחת אחר השנייה – במקרה זה סדר הקריאה אינו משנה כיוון ש**אותו ערך נקרא ע"י שתי הטרנזקציות** *Ti* ו*Tj* ללא קשר לסדר הרצתם.

T_i	T_j
read(Q)	
	read(Q)

(2 Ii = read(Q) , Ij = write(Q)

אם Ii מתרחש לפני Ij אזי Ti לא קורא את הערך של Q שנכתב ע"י Tj בפעולה Ij. אך אם Ij מתרחש לפני Ii אז Ti קורא את הערך של Q שנכתב ע"י Tj ומכאן סדר הקריאה של Ii ו Ij חשוב **ומשפיע**.

T_i	T_j
read(Q)	
	write(Q)

(3 Ii = write(Q) , Ij = read(Q)

סדר הפעולות של Ii ו Ij חשוב **ומשפיע** (לפי אותו מקרה שתואר באפשרות הקודמת).

T_i	T_j
write(Q)	
	read(Q)

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות התנגשותית

4) I_i = write(Q) , I_j = write(Q)

T_i	T_j
write(Q)	
	write(Q)

מכיוון ששתי הפעולות הן פעולות כתיבה, סדר הפעולות לא משפיע על Ti או Tj – אבל, האם הוא יכול להשפיע על ההמשך ?

ובכן, חשוב לדעת שהערך (Q) שיטען בפעולת הקריאה הבאה של התזמון S עלול להיות שונה בכל תרחיש וזאת כיוון שרק ערך פעולת הכתיבה האחרונה נשמר במסד הנתונים (הכתיבה הקודמת תידרס).

ולכן אם אין פקודת כתיבה נוספת write(Q) אחרי Ii ו Ij בתזמון S אזי סדר הפקודות של Ii ו Ij משפיע ישירות על התוצאה הסופית של Q במסד הנתונים בהרצת תזמון S ולכן **סדר הכתיבה משפיע**.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות התנגשותית - מסקנה

הצגנו שקיימות 4 אפשרויות בהן נוכל להיתקל בטרנזקציות השונות:

1) **Ii = read(Q) , Ij = read(Q)**
2) **Ii = read(Q) , Ij = write(Q)**
3) **Ii = write(Q) , Ij = read(Q)**
4) **Ii = write(Q) , Ij = write(Q)**

אך לאחר ביצוע ניתוח מעמיק, ראינו שרק במקרה שיש 2 פעולות קריאה רצופות (מקרה ראשון), הסדר של הרצת הפעולות **אינו משפיע** על התוצאה.

מסקנה: **נאמר ש Ii ו Ij הן פעולות מתנגשות אם הן פקודות של תזמונים שונים על גבי אותו פריט מידע ולפחות אחת מפקודות אלו היא פקודת כתיבה** (שכן שילוב של פעולת כתיבה בתזמון עלולה לגרום לתוצאות לא עקביות ומכאן חשוב לבדוק את המקרה).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות התנגשותית

בכדי להמחיש את רעיון הפעולות המתנגשות, נסתכל על טרנזקציה T_1 ועל טרנזקציה T_2 בתזמון S_1.

הפעולה write(A) של T_1 מתנגשת עם פקודת ה read(A) של T_2 (שתיהן ניגשות לאותו פריט מידע A)

למרות זאת פעולת write(A) של T_2 לא מתנגשת עם פעולת read(B) של T_1 כי שתי הפעולות ניגשות לשני פרטי מידע שונים.

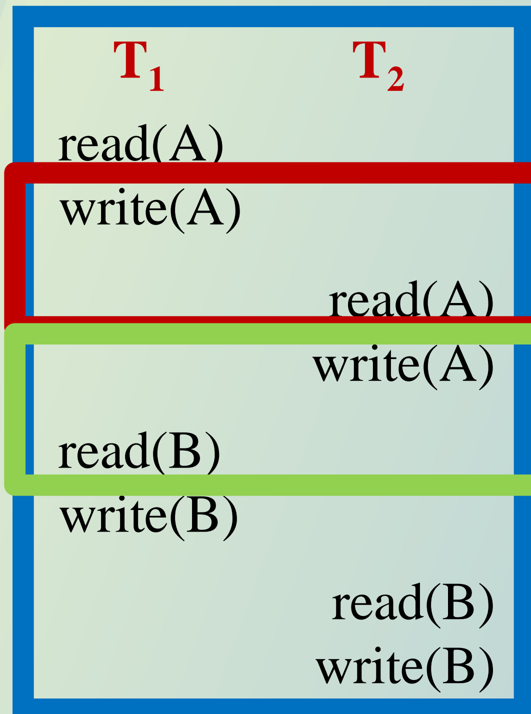
ומכאן נוכל לשרטט כעת מסקנה לגבי החלפות בין פעולות שניתן לבצע ושלא יגררו התנגשויות.

S1

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

שקילות התנגשותית

<u>ביצוע החלפה</u>: יהיו Ii ו Ij פעולות עקביות של תזמון S1, **אם Ii ו Ij הן פעולות של טרנזקציות שונות כאשר Ii ו Ij אינן מתנגשות אזי ניתן להחליף את סדר הפעולות** של Ii ו Ij במטרה ליצור תזמון חדש S_2.



יש התנגשות כי סדר ביצוע הפעולות ביניהם על אותו פריט מידע עלול ליצור תוצאות שונות **ולכן אין אפשרות לבצע החלפה**

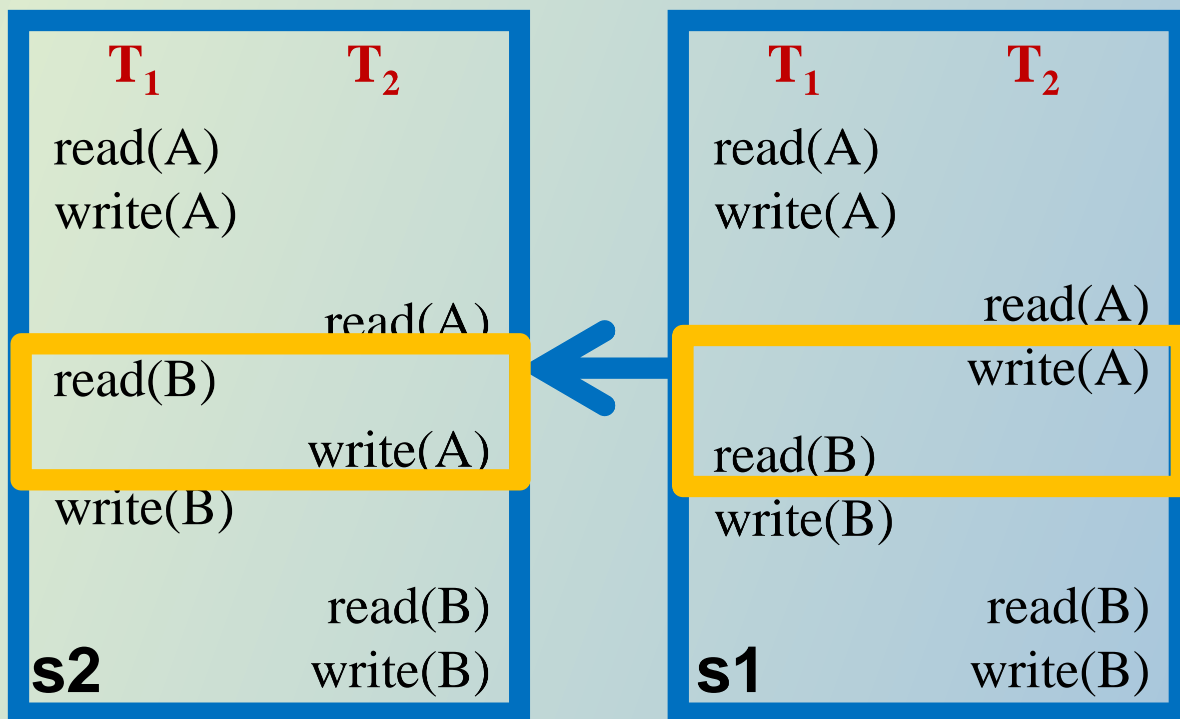
אין התנגשות כי פונים לפריטי מידע שונים והתוצאה עם / ללא החלפה תהיה זהה **ולכן ניתן להחליף בין הפקודות**

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

שקילות התנגשותית

S_1 יהיה **תזמון שקול** ל S_2 כיוון שהם שקולים בכל הפעולות המופיעות בשניהם באותו הסדר חוץ מאשר פעולות Ii ו Ij שהרי הסדר שלהם אינו משנה.

מכיוון שפעולת write(A) של T_2 אינה מתנגשת עם פעולת read(B) של T_1, נוכל **להחליף בין הפעולות** בכדי ליצור תזמון שקול S_2 כך ששני תזמונים אלו מסתיימים באותו מצב סופי והעקביות נשמרת.



שקילות התנגשותית

עבור התזמון השקול S_2 ניתן להמשיך ולבצע החלפות בין הפעולות הלא מתנגשות בטרנזקציות השונות:

- החלפת פעולת read(B) של T_1 עם פעולת read(A) של T_2.
- החלפת פעולת write(B) של T_1 עם פעולת write(A) של T_2.
- החלפת פעולת write(B) של T_1 עם פעולת read(A) של T_2.

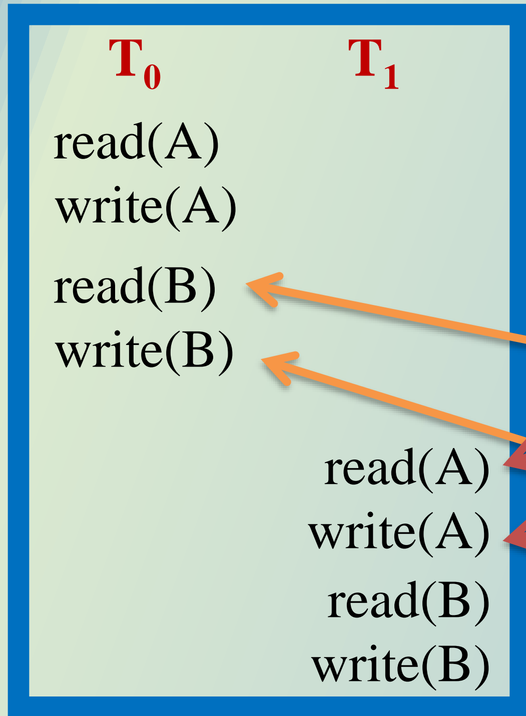
s2

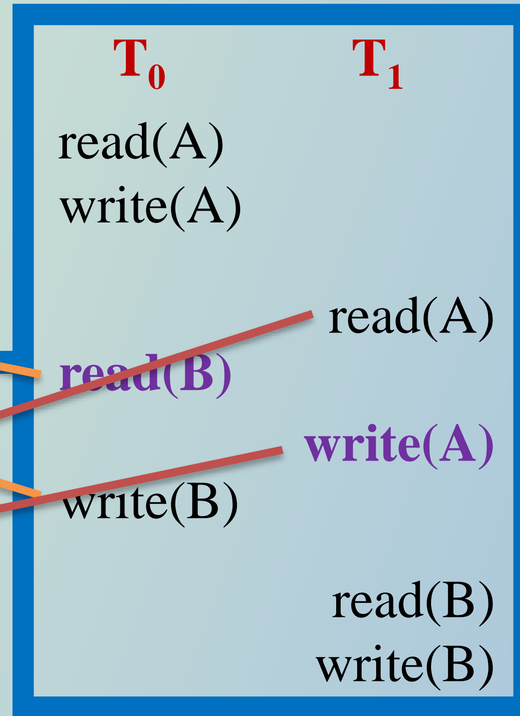
T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

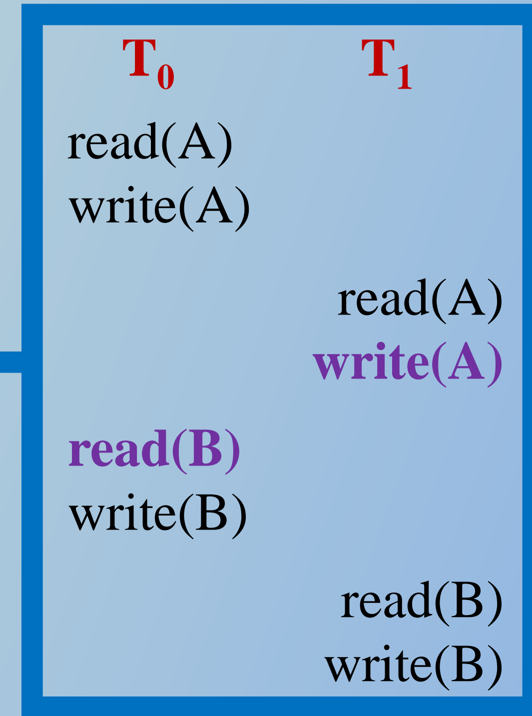
לאור העובדה שפעולות אלו אינן מתנגשות, נקבל בסיום כל ההחלפות הנ"ל תזמון השקול ל S_2 (וגם כמובן תזמון השקול ל S_1 המקורי) – **נראה המחשה...**

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים – שקילות סדרתית





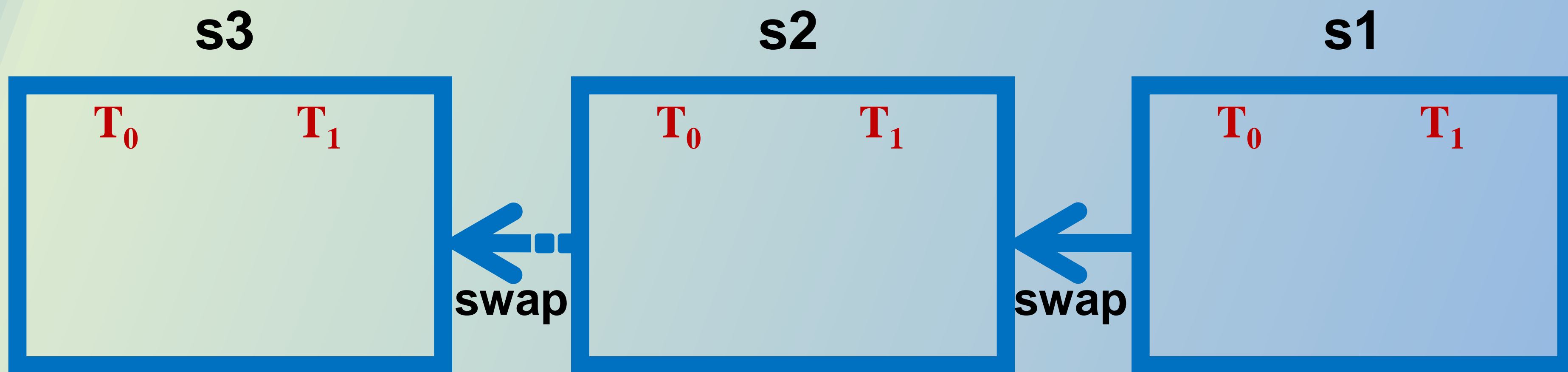


תוצאת כל פעולות ההחלפות תיתן תזמון סדרתי <T_0,T_1> (S_3), ומכאן הראנו ש **S_1 הוא תזמון השקול לתזמון סדרתי S_3 ("שקול סדרתית").**

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תזמונים שקולים סדרתית

שקילות זו מוכיחה שלמרות המצב ההתחלתי השונה, תזמון S_1 המהווה תזמון בו-זמני, יסתיים באותה הצורה בדיוק כמו התזמון הסדרתי S_3.



הגדרה: אם תזמון S יכול להשתנות לתזמון 'S ע"י סדרת פעולות החלפה שאינן מתנגשות, נאמר ש S ו 'S הם **תזמונים שקולים סדרתית**.

*** שקילות זאת בספרות מצוינת גם ע"י המינוח "**שקילות התנגשותית**".

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

סידורים סדרתיים של תזמונים בו-זמניים – סיכום ביניים:

פעולות מתנגשות (conflicting operations) שתי פעולות עוקבות Ii ו Ij השייכות לטרנזקציות שונות Ti ו Tj הן פעולות מתנגשות אם הן מתייחסות לאותו פריט מידע ואם לפחות אחת מהן היא פעולת כתיבה.
[סדר ביצוע פעולות מתנגשות עשוי להשפיע על התוצאה הסופית]

שקילות התנגשותית (conflict equivalence) שני תזמונים S ו `S הם שקולי התנגשות אם ניתן לעבור בניהם על ידי סדרה של החלפות של פעולות שאינן מתנגשות.
[מעבר ע"י החלפת סדר פעולות ללא השפעה על התוצאה הסופית].

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים לתמיכה בטרנזקציות מקביליות

פרוטוקולים לתמיכה בבקרת בו-זמניות

ראינו שאחת התכונות החשובות של טרנזקציות היא בידוד (isolation).

כאשר מספר טרנזקציות מורצות במקביל יתכן ותכונת הבידוד לא תשמר. בכדי לשמור על תכונה זו, המערכת חייבת לבצע בקרה של האינטראקציה בין הטרנזקציות הרצות במקביל, בקרה זו מושגת ע"י פרוטוקול לבקרת בו-זמניות.

קיימים מספר פרוטוקולים לתמיכה בבקרת בו זמניות בהם נוכל להשתמש כאשר נריץ מספר טרנזקציות במקביל.

המטרה העיקרית אותה הצגנו שבעקבותיה ניתן הפתרון של הרצת טרנזקציות במקביל הוא **למנוע ריצה סדרתית בעלת ביצועים חלשים** וחוסר ניצול של המשאבים הקיימים.

פרוטוקולים לתמיכה בבקרת בו-זמניות

אנו נעסוק בשלושה פרוטוקולים עיקריים* לתמיכה בבקרה בו-זמנית:

א. פרוטוקול מבוסס נעילה

ב. פרוטוקול תגי זמן

ג. פרוטוקול מבוסס אימות

מטרת פרוטוקולים אלו הינה לאפשר עבודה מקבילית תוך הבטחה שהתזמונים המורצים ניתנים לארגון סדרתי או לארגון תצפיתי.

*השוני הינו ברמת ה trade-off בין רמת המקביליות המתאפשרת לרמת התקורה הנדרשת מהרצתם.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

א) פרוטוקולים מבוססי נעילה

אחת השיטות להבטחת שקילות סדרתית היא לדאוג שגישה לפרטי מידע ע"י טרנזקציות שונות תתבצע בצורה בלעדית, ז"א שכאשר טרנזקציה T1 ניגשת לפריט מידע, אף טרנזקציה אחרת לא יכולה לשנות את פריט מידע זה.

השיטה הנפוצה ביותר למימוש דרישה זו היא לאפשר לטרנזקציה לגשת לפריט מידע רק כאשר הטרנזקציה ביצעה **נעילה** (**lock**) על פריט מידע זה.

בכדי להמחיש מקרים של שימוש בנעילות, נגדיר תחילה מושגי בסיס מתחום זה ואת סוגי הנעילות השונים.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה - מושגים

- **READ** - קריאת נתונים ממסד נתונים לזיכרון.
- **OPERATION** - כל פעולת חישוב כלשהי.
- **WRITE** - כתיבת נתונים מהזיכרון למסד הנתונים.
- **LOCK** - חסימת גישה לפריט נתון למול תוכניות אחרות שרצות במקביל.
- **UNLOCK** - שחרור הגישה לפריט נתון, כך ששאר התכניות שרצות במקביל תוכלנה לגשת אליו.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה - מושגים

<u>סוגי הנעילות:</u>

נעילה משותפת (Shared lock) - אם טרנזקציה Ti קיבלה נעילה משותפת (המסומנת ב-**S**) על פריט מידע Q, אזי Ti רשאית לקרוא (פקודת Select) פריט זה אך לא לכתוב אותו.

<u>הסבר</u>: תוכנית המבקשת לקרוא רשומה מסוימת, תחזיק אותה בסטטוס "נעילה משותפת" ובכך **תאפשר לתוכניות אחרות לקרוא במקביל** את אותה הרשומה (אך לא תאפשר לעדכן אותה). ברגע שהתוכנית מבקשת לעדכן את הרשומה היא צריכה להעביר את הנעילה מ- "שיתופית" ל- "בלעדית".

נעילה בלעדית (eXclusive lock) - אם טרנזקציה Ti קיבלה נעילה בלעדית (המסומנת ב **X**) על פריט מידע Q, אזי רק Ti רשאית לקרוא ולכתוב פריט זה.

<u>הסבר</u>: אף **תוכנית יישום אחרת אינה מורשית** לקרוא (Select) או לעדכן (Update) את הנתונים בזמן ש Ti רצה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

מטריצת תאימות הנעילות

תוכנית יישום א'

תוכנית יישום ב'	לא נעול	נעילת קריאה	נעילת כתיבה
לא נעול		בצע נעילת קריאה	בצע נעילת כתיבה
נעילת קריאה	בצע נעילת קריאה	בצע נעילת קריאה	המתן
נעילת כתיבה	בצע נעילת כתיבה	המתן	המתן

טבלת החלטות לקבלת נעילות.

כל טרנזקציה תפעיל נעילה בגישה לפריט מידע.

סוג הנעילה יקבע לפי הפעולות אותן תרצה הטרנזקציה לבצע על פריט המידע.

פרוטוקולים מבוססי נעילה

נדרוש שכל טרנזקציה תפעיל נעילה בגישה לפריט מידע Q. כאשר סוג הנעילה יקבע לפי הפעולות אותן תרצה הטרנזקציה לבצע על Q.

מטריצת תאימות הנעילות

ז"א שאם טרנזקציה T_1 הפעילה נעילה משותפת (S) על רכיב מידע Q אזי טרנזקציה T_2 גם תוכל להפעיל נעילה משותפת אך לא נעילה בלעדית (X).

	S	X
S	true	false
X	false	false

כאשר טרנזקציה רוצה להפעיל נעילה בלעדית על פריט מידע הנעול בנעילה בלעדית ע"י טרנזקציה אחרת, **יהיה עליה להמתין עד לשחרור הפריט**.

* על טרנזקציה לבצע נעילת פריט מידע טרם תקבל הרשאה לגשת אליו.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה

<u>כללי השימוש בנעילות:</u>

- לפני הגישה לפריט מידע כלשהו (Q) טרנזקציה חייבת לנעול אותו, ע"י בקשת **נעילה משותפת** (לגישה של קריאה בלבד) **ונעילה בלעדית** (לגישת קריאה וכתיבה). טרנזקציה תמשיך להחזיק את הנעילה כל עוד ניגשת ל-Q.
- אם פריט המידע אינו נעול על ידי טרנזקציה אחרת, הנעילה תאושר.
- אם פריט המידע נעול, ה-DBMS קובע מתי הבקשה תואמת את הנעילה הקיימת. אם הבקשה היא לנעילה משותפת של פריט שקיימת עליו כבר נעילה משותפת, הנעילה תאושר.
- טרנזקציה ממשיכה להחזיק בנעילה כל עוד היא צריכה גישה לפריט המידע וזאת עד אשר היא משחררת (unlock) את הנעילה באופן מפורש במהלך הביצוע או כאשר היא מסיימת פעולתה. תוצאות פעולת כתיבה הופכות לזמינות לטרנזקציות אחרות רק כאשר משתחררת נעילת הכתיבה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה - דוגמא

<u>בעיה</u>: נניח שתוכנית A מפיקה דו"ח של מספר הסטודנטים שנרשמו בפועל לפי קורס, כאשר תוכנית זו מתחילה לעבוד ולשלוף נתונים מבסיס הנתונים.

תוך כדי העבודה מתחילה לפעול **(במקביל)** תוכנית B לרישום סטודנטים לקורס ומעדכנת את נתוני אחד הקורסים שכבר נקראו ע"י תוכנית A.

כאשר תוכנית A מסיימת לעבוד היא מציגה נתונים שכבר אינם נכונים מאחר ותוך כדי פעולתה בסיס הנתונים התעדכן ע"י תוכנית B.

<u>פתרון</u>: הגדרת פעולה המבצעת סימון של אובייקט בבסיס הנתונים בכדי להודיע על כך שהאובייקט נמצא כרגע בתהליך של עדכון ולכן הוא חסום לגישה ע"י משתמשים/תוכניות אחרות.

כאשר תוכנית מבקשת לנעול אובייקט נעול היא תכנס לתור התוכניות הממתינות לאובייקט זה עד אשר התוכנית הקודמת לה תשחרר את הנעילה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה - דוגמא

דוגמא: נחזור למערכת הבנקאית

נתון: תהנה 2 טרנזקציות T_1 ו T_2 ויהיו קיימים A ו B שני חשבונות בנק.

T_1	T_2
Lock-X(B)	Lock-S(A)
read(B)	read(A)
B := B - 50	Unlock(A)
write(B)	Lock-S(B)
Unlock(B)	read(B)
Lock-X(A)	Unlock(B)
read(A)	Display(A+B)
A := A + 50	
write(A)	
Unlock(A)	

פעולות:

- טרנזקציה T_1 מעבירה $50 מחשבון B לחשבון A
- טרנזקציה T_2 מציגה את סכום הכסף הכולל הקיים בחשבונות A ו B.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי נעילה - דוגמא

T_1
Lock-X(B)
read(B)
B := B - 50
write(B)
Unlock(B)
Lock-X(A)
read(A)
A := A + 50
write(A)
Unlock(A)

T_2
Lock-S(A)
read(A)
Unlock(A)
Lock-S(B)
read(B)
Unlock(B)
Display(A+B)

נניח שהערך ההתחלתי של חשבון A הוא $100 ושל חשבון B הוא $200.

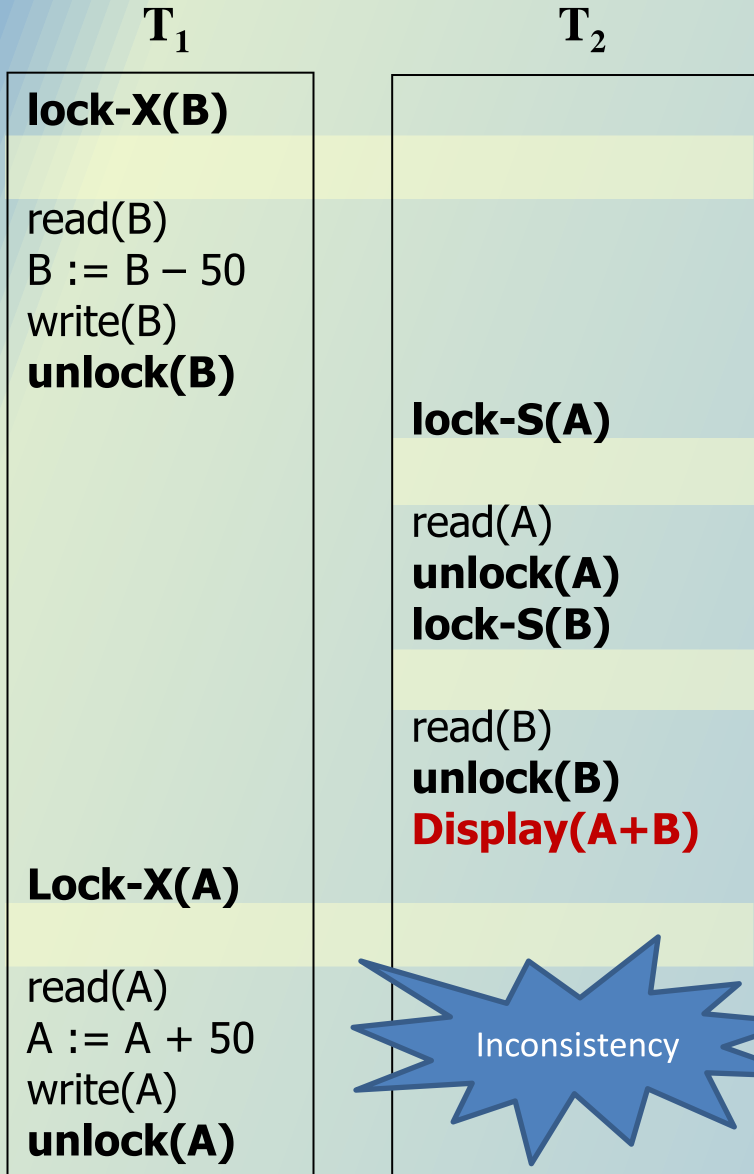
אם שתי הטרנזקציות יורצו בצורה סדרתית ($<T_1,T_2>$ או $<T_2,T_1>$) אזי טרנזקציה T_2 תציג את הערך $300

נראה כעת הרצה בו זמנית המפעילה נעילות למול מערכת ה DBMS.

חשוב להבין שדרישת הנעילה צריכה להיות מאושרת ע"י ה DBMS לפני שהטרנזקציה תוכל לבצע את הפעולה הבאה (אנו נניח שהמערכת מבצעת זאת בזמן אפסי ולכן נפסיק לציין זאת בעתיד).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

דוגמא לשימוש בנעילות



T_2 מציגה את הערך השגוי $250 , פעולה שהסתיימה במצב שאינו עקבי (הסכום בסיום אינו עומד על $300).

טרנזקציה T2 מציגה את הערך השגוי $250 בגלל שטרנזקציה T1 **שיחררה את פריט מידע B מוקדם מדי (שכן הכסף יצא מחשבון B אך טרם נכנס לחשבון A)**, **פעולה שהסתיימה במצב שאינו עקבי** (הסכום בסיום אינו נשמר ואינו עומד על $300).

בעיות בשימוש בנעילות

T_3	T_4
Lock-X(B)	
read(B)	
B:=B-50	
write(B)	
	Lock-S(A)
	read(A)
	Lock-S(B)
Lock-X(A)	
read(A)	
A:=A+50	
write(A)	
unlock(B)	
unlock(A)	
	read(B)
	Display(A+B)
	unlock(A)
	unlock(B)

Deadlock

גם השימוש בנעילות יכול לגרור מצבים בלתי רצויים: נסתכל בתזמון הבא שבו T_3 הפעילה נעילה בלעדית על B ולאחר מכן T_4 מבקשת לבצע נעילה משותפת על B, ז"א ש T_4 ממתינה לכך ש T_3 תשחרר את B.

במקביל למצב זה, T_4 הפעילה נעילה משותפת על A ומאוחר יותר T_3 מבקשת לבצע נעילה בלעדית על A, ז"א ש T_3 ממתינה לכך ש T_4 תשחרר את A.

הגענו למצב שאף אחת מהטרנזקציות לא יכולה להמשיך לרוץ – מצב זה מכונה "מבוי סתום" ובשפה המקצועית - **Dead-Lock**.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

מבוי סתום – Dead Lock

במצב של מבוי סתום (נעילה ללא מוצא) אנו נמצאים בתרחיש שבו תוכנית יישום (אחת או יותר) נועלת אובייקט שהתוכנית השנייה מבקשת להשתמש בו.

<u>מקרה שבו שתי טרנזקציות המעדכנות שתי רשומות שונות אולם בסדר הפוך:</u>

- טרנזקציה A מתחילה לפעול ושולפת שורה X ומבצעת לה נעילת בלעדית לקראת ביצוע פעולת עדכון.
- טרנזקציה B מתחילה לפעול ושולפת שורה Y ומבצעת לה נעילה בלעדית.
- טרנזקציה A מבקשת לקרוא את שורה Y ולבצע לה נעילה בלעדית אך מכיוון ששורה זו כבר נעולה הטרנזקציה נכנסת למצב המתנה.
- טרנזקציה B מבקשת לקרוא את שורה X ולבצע לה נעילה בלעדית אך מכיוון ששורה זו כבר נעולה הטרנזקציה מוכנסת למצב המתנה.

- ← שתי התוכניות ממתינות (מצב ללא מוצא).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

בעיות בשימוש בנעילות – מבוי סתום

T_3	T_4
Lock-X(B)	
read(B)	
B:=B-50	
write(B)	
	Lock-S(A)
	read(A)
	Lock-S(B)
Lock-X(A)	
read(A)	
A:=A+50	
write(A)	
unlock(B)	
unlock(A)	
	read(B)
	Display(A+B)
	unlock(A)
	unlock(B)

Deadlock

במקרה הקודם ניתן לראות תרחיש שבו רצות 2 טרנזקציות במקביל, כאשר שתיהן ביצעו נעילות המונעות את התקדמותן ונוצר מצב של "מבוי סתום".

במצב זה קיימות שתי תוכניות או יותר המחכות כל אחת לשנייה שתשחרר נעילת פריט כלשהו, וכך נוצר מצב ששתיהן ישארו **ב-מצב המתנה אינסופי**, כי אף אחת מהן לא תבצע שחרור של הנעילה (מצב הדומה ללולאה אינסופית).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

בעיות בשימוש בנעילות – מבוי סתום

T_3	T_4
Lock-X(B)	
read(B)	
B:=B-50	
write(B)	
	Lock-S(A)
	read(A)
	Lock-S(B)
Lock-X(A)	
read(A)	
A:=A+50	
write(A)	
unlock(B)	
unlock(A)	
	read(B)
	Display(A+B)
	unlock(A)
	unlock(B)

Deadlock

במצב של Dead-Lock המערכת תצטרך לבצע גלגול לאחור לאחת הטרנזקציות וכך פרטי המידע שהיו נעולים ע"י טרנזקציה זו ישוחררו ויהיו זמינים לטרנזקציה השנייה שתוכל להמשיך בריצה.

למרות שנרצה להימנע ממצבי Dead-Lock שלא היו קיימים טרם השימוש בנעילות, מצב זה עדיף על גבי מצב של חוסר עקביות בנתונים **שכן ב Dead-Lock ניתן לטפל ע"י פעולה של גלגול לאחור** של אחת הטרנזקציות, בעוד מצב של חוסר עקביות יכול להוביל לבעיות שלא ניתנות לטיפול ע"י בסיס הנתונים.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

בעיות בשימוש בנעילות – הרעבה

מצב בו קיימות מספר טרנזקציות שרצות במקביל וחלק מהטרנזקציות מבקשות לבצע **נעילה משותפת** (נעילת קריאה) על פריט מידע לתקופת זמן קצרה מאוד.

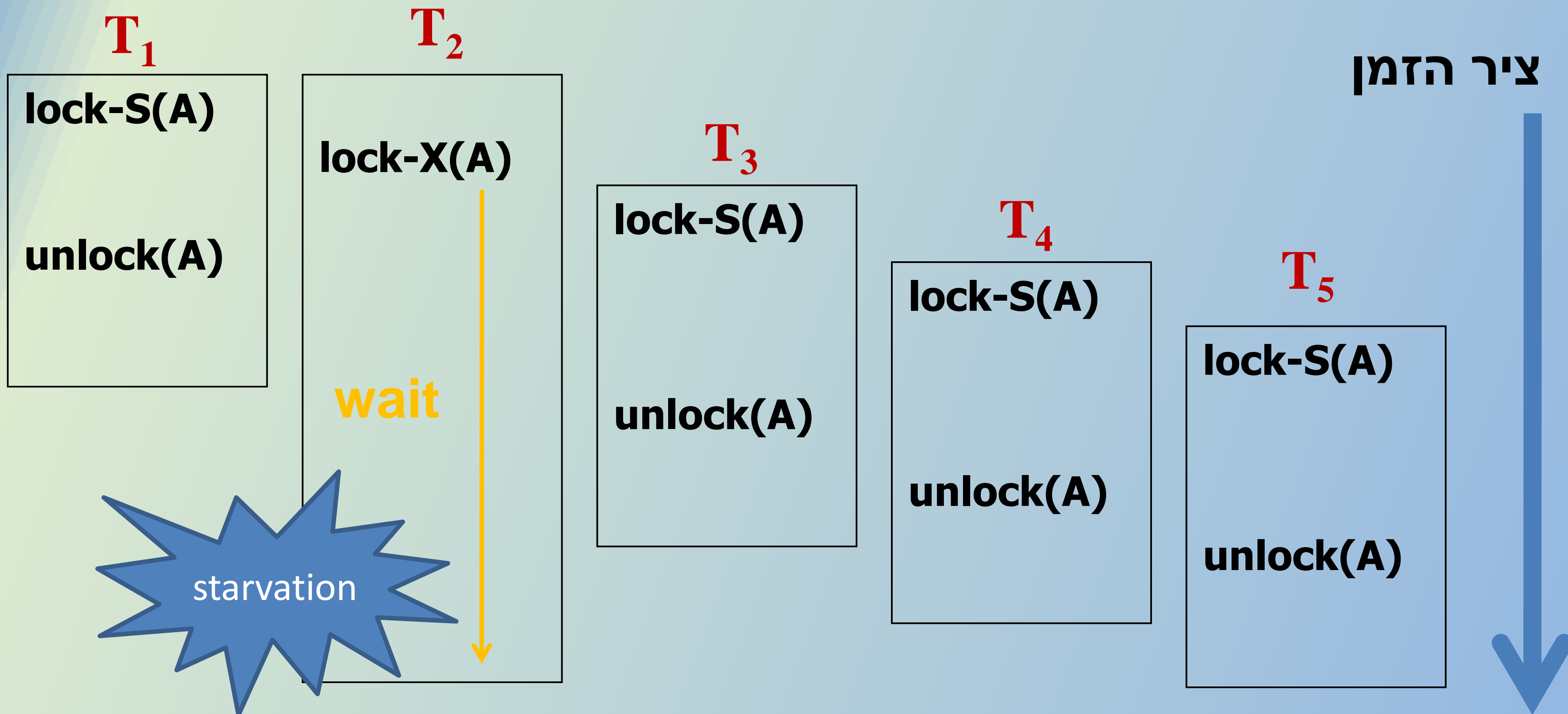
במקביל קיימת טרנזקציה שמעוניינת לקבל אישור **לנעילה בלעדית** על אותו פריט מידע.

הבעיה: הטרנזקציה שמחכה לקבלת אישור לנעילה בלעדית, אף פעם לא תקבל זאת ותמשיך להמתין כי בינתיים קיימות טרנזקציות אחרות שמסתפקות בביצוע נעילה משותפת – מצב מסוג זה נקרא מצב של "הרעבה".

נראה המחשה...

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

מניעת "הרעבות" (starvation)



ולכן נשאף שאישור נעילה יינתן אם שני התנאים הבאים מתקיימים:

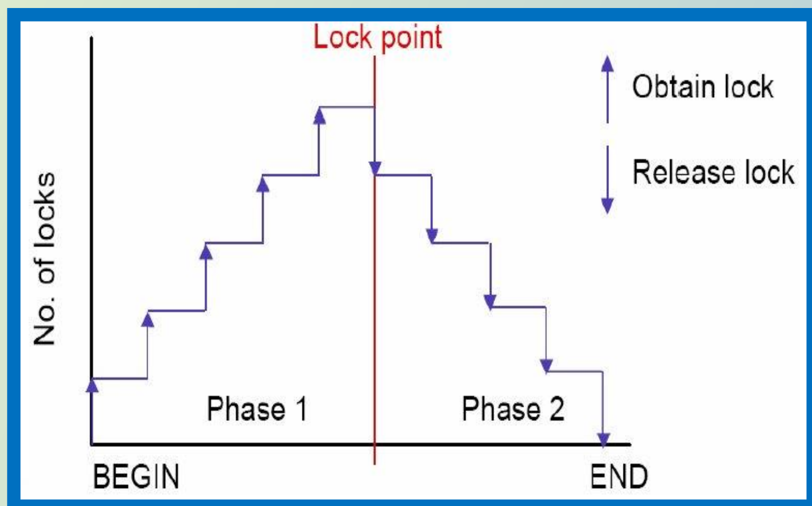
- פריט המידע אינו נעול, או שניתן לבצע נעילה משותפת
- אין טרנזקציה אחרת הממתינה לנעילת הפריט (הגישה בקשה קודמת).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקול "נעילה דו-שלבית"

בכדי לטפל נכונה במקרים של Dead-Locks נוודא שכל טרנזקציה במערכת המבצעת נעילות ושחרורים, תעקוב אחר פרוטוקול נעילה מוגדר.

פרוטוקול נעילה: אוסף כללים לתיאור סדר הבקשות לביצוע נעילות והבקשות לשחרור נעילות ע"י הטרנזקציות.



אחד הפרוטוקולים שמבטיח תזמון סדרתי הוא פרוטוקול הנעילה הדו-שלבית (Two Phase Locking = 2PL).

<u>לקריאה נוספת:</u>

http://www.cubrid.org/blog/cubrid-life/all-about-two-phase-locking-and-a-little-bit-mvcc/

ב) פרוטוקול תגי זמן

השיטה הנפוצה ביותר לקביעת סדר ההרצה של הטרנזקציות (בצורה סדרתית) היא פרוטוקול תגי הזמן (Timestamp protocol).

כל טרנזקציה Ti מקבלת טרם ריצתה **חותמת זמן / תג זמן - timestamp** המסומנת ע"י **TS(Ti)** שהיא ייחודית וקבועה על ידי המערכת*.

חותמת הזמן תקבע בסדר עולה עבור כל טרנזקציה חדשה המתווספת לתזמון ולפי תזמון זה יקבע סדר ההרצאה בפועל.

תג הזמן של הטרנזקציות קובע את סדר הריצה, כך שאם TS(Ti) < TS(Tj) אזי המערכת חייבת לוודא שהתזמון שנוצר שקול לתזמון סדרתי שבו Ti רץ לפני Tj ז"א לתזמון הסדרתי <Ti,Tj>.

* תג הזמן נקבע לרוב לפי שעון המערכת ברגע התחלת ריצת הטרנזקציה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקול תגי זמן

חותמת הזמן מעידה על המועד היחסי של תחילת הטרנזקציה, בנוסף כל פריט מידע (Q) מקבל שתי חותמות זמן:

- **W-timestamp(Q)** הערך המרבי מבין כל חותמות הזמן של הטרנזקציות שכתבו בהצלחה את פריט המידע Q עד שלב זה (פקודת write(Q)).

- **R-timestamp(Q)** הערך המרבי מבין כל חותמות הזמן של טרנזקציות שקראו בהצלחה את פריט המידע Q עד שלב זה (פקודת read(Q)).

ערכי חותמות זמן אלו מתעדכנים במהלך ריצת הטרנזקציה, בכל פעם שפקודות read(Q) או write(Q) מבוצעות על פי:

R-timestamp(Q) = Max(R-timestamp(Q) , TS(Ti))
W-timestamp(Q) = Max(W-timestamp(Q) , TS(Ti))

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקול תגי זמן

בשיטה זו לכל טרנזקציה יש 2 סוגי נעילות: Read Lock ו- Write lock.

בפרוטוקול תגי הזמן אם טרנזקציה אחת נועלת פריט כלשהו ב Read Lock אזי לטרנזקציה אחרת מותר לגשת לפריט זה ואף לנעלו ב Read Lock משלה אבל אסור לה לכתוב לפריט זה.

אם טרנזקציה נועלת פריט מסוים ב Write Lock אזי לאף טרנזקציה אחרת אסור לגשת לפריט זה.

פרוטוקול תגי הזמן עובד כך שאם טרנזקציה כלשהי מפרה את השקילות הסדרתית אזי מוציאים אותה מהתזמון ומשחזרים* את בסיס הנתונים כך שהנתונים יוחזרו למצב שהיה כאילו טרנזקציה זו מעולם לא התבצעה.

* שחזור זה מתבצע על ידי אלגוריתמי התאוששות (יוזכר בהמשך).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקול תגי זמן

<u>תכונות הפרוטוקול:</u>

- הפרוטוקול מבטיח תזמונים הניתנים לארגון סדרתי conflict serializable (שקילות סדרתית לתזמון עפ"י תגי זמן), הסיבה: פעולות מתנגשות מבוצעות עפ"י סדר תגי הזמן (לפי עקרונות הפרוטוקול).
- הפרוטוקול מבטיח **תזמונים נקיים מ- deadlocks** (כי אף טרנזקציה לא מחכה למשאבים – מתרחשת או מגולגלת לאחור).
- **תיתכן "הרעבה"** (במידה ונדרש אתחול מחודש רב פעמי של אותה טרנזקציה).
- **<u>לקריאה נוספת</u>**: קיים פרוטוקול יעיל יותר של תגי הזמן (שלא נעסוק בו בקורס) שנקרא - **Thomas' write rule**

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

ג) פרוטוקולים מבוססי אימות (Validation)

במקרה שרוב הטרנזקציות בתזמון הינן טרנזקציות לקריאה בלבד, יחס ההתנגשויות בין הטרנזקציות יהיה נמוך יחסית.

ולכן חלק ניכר מהטרנזקציות הללו גם אם הן יורצו ללא בקרת מקביליות ריצתם תסתיימנה במצב עקבי.

בקרת המקביליות כופה תקורת תפעול נוספת ועלולה לגרור עיכובים במהלך הריצה ולכן לפעמים נעדיף להשתמש באלטרנטיבה בעלת תקורה נמוכה יותר.

הקושי בהקטנת התקורה היא שאנו לא יודעים מראש איזה טרנזקציות יהיו חלק מההתנגשויות ועל מנת לקבל את הידע הזה נצטרך **סכמת ניתור** למערכת שתוגדר בהמשך ותאפשר את הקטנת התקורה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי אימות (Validation)

אנו נניח שכל טרנזקציה Ti מפעילה מספר שלבים שונים במהלך חייה, תלוי האם היא טרנזקציה לקריאה בלבד או טרנזקציית עדכון.

<u>**שלושת השלבים**</u>:

- **read phase** – במהלך שלב זה, המערכת מריצה את טרנזקציה Ti וקוראת את הערכים של פרטי המידע ומאחסנת אותם במשתנים מקומיים זמניים (מבלי לעדכן עדיין את מסד הנתונים).
- **validation phase** – טרנזקציה Ti מבצעת בדיקה לקביעה האם היא יכולה להעתיק למסד הנתונים את המשתנים הזמניים מבלי לגרום לפגיעה בסדרתיות.
- **write phase** – אם טרנזקציה Ti מצליחה בבדיקת התקינות של השלב הקודם אזי המערכת מבצעת בפועל את העדכונים למסד הנתונים, אחרת מבוצע גלגול לאחור לטרנזקציה ל Ti.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוטוקולים מבוססי אימות (Validation)

בזמן ביצוע טרנזקציה לא מתבצעת בדיקה של serializability, אלא רק לאחר סיומה, היתרונות מביצוע זה הם:

- חסכון בהפעלת בקרת בו-זמניות, שהוא תהליך יקר
- יעיל כשלא צפויות הרבה התנגשויות, למשל שעיקר הפעילות היא קריאת נתונים וביצוע פעולות חישוביות.

טרם פעולת הכתיבה תתבצע בדיקה, אך עד פעולת הבדיקה כל פעולות הקריאה ופעולות החישוב יכולות להתבצע. פעולת הבדיקה תתרחש לקראת סוף הטרנזקציה (פרוטוקול אופטימי).

לקריאה נוספת: http://coddicted.com/validation-based-protocol/

בסיסי נתונים – קורס מתקדם – רועי זרחיה ©

אלגוריתמי התאוששות

סקרנו את 3 הפרוטוקולים ולכן חשוב לדעת שקיימים מספר אלגוריתמי התאוששות שמטרתן לבצע שחזור לבסיס הנתונים למצב שהיה לפני ביצוע פעולות מסוימות שהתגלו כלא תקינות.

<u>מתי נצטרך להפעיל אלגוריתמי התאוששות ?</u>

- במקרה של Deadlock שבו ייתכן ונרצה להסיר טרנזקציה אחת מהתזמון ולשחזר את בסיס הנתונים למצב שהיה אלמלא הופעלה טרנזקציה זו.
- בהרצת אלגוריתם Timestamps יש מקרים בהם טרנזקציה שמפריעה לשקילות הסדרתית יורדת, ואז צריך לשחזר את בסיס הנתונים למצב שהיה אלמלא הופעלה טרנזקציה זו.
- במקרה שבוצעה פעולה לא חוקית במהלך הטרנזקציה (כמו חלוקה באפס) גם במקרה זה יש לבטל טרנסקציה זו.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תוספות לשפת SQL

תוספות לשפת SQL

מכיוון ששפת SQL היא שפה הצהרתית (הגדרה של מה לעשות), אזי בכדי לענות על הצורך הקיים בשפה פרוצדורלית (איך לעשות), הוציאה חברת אורקל פתרון הנקרא **PL/SQL** המאפשר לכתוב שאילתות התומכות גם במשתנים, טריגרים, פונקציות ופרוצדורות המאוחסנים בתוך בסיס-הנתונים.

על אותו משקל הוציאה חברת Microsoft פתרון הנקרא **T-SQL** המאפשר גם את הרחבת השפה לשאילתות התומכות במשתנים, טריגרים, פונקציות ופרוצדורות.

PL/SQL: Procedural Language/Structured Query Language
T-SQL: Transact-SQL

קיימים הבדלים מבחינת סינטקס בין שתי השפות אך העיקרון של שתיהן דומה, בדוגמאות הבאות נשתמש ב PL/SQL שכן MySQL תומך בה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תוספות לשפת SQL

בשפה זו יהיה לנו **אזור להגדרת משתנים (Declare)** ולאחר מכן אזור מתוחם שבו ניתן לכתוב פקודות המשתמשות במשתנים אלו, אזור זה יתחיל בפקודה BEGIN ויסתיים בפקודה END:

```
DECLARE
        v_sname      text;
        v_rating     int;
BEGIN
    SELECT  sname, rating
    INTO    v_sname, v_rating
    FROM    Sailors
    WHERE   sid = '112';
END;
```

ע"י הרצת פקודת SELECT זו נוכל להריץ שאילתא ולהכניס את תוצאותיה (בהנחה ותחזיר שורה אחת בלבד) לתוך משתנים, שמאוחר יותר נוכל לפנות ולהשתמש בערכים המאוחסנים בהם.

פרוצדורות ופונקציות

1) פרוצדורות

פרוצדורה (המכונה לעתים קרובות גם Stored Procedure) היא תת תוכנית **<u>השמורה</u>** בבסיס הנתונים.

לכל פרוצדורה יש שם, רשימת פרמטרים, והיא מכילה אוסף של פקודות SQL.

<u>יש שני סוגים של פרוצדורות:</u>

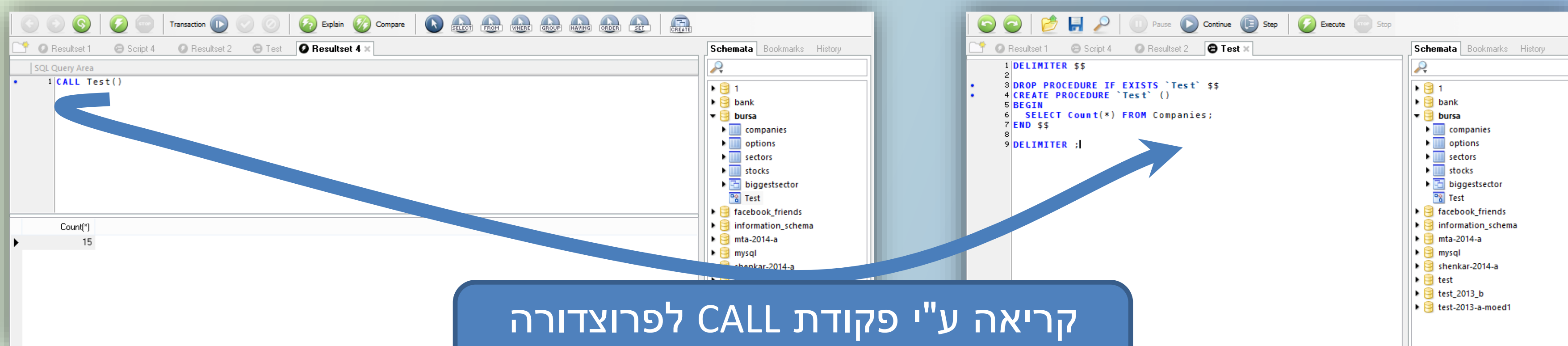
- **פרוצדורות** המאוחסנות ב-DB שניתן להפעילן על ידי קריאה (call) – מטרתן לבצע עדכונים בבסיס הנתונים.
- **פונקציות** המחזירות ערכים שבהם ניתן להשתמש במשפטי SQL אחרים באותו אופן שאנו משתמשים בפונקציות מערכת הקיימות ב- SQL.

ההבדל העיקרי הוא שפונקציות (UDF) יכולות לשמש כמו כל ביטוי אחר בתוך משפטי SQL, ואילו פרוצדורות (SP) חייבת להיות מופעלות ע"י CALL.

פרוצדורות ב SQL

הפרוצדורות הן מהירות כיוון שה DBMS משתמש ביתרון של **אחסון בזיכרון** ומכאן "הרווח המהיר" העיקרי שמגיע מהפחתת תעבורת הרשת.

לאור כך, אם יש לנו משימה החוזרת על עצמה שדורשת בדיקה, לולאה או הרצת **שאילתות מרובות** ללא התערבות מצד המשתמש, מומלץ לעשות זאת ע"י קריאה בודדת של פקודת CALL לפרוצדורה המאוחסנת בשרת.



בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוצדורות ב SQL

<u>תו מפריד</u>: הינו תו (או מחרוזת תווים) המשמשים להגדרת סיום פקודת SQL. כברירת מחדל אנו משתמשים בתו " ; " כ-מפריד, אבל זה גורם לבעיה בהרצת פרוצדורות שבהן יכולות להיות שאילתות רבות שכולן צריכות להסתיים ב (;) ולכן נבחר תו מפריד (נוסף) חדש עבור סימון סיום הפרוצדורה, לדוגמא: **$$.**

<u>הגדרת התו המפריד להיות $$:</u>

```
DELIMITER $$ ;

SELECT * FROM user $$

DELIMITER ; $$
```

הגדרת תו מפריד חדש

שימוש בתו המפריד החדש בשאילתא

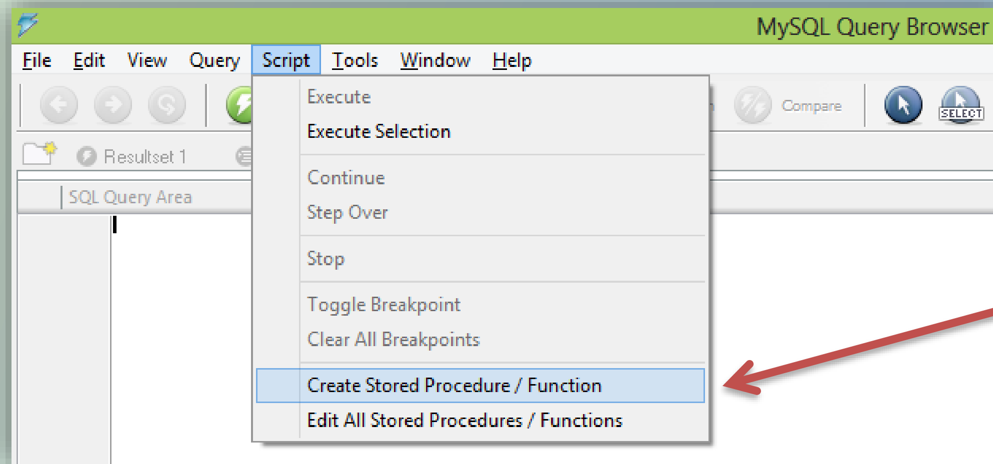
החזרת התו המפריד להיות (;)

**** יש לזכור בסוף התהליך להחזיר חזרה את המפריד להיות (;).**

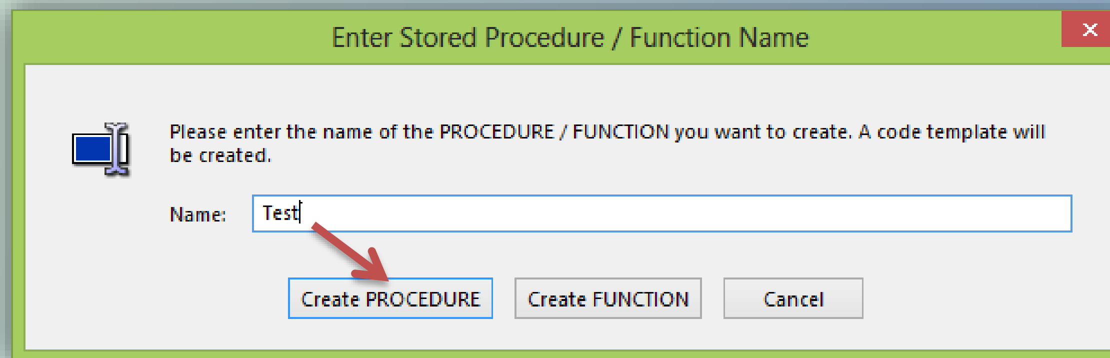
בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

יצירת פרוצדורות ב MYSQL

על מנת ליצור פרוצדורה בפלטפורמת MYSQL ניכנס לתפריט Scripts:



ואז יפתח חלון בו נצטרך להכניס את שם הפרוצדורה/הפונקציה החדשה:



בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

הרצת פרוצדורה

הרצת פרוצדורה (סקריפט)

שם הפרוצדורה

תוכן הפרוצדורה

הופעת הפרוצדורה לאחר הריצה

```
1 DELIMITER $$
2
3 DROP PROCEDURE IF EXISTS `Test` $$
4 CREATE PROCEDURE `Test` ()
5 BEGIN
6   SELECT Count(*) FROM Companies;
7 END $$
8
9 DELIMITER ;
```

לעדכון פרוצדורה נשתמש בפקודת ALTER ולמחיקתה בפקודת DROP.

הפעלת פרוצדורה קיימת

הרצת שאילתא (קריאה לפרוצדורה)

```
1 CALL Test()
```

הפעלת הפרוצדורה ע"י CALL

Count(*)
15

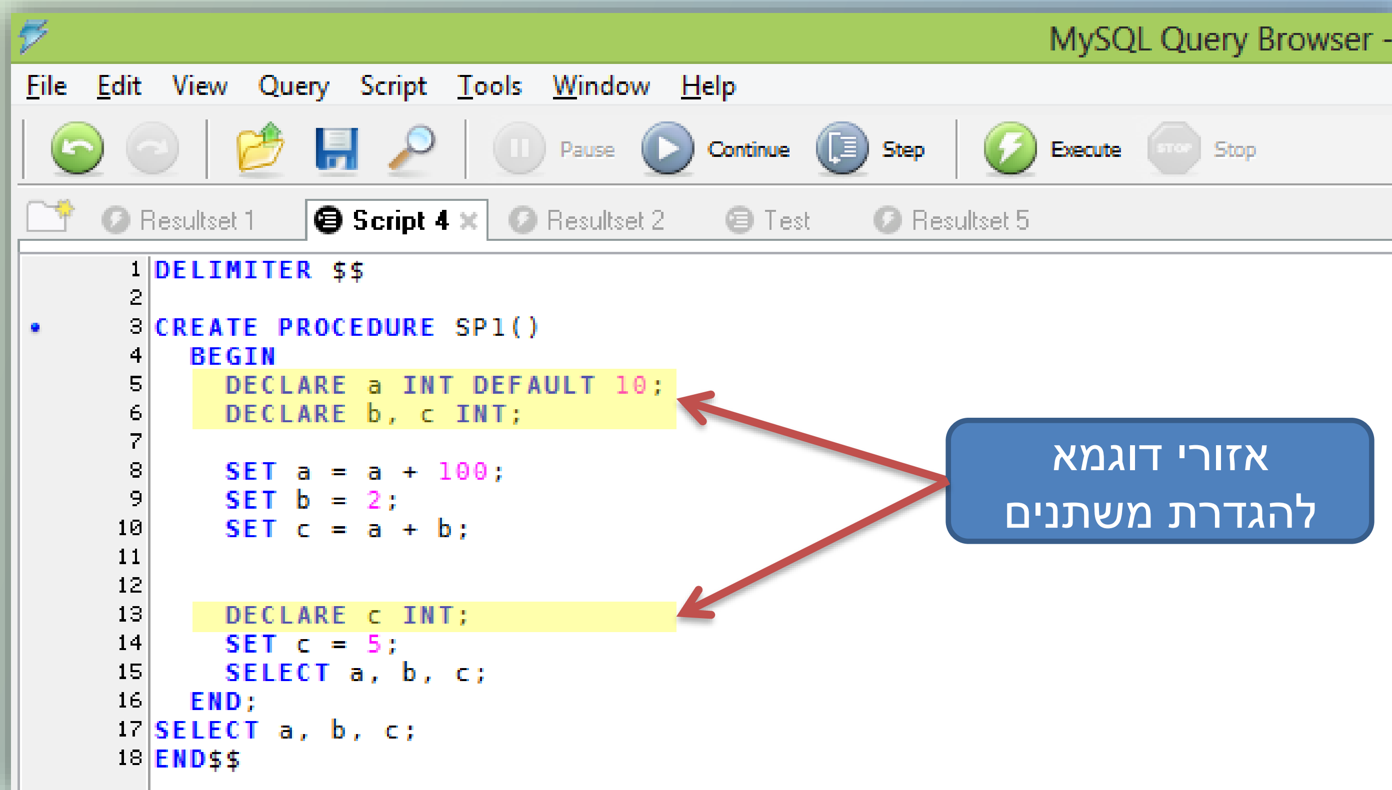
תוצאת הפעלת הפרוצדורה

בהנחה והפרוצדורה מקבלת פרמטרים, נצטרך לרשום אותם בתוך הסוגריים מייד לאחר שם הפרוצדורה בפקודת ה CALL.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

הגדרת משתנים פנימיים

בהנחה ויש צורך להגדיר משתנים במהלך הפרוצדורה שישמרו מידע, נוכל לעשות זאת ע"י הוספת המילה DECLARE בתחילת הפרוצדורה ותחתיה להגדיר את המשתנים שנרצה.



פרוצדורה לדוגמא

בהנחה ויש לנו טבלה של אופציות בבורסה לני"ע כאשר לכל אופציה יש תאריך פקיעה, אנו נרצה להגדיר פרוצדורה שברגע שנפעיל אותה היא תעדכן את כל תאריכי הפקיעה להיות התאריך של היום (תאריך יום הרצת הפרוצדורה):

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `ChangeExpDate` $$
CREATE PROCEDURE `ChangeExpDate` ()

BEGIN
  DECLARE EndDate date;
  SET EndDate = curdate();

  UPDATE options SET Exp_Date = EndDate;

  SELECT * FROM options;
END $$

DELIMITER ;
```

הגדרת משתנה מסוג תאריך

השמת התאריך של היום לתוך המשתנה החדש

עדכון ערך האופציה בטבלה

שאילתה להצגת המידע

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוצדורה המקבלת פרמטרים

פרוצדורה חדשה שנוצרת לא חייבת להכיל פרמטרים (סוגרים ריקיים) אך אם יש צורך היא יכולה להכיל פרמטרים מ-3 סוגים שונים כך שהפרוצדורה מקבלת ערכים מהמשתמש או מהיישום המורץ ומשתמשת בהם במהלך ההרצה.

<u>סוגי הפרמטרים:</u>

- **<u>פרמטרים מסוג IN</u>**: מקרה בו המשתמש מעביר פרמטרים לתוך (IN) הפרוצדורה. הפרוצדורה יכולה לעדכן את הערך אך עדכון זה לא יהיה זמין למשתמש בסיום הרצת הפרוצדורה (**זו ברירת המחדל**).
- **<u>פרמטרים מסוג OUT</u>**: מקרה בו הפרוצדורה מוציאה (OUT) ערך למפעיל כאשר ערכה ההתחלתי הוא NULL והערך המעודכן יהיה זמין למפעיל בסיום הרצת הפרוצדורה.
- **<u>פרמטרים מסוג INOUT</u>**: שילוב של 2 המקרים הקודמים: המפעיל מעביר משתנה עם ערך שערכו המעודכן לאחר הריצה יהיה זמין למפעיל.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוצדורה המקבלת פרמטרים (IN)

ניקח את הדוגמא הקודמת של שינוי תאריך האופציה רק שכעת במקום להשתמש בתאריך הנקבע בפרוצדורה, נעביר אותו כפרמטר לפרוצדורה:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `ChangeExpDate2` $$
CREATE PROCEDURE `ChangeExpDate2` (IN EndDate date)

BEGIN
  UPDATE options SET Exp_Date = EndDate;

  SELECT * FROM options;
END $$

DELIMITER ;
```

פרמטר המוכנס לתוך הפרוצדורה ע"י המפעיל

עדכון ערך האופציה מתוך הפרמטר שהתקבל

קריאה חיצונית לפרוצדורה עם פרמטר של תאריך רצוי (יש לשלוח שם פרמטר או ערך):

```
CALL ChangeExpDate2('2023-09-22')
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פרוצדורה המקבלת פרמטרים (OUT)

פרוצדורה המחשבת מחיר אופציה ממוצע ומחזירה אותו למשתמש:

```
DELIMITER $$
CREATE PROCEDURE `ChangeExpDate3` (OUT AvgPrice float)
BEGIN
 SELECT AVG(price) INTO AvgPrice FROM options;
END $$
DELIMITER ;
```

פרמטר המוגדר בפרוצדורה ויעודכן על ידה עבור המפעיל

שליפת ערך מחיר האופציה לתוך הפרמטר החוזר אלינו

קריאה חיצונית לפרוצדורה עם שם של פרמטר משתמש (יש להגדירו בשלב הקריאה עם קידומת של @ כי זה משתנה גלובלי שחוזר אלינו):

```
CALL ChangeExpDate3(@M)
```

נוכל כמובן להריץ לאחר מכן **select @m;** בכדי לקבל את ערך הפרמטר.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

במהלך כתיבת פרוצדורות נוכל להשתמש בשאילתות רגילות אך גם בפקודות הבאות: IF, CASE, ITERATE, LEAVE, LOOP, WHILE, REPEAT.

את רוב הפקודות טרם למדנו ולמרות זאת חשוב להבין שבמקרים של פקודות שכבר נלמדו ב SQL צריך להבין שהם יוצגו כאן במבנה פקודה שונה כאשר נשתמש בהם תחת פרוצדורה:

פקודת IF:

```
IF <condition>
        THEN <command>
ELSEIF < condition2>
        THEN <command2>
ELSE
        <command3>
END IF;
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

פקודות CASE:

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

משתנה איטרציה:

על מנת שנוכל להריץ לולאות בתוך פרוצדורות נצטרך משתנה איטרציה שישתנה בכל הרצה בודדת של לולאה מסוג LOOP , REPEAT , WHILE.

הגדרת משתנה הלולאה:

ITERATE label

הודעת יציאה מהלולאה:

LEAVE label

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

פקודות LOOP:

```
[begin_label:] LOOP
statement_list
END LOOP
[end_label]
```

```
loop_label: LOOP
INSERT INTO number VALUES (rand());
SET x = x + 1;
IF x >= num
  THEN LEAVE loop_label;
END IF;
END LOOP;
```

לולאה שבכל איטרציה מכניסה ערך
רנדומלי לטבלת NUMBER כל עוד ש-X
קטן מהמשתנה **NUM** שהתקבל
כפרמטר לפרוצדורה ורק במקרה ש-X
גדול מהמספר אזי יוצאים מהלולאה

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

פקודות REPEAT:

```
[begin_label:] REPEAT
statement_list
UNTIL search_condition
END REPEAT
[end_label]
```

```
REPEAT
IF mod(@x, 2) = 0 THEN SET @sum = @sum + @x;
END IF;
SET @x = @x + 1;
UNTIL @x > n
END REPEAT;
```

בלולאה זו, המשתמש מעביר פרמטר מספרי לפרוצדורה וזו מבצעת סכימה של כל המספרים הזוגיים מהמספר 1 (בלולאה) עד המספר שהוכנס

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פקודות נוספות לשימוש בתוך פרוצדורות

פקודות WHILE:

```
[begin_label:] WHILE
DO
statement_list
END WHILE
[end_label]
```

```
WHILE @x<n DO
IF mod(@x, 2) <> 0 THEN SET @sum = @sum + @x;
END IF; SET @x = @x + 1;
END WHILE;
```

בלולאה זו המשתמש מעביר פרמטר מספרי
לפרוצדורה וזו מבצעת סכימה של כל המספרים האי-
זוגיים מהמספר 1 (בלולאה) עד המספר שהוכנס

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

2) פונקציות

כשנרצה להריץ אוסף פקודות או מספר שאילתות אחת אחרי השנייה, נוכל להשתמש בפרוצדורה ובכך אנו בעצם יוצרים רוטינה במסד הנתונים, שמרגע יצירתה ועד מחיקתה נוכל להפעיל אותה בצורה פשוטה ע"י קריאת CALL בכל זמן שרק נרצה (עם או בלי פרמטרים).

<u>**פונקציה**</u> הינה כלי כמעט זהה לפרוצדורה ויכולה לקבל פרמטרים חיצוניים מסוג IN (זו ברירת המחדל ואין צורך לציינה) אך עם הבדל אחד מהותי, **הפונקציה מחזירה ערך** וזאת ע"י שימוש בפקודת **RETURNS** המגדירה את סוג ה TYPE שמחזירה הפונקציה כאשר בתוך הפונקציה חייבת להיות לפחות פקודת **RETURN** אחת.

```
CREATE FUNCTION NAME (parameters)
RETURNS datatype
BEGIN
        Statements;
END;
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

פונקציה לדוגמא

```
DELIMITER $$

CREATE FUNCTION rating_price(stockPrice int)
RETURNS text
BEGIN
  IF stockPrice > 300 THEN return 'The price is expensive';
  ELSEIF stockPrice >= 150 THEN return 'The price is OK';
  ELSE return 'The price is not expensive';
  END IF;
END $$

DELIMITER ;
```

סוג ה TYPE של תוצאת הפונקציה

פונקציה הבודקת האם מחיר מניה יקר או זול

וכעת אם נריץ את הפקודה **SELECT rating_price(350);** נקבל שמחיר המניה גבוה ואילו אם נריץ **SELECT rating_price(250);** נקבל שמחירה OK.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טריגרים

טריגר - Trigger

טריגר הוא פעולה (פרוצדורה או פונקציה) שמסד הנתונים צריך לבצע בעקבות פעולה אחרת שבוצעה וגררה שינוי במסד הנתונים עצמו.

לדוגמא:

נגדיר טריגר שיופעל בעת ביצוע פקודת INSERT למסד הנתונים, כאשר מטרת הטריגר הינה לבדוק האם הערכים המוכנסים תקינים.

כאשר לקוח מעוניין למשוך כסף מהכספומט, נבדוק קודם לכן שהוא נמצא ביתרת זכות ושפעולת המשיכה לא תגרור חריגה ממסגרת האשראי, ז"א נכין טריגר שיבדוק בכל פעם שתתבצע פעולת משיכה (INSERT) את היתרה הנוכחית טרם מתן האישור למשיכת הכספים.

גוף הטריגר בעצם מכיל אוסף פקודות (PL/SQL או T-SQL) המופעלות בעקבות פעולות: עדכון, מחיקה והוספה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טריגר - Trigger

הטריגר מופעל בצורה אוטמטית (אין אפשרות לקרוא/להפעיל טריגר).

הפעלתו תתרחש בעקבות פעולת INSERT / UPDATE / DELETE על גבי טבלה במסד הנתונים (לא ניתן להפעיל על VIEW או על טבלה מדומה).

הפקודה שהפעילה את הטריגר, כולל הטריגר עצמו מהווים טרנזקציה אחת, ז"א שביצוע גילגול לאחור במסגרת הטריגר יגרור את ביטול כל הפעולות שבוצעו במסגרת הטריגר, כולל הפעלת הטריגר עצמו.

מניעה מראש לעומת תגובה בדיעבד:
למדנו בעבר שבעת יצירת טבלה חדשה נוכל להוסיף אילוצים שמטרתם היא למנוע הכנסת נתונים שגויים ואילו טריגר מטרתו להפעיל תגובה על ביצוע עדכון שאינו תקין ולכן תמיד **עדיף להגדיר אילוץ מאשר לבנות טריגר**.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

יצירת טריגר חדש

<u>מבנה הפקודה:</u>

```
CREATE TRIGGER TriggerName
BEFORE | AFTER
INSERT | UPDATE | DELETE
ON table_name
FOR EACH ROW
BEGIN
    גוף הטריגר
END $$
```

מועד הפעלת הטריגר

סיבת הפעלת הטריגר

*** עדכון טריגר יבוצע ע"י ALTER ומחיקת טריגר ע"י DROP.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

יצירת טריגר חדש

בתוך גוף הטריגר ניתן לפנות לעמודות בטבלה עליה מופעל הטריגר בשני כינויים NEW ו OLD.

השימוש ב **OLD.columnName** מתיחס לעמודה של שורה קיימת לפני שבוצע בה עדכון (פקודת Update) או לפני מחיקתה (פקודת Delete).

השימוש ב **NEW.columnName** מתיחס לעמודה של שורה חדשה שעתידה להיכנס (פקודת INSERT) או לשורה קיימת לאחר שהערך שלה עודכן (פקודת UPDATE).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טריגרים – דוגמא 1

הרצה רגילה ללא צורך בסקריפט

<u>שלב 1: ניצור טבלה חדשה בבסיס הנתונים:</u>

```
CREATE TABLE account (num INT, amount float);
```

<u>שלב 2: ניצור טריגר על טבלה זו:</u>

```
CREATE TRIGGER SumValues
BEFORE INSERT
ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

הטריגר מופעל לפני שערך מוכנס לבסיס הנתונים בפקודת INSERT

<u>גוף הטריגר</u>: עבור כל שורה מוכנסת, תתבצע סכימה של הערכים **החדשים** (אלו שעתידים להיכנס) של עמודת amount

הרצה רגילה ללא צורך בסקריפט

טריגרים – דוגמא 1

שלב 3: להפעלת הטריגר, נאפס את משתנה הסכום (@sum) ונפעיל פקודת INSERT עם מספר ערכים ולבסוף נבדוק את תוכן משתנה ה SUM:

```
SET @sum = 0;

INSERT INTO account
VALUES (1, 500.00), (2, 300.00), (3, -200.00);

SELECT @sum AS 'Total amount inserted';
```

3 פקודות שצריכות להיות מורצות בנפרד

תוצאה:

Total amount inserted
600

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טריגרים – דוגמא 2

מבנה הפקודה:

```
DELIMITER //
CREATE TRIGGER upd_check
BEFORE UPDATE
ON account
FOR EACH ROW
BEGIN
	IF NEW.amount < 0
		THEN SET NEW.amount = 0;
	ELSEIF NEW.amount > 100
		THEN SET NEW.amount = 100;
	END IF;
END;//
DELIMITER;
```

טריגר זה בא לוודא שערך שמתוכנן להתעדכן בטבלה לא יחרוג מהטווח של 0..100

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

מה חדש בתחום ? נושאים נוספים...

NO SQL

- NOSQL שפירושה היא שפה שבה הפקודות הן **<u>לא רק</u>** בשפת SQL.
- המטרה של NOSQL הינה להשתחרר מסכמות (חוסך לבצע המרות מורכבות) ומשימוש יקר בפקודת הצירוף (JOIN).
- היתרון הוא **ביצועי כתיבה מהירים מאוד** ולכן מתקשר במיוחד למונחים מודרנים כמו BIG DATA ועבודה עם כמויות מידע אדירות באינטרנט.
- מסדי נתונים מסורתיים לא יכולים לשמור על קשר עם הביצועים הנדרשים על ידי יישומים לניתוח בקנה מידה גדולים מאוד (OLAP) וחברות אינטרנט צריכות להתמודד עם כמויות עצומות של נתונים והמידע צריך להיות זמין כל הזמן עם זמן תגובה מהיר.
- ל-NOSQL יש יכולת לשמור מידע ולאחזר אותו במהירות רבה ע"י **חלוקת המידע לכמה מקורות אחסון**, ובכך ליצור מסד נתונים מהיר עם מעט צווארי בקבוק (שימוש במספר רב של שרתים שאינם תלויים אחד בשני - **ביזור**).

NO SQL

יתרונות:

- **גמישות** – במקום להשתמש בשרתים גדולים יותר שיוכלו לעמוד בעומס נתונים רב בעלי עלות גבוהה, ה-NOSQL משתמש במספר גדול של שרתים בעלות נמוכה.
- **חסכון בעלויות** – קונספט ה NOSQL מתוכנן בצורה הדורשת פחות ניהול: תיקון אוטומטי, הפצת נתונים ומודלי נתונים. בצורה זו ניתן לצמצם את השימוש ב-DBA, ולהפחית עלויות.
- שימושי מאוד להצגת נתונים **בזמן אמת**

NO SQL

<u>חסרונות:</u>

- לא ניתן לקשר בין טבלאות ולכן אם יש הרבה קשרים בין מידע למידע אחר במערכת – פתרון זה אינו יתאים.
- שימוש בטרנזקציות יפעל טוב יותר ב SQL מאשר ב NOSQL.
- **בשלות** – NOSQL יחסית "חדשה בשוק", בזמן שה RDBMS **יציבה**, פונקציונאלית ואמינה ומוטמעת בקרב מנהלי מערכות מידע רבים וכבר הוכיחה את עצמה לאורך המון שנים.
- **תמיכה** – עבור מערכות ישנות יש תמיכה שוטפת, מוסמכת ואמינה בזמן, ואילו עבור NOSQL החדשה והפחות מוכרת יש פחות תמיכה.
- **בינה עסקית** – תיכנות ב-NOSQL דורש ידע רב ומומחיות מעבר לאלה הדרושים במערכות אחרות, מה שיכול להקשות על העבודה איתה.

NO SQL

האם RDBMS גוססים:

- ממש לא !!! RDBMS ו SQL הם ברירות המחדל בטכנולוגיות מסדי נתונים ברחבי העולם ועדיין נמצאים בשימוש רחב מאוד.
- החברות השונות דואגות להמשיך ולבצע שיפורים רבים וקפיצות מדרגה בביצועים בשל התקדמות בטכנולוגיות אחסון ואופטימיזציות אחרות, מה שהופך אותם מתאימים ליישומי OLAP תובעניים.
- הרבה אתרי אינטרנט מודרניים מסתמכים על מסדי נתונים מרובים, כל אחד מסוג אחר, להיבטים השונים שלהם.

המשך תרגול...

תרגיל כיתה 2
שאלות 8-11

<u>רשימת שאלות סיכום (שעתיים)</u>

שאלה8 – הכנסה עם תת שאילתא
שאלה9 – תתי שאילתות משורשרות
שאלה10 – שאלת בונוס עם תצפית
שאלה11 – כתיבת פונקציה חדשה

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©