ניהול תנועות
חלק 2

Transaction Management
Part 2

1

חזרה על מונחים והגדרות

2

תנועה (או עסקה) Transaction

- תנועה (טרנסקציה או עסקה) היא סדרה של פעולות קריאה וכתיבה של *פריטים* (items) מהמסד
- תנועה היא הפשטה של תוכנית
- הפריטים יכולים להיות רשומות, בלוקים (דפים), טבלאות (יחסים) וכד'

```
Read(A)
Write(A)
Read(B)
Write(B)
```

3

תזמון Schedule

- תזמון הוא ביצוע של מספר תנועות במקביל
- בכל נקודת זמן מתבצעת פעולה אחת בלבד
- פעולת קריאה/כתיבה מתבצעת בצורה אטומית
- התזמון מייצג את הסדר על ציר הזמן (הציר האנכי) בו מתבצעות הפעולות

```
Read(A)
                Read(C)
Write(A)
Read(B)
                Write(C)
                Read(B)
                Write(B)
Write(B)
```

4

תזמון סדרתי
Serial Schedule

- תזמון סדרתי הוא תזמון שבו התנועות מתבצעות זו אחר זו באופן סדרתי
- כל תזמון סדרתי הוא נכון

```
Read(A)
Write(A)
Read(B)
Write(B)
                Read(C)
                Write(C)
                Read(B)
                Write(B)
```

5

מהי שקילות של תזמונים?

- **שני תזמונים הנם *שקולים* אם**
  - שני התזמונים מורכבים מאותן תנועות (ולכל תנועה אותן פעולות בשני התזמונים)
  - לשני התזמונים אותה השפעה על המסד, כלומר למסד יש אותם ערכים עם סיומו של כ"א מהתזמונים
  - בשני התזמונים כל עסקה מייצרת אותו פלט למשתמש, כלומר מציגה בפני המשתמש אותם ערכים לכל פריט שהיא קוראת
- **השקילות חייבת להתקיים לכל חישוב אפשרי**

6

ההגדרה הפורמלית של שקילות במקרה הכללי

7

ערך שנכתב תלוי בערכים שנקראים

- הערך שתנועה כותבת הוא תמיד פונקציה של הערכים שהיא קוראת לפני הכתיבה
- לדוגמה, הערך שנכתב עבור A הוא פונקציה של הערך שנקרא עבור A
- הערך שנכתב עבור B הוא פונקציה של הערכים שנקראים עבור A ועבור C

Read(A)
Write(A)
Read(C)
Write(B)

8

כתיבה עיוורת (Blind Write)

- תנועה מבצעת כתיבה עיוורת אם היא כותבת ערך בלי לקרוא אותו
- העסקה בצד ימין מבצעת כתיבה עיוורת של הפריט B

Read(A)
Write(A)
Read(C)
Write(B)

9

נניח ששני תזמונים (המורכבים מאותן תנועות) כותבים אותם ערכים לכל חישוב אפשרי

- לפיכך, כל עסקה T חייבת לקיים:
  - אם T כותבת פריט כלשהו A, אז בשני התזמונים T קוראת בדיוק אותם ערכים לפני הכתיבה של A

<u>הנמקה:</u> אם בשני התזמונים, T קוראת ערכים שונים לפני הכתיבה של A, אז תמיד אפשר למצוא חישוב ספציפי שלפיו T תכתוב ערכים שונים עבור A בשני התזמונים

10

איך נבטיח שכל תנועה קוראת אותם ערכים בשני התזמונים?

- הדרך היחידה להבטיח ש- T_i קוראת בשני התזמונים את אותו הערך עבור A היא:
  - בשני התזמונים T_i קוראת עבור A את הערך שנכתב ע"י אותה עסקה T_j

<u>הנמקה:</u> אם בשני התזמונים, T_i קוראת עבור A ערכים שנכתבים ע"י עסקאות שונות T_m ו- T_k, אז תמיד אפשר למצוא חישובים של T_m ו- T_k שלפיהם T_m ו- T_k כותבות ערכים שונים עבור A

11

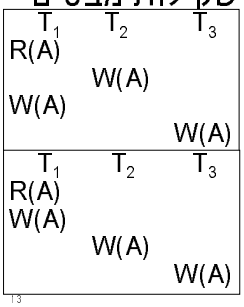
ההגדרה של שקילות מבטים View Equivalence

- תזמונים S_1 ו- S_2 הנם *שקולי מבטים* אם:
  - הם מורכבים מאותן תנועות
  - אם T_i קוראת ערך התחלתי של A ב- S_1, אז T_i קוראת גם ב- S_2 ערך התחלתי של A
  - אם T_i קוראת ערך של A שנכתב ע"י T_j ב- S_1, אז T_i קוראת גם ב- S_2 ערך של A שנכתב ע"י T_j
  - אם T_i כותבת ערך סופי של A ב- S_1, אז T_i כותבת גם ב- S_2 ערך סופי של A

12

דוגמה לשני תזמונים המקיימים את ההגדרה של שקילות מבטים

- בשניהם T_1 קוראת ערך התחלתי של A מהמסד ו- T_3 כותבת ערך סופי של A למסד (זאת כתיבה עיוורת)
- אף עסקה (חוץ מ- T_1) לא קוראת ערך של A או כל ערך אחר

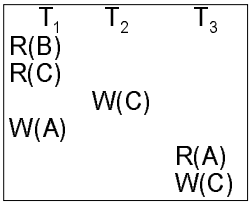


מסקנות הנובעות מההגדרה של שקילות מבטים

- ההגדרה של שקילות מבטיחה שכל תנועה T_i קוראת אותם ערכים בשני התזמונים ולכן גם כותבת אותם ערכים (לכל חישוב אפשרי)
- לערכים שתנועה T_i כותבת יש השפעה רק אם אלה ערכים סופיים שנכתבים למסד או שתנועה אחרת קוראת אותם

דוגמה של תנועה חסרת השפעה

- בדוגמה בצד שמאל כל הכתיבות הן עיוורות
- הערך של C שנכתב ע"י T_2 אינו נקרא ע"י אף תנועה אחרת ואינו הערך הסופי של C שנכתב למסד
- בנוסף, T_2 אינה קוראת אף ערך; לפיכך:
- ל- T_2 אין שום השפעה – אפשר פשוט למחוק אותה מהתזמון



הסדר בין תנועות בתזמון סדרתי שקול (לפי שקילות מבטים)

מה צריך להיות הסדר בתזמון סדרתי שקול?

- נניח שבתזמון S, התנועה T_2 קוראת ערך עבור A שנכתב ע"י T_1
  - ברור שבתזמון סדרתי שקול T_2 תופיע אחרי T_1
- נניח שבתזמון S, התנועה T_3 כותבת את A אחרי ש- T_2 קראה אותו
- היכן יכולה T_3 להופיע בתזמון סדרתי שקול?
  - אחרי T_2, או
  - לפני T_1, בהנחה ש- T_1 אינה מושפעת מהערך של A ש- T_3 כותבת (כלומר, יש כתיבות עיוורות)

מה צריך להיות הסדר בתזמון סדרתי שקול?

- נניח שבתזמון S, התנועה T_2 קוראת ערך עבור A שנכתב ע"י T_1
  - ברור שבתזמון סדרתי שקול T_2 תופיע אחרי T_1
- נניח שבתזמון S, התנועה T_3 כותבת את A אחרי ש- T_2 קראה אותו
- היכן יכולה T_3 להופיע בתזמון סדרתי שקול?

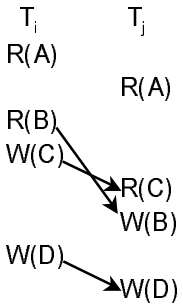
אם אין כתיבות עיוורות, אז T_3 חייבת להופיע אחרי T_2

שקילות קונפליקטים

19

קונפליקטים בין תנועות

- בין שתי פעולות של תנועות שונות על אותו פריט קיים קונפליקט, אלא אם מדובר בשתי פעולות קריאה
- בדוגמה מימין יש 3 קונפליקטים



20

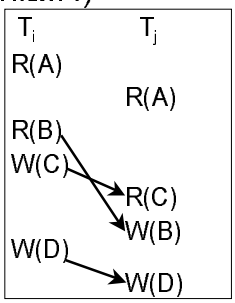
הגדרה של שקילות קונפליקטים Conflict Equivalence

- שני תזמונים הנם *שקולי קונפליקטים* אם
  - הם מורכבים מאותן תנועות
  - כל צמד של פעולות שיש ביניהן קונפליקט מופיע בשני התזמונים באותו סדר
- *הערה:* הקונפליקטים הם
  - קריאה ולאחריה כתיבה על אותו פריט
  - כתיבה ולאחריה קריאה על אותו פריט
  - כתיבה ולאחריה כתיבה נוספת על אותו פריט

21

קונפליקטים בין תנועות (דוגמה קודמת)

- בדוגמה מימין יש 3 קונפליקטים, ולפי כל אחד מהם T_j חייבת להופיע לפני T_j בכל תזמון שקול
- לכן, התזמון הסדרתי שבו T_i מתבצעת ראשונה ו- T_j שניה הנו שקול לתזמון הנתון (אם אין תנועות נוספות)



22

שקילות קונפליקטים גוררת שקילות מבטים

- *למה:* אם S_1 ו- S_2 הנם שקולי קונפליקטים אז הם גם שקולי מבטים
- *הוכחה:* בגלל שהסדר בין כל הקונפליקטים נשמר בשני התזמונים, קל להראות ששני התזמונים מקיימים את כל התנאים של ההגדרה של שקילות מבטים

23

סיכום לגבי שתי ההגדרות של שקילות

- שקילות מבטים היא המושג הכללי ביותר של שקילות בין תזמונים, כאשר לא ידוע דבר לגבי החישובים שעושות התנועות
- שקילות קונפליקטים היא המושג הכללי ביותר של שקילות בין תזמונים אם אין כתיבות עיוורות (ולא ידוע דבר על החישובים)
- שקילות קונפליקטים גוררת שקילות מבטים, אבל לא להפך אם יש כתיבות עיוורות

24

תזמונים ברי-סדרתיות

25

תזמון בר-סדרתיות
Serializable Schedule

- תזמון הנו *בר-סדרתיות* אם הוא שקול לתזמון סדרתי כלשהו
- תזמון בר-סדרתיות הוא תזמון נכון
- צריך דרך לקבוע שתזמון הוא בר-סדרתיות מבלי לדעת את החישובים שהוא מבצע, כלומר התזמון צריך להיות בר-סדרתיות לכל חישוב אפשרי

26

תזמון בר-סדרתיות קונפליקטית
Conflict Serializable Schedule

- תזמון S_1 הנו *בר-סדרתיות קונפליקטית* אם קיים תזמון סדרתי S_2 כך ששני התזמונים הנם שקולי קונפליקטים

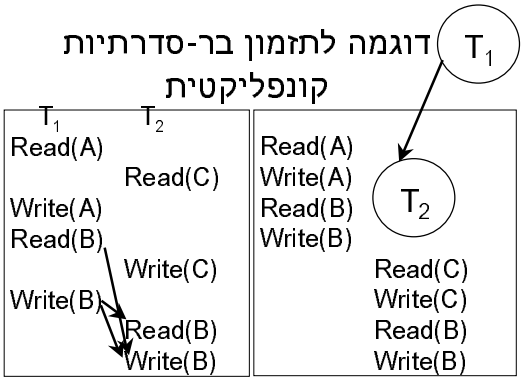
27

גרף הקדימויות
Precedence Graph

- גרף הקדימויות עבור תזמון נתון מכיל
  - צומת לכל תנועה
  - צלע מכוונת מ- T_i ל- T_j אם יש קונפליקט בין T_i לבין T_j שקובע ש- T_i צריכה להופיע לפני T_j
- *משפט:* תזמון הנו בר-סדרתיות קונפליקטית אם ורק אם גרף הקדימויות שלו הנו חסר מעגלים

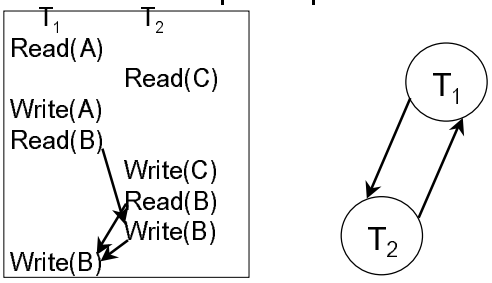
28

דוגמה לתזמון בר-סדרתיות קונפליקטית



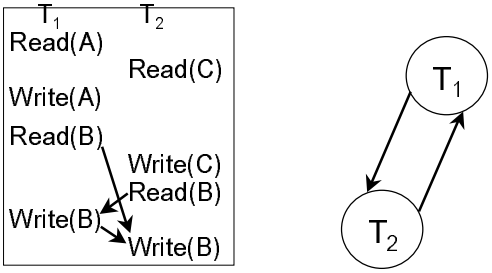
29

דוגמה לתזמון שאינו בר-סדרתיות קונפליקטית



30

עוד דוגמה לתזמון שאינו בר-סדרתיות קונפליקטית



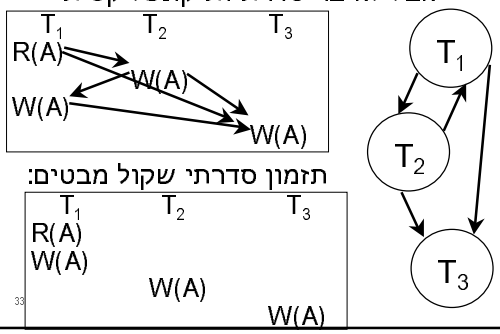
31

בר-סדרתיות מבטית View Serializability

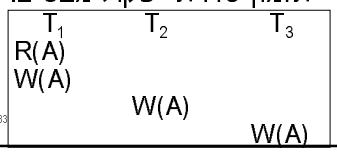
- תזמון S_1 הנו *בר-סדרתיות מבטית* (view serializable) אם קיים תזמון סדרתי S_2 כך ששני התזמונים הנם שקולי מבטים
- ברור שתזמון בר-סדרתיות קונפליקטית הוא בר-סדרתיות מבטית (אבל ההפך אינו בהכרח נכון כאשר יש כתיבה עיוורת – דוגמה בהמשך)
- הבעיה האם תזמון S הנו בר-סדרתיות מבטית הנה קשה (NP שלמה) אם יש כתיבות עיוורות

32

דוגמה לתזמון שהוא בר-סדרתיות מבטית אבל לא בר-סדרתיות קונפליקטית

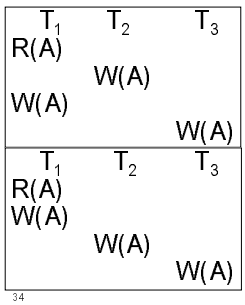


תזמון סדרתי שקול מבטים:



33

שני התזמונים שקולי מבטים, כי



- בשניהם T_1 קוראת ערך התחלתי של A מהמסד ו- T_3 כותבת ערך סופי של A למסד (זאת כתיבה עיוורת)
- אף עסקה (חוץ מ- T_1) לא קוראת ערך של A או כל ערך אחר

34

בקרת מקבילות ע"י נעילות

35

נעילות Locking

- המטרה היא להבטיח שנוצרים רק תיזמונים ברי-סדרתיות
- דרך אחת לעשות זאת היא ע"י שימוש במנעולים (locks)
- לכל פריט יש שני סוגי מנעולים
  - מנעול משותף (shared), שמסומן ע"י S
  - מנעול בלעדי (exclusive), שמסומן ע"י X

36

לאיזה צורך נועלים?

- כדי לקרוא פריט A צריך תחילה לנעול אותו במנעול משותף
  - בעקרון אפשר לשחרר את המנעול אחרי הקריאה
- כדי לכתוב פריט A צריך תחילה לנעול אותו במנעול בלעדי
  - בעקרון אפשר לשחרר את המנעול אחרי הכתיבה
- אם אחרי קריאה צריך לכתוב, אז עסקה יכולה לשדרג מנעול משותף למנעול בלעדי
- הבקשות למנעולים מופנות למנהל הנעילות

37

מתי מותר לנעול?

- מספר עסקאות יכולות להחזיק בו-זמנית מנעול משותף על פריט A
  - בתנאי שאין אף עסקה שמחזיקה מנעול בלעדי על A
- רק עסקה אחת יכולה להחזיק מנעול בלעדי על פריט A
  - ואז לאף עסקה אחרת אין מנעול משותף או בלעדי על A
- מנהל הנעילות מוודא שהכללים נשמרים

38

פרוטוקול נעילות בשתי פאזות

Two-Phase Locking (2PL)

39

פרוטוקול נעילות בשתי פאזות Two-Phase Locking (2PL)

- עסקה חייבת לקבל
  - מנעול משותף לפני קריאה של פריט
  - מנעול בלעדי לפני כתיבה של פריט
- לאחר שעסקה משחררת מנעול כלשהו היא לא יכולה לבקש מנעולים נוספים
- לכן הפרוטוקול קרוי "נעילות בשתי פאזות"
  - תחילה פאזה של קבלת מנעולים
  - ולאחריה פאזה של שחרור מנעולים

40

תזמון שמקיים 2PL הוא בר-סדרתיות קונפליקטית

- *משפט*: תזמון שבו כל עסקה מקיימת פרוטוקול 2PL הנו בר-סדרתיות קונפליקטית
- *הוכחה*: יהי S תזמון שמקיים 2PL ונניח שבגרף הקדימויות יש מעגל
- המשך ההוכחה בשקפים הבאים

41

צלע מ- T_1 ל- T_2 פירושה:

- אם יש צלע $T_1 \rightarrow T_2$ אז פירוש הדבר שיש פריט כלשהו כך שאחת משתי העסקאות כותבת אותו (ולכן מחזיקה עליו מנעול בלעדי) והשניה קוראת או כותבת אותו, ולכן גם היא מחזיקה עליו מנעול
- *מסקנה*: T_1 משחררת מנעול על פריט כלשהו לפני ש- T_2 מקבלת מנעול על אותו פריט

42

אם יש מסלול $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$ אז נובע ש-

- T_1 משחררת מנעול לפני ש- T_2 מקבלת מנעול
- T_2 משחררת מנעול לפני ש- T_3 מקבלת מנעול
- T_3 משחררת מנעול לפני ש- T_4 מקבלת מנעול
- ובאופן טרנזיטיבי:
  - T_1 משחררת מנעול על פריט כלשהו לפני ש- T_4 מקבלת מנעול על פריט אחר (יתכן אבל לא הכרחי ששני המנעולים על אותו פריט)

לפיכך, אם יש מעגל $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$ אז נובע ש-

- T_1 משחררת מנעול על פריט כלשהו לפני ש- T_1 מקבלת מנעול על פריט אחר
- *מסקנה*: T_1 אינה מקיימת פרוטוקול 2PL
- מכיוון שהגענו לסתירה, נובע שבגרף הקדימויות אין מעגל והתזמון S הוא בר-סדרתיות קונפליקטית
- מ.ש.ל.

מנהל המנעולים וטיפול במצבי קיפאון

מנהל מנעולים
Lock Manager

- מנהל המנעולים מטפל בבקשות למנעולים ובשחרור מנעולים
- מחזיק טבלה שיש בה כניסה לכל פריט A עם
  - רשימת העסקאות שכרגע מחזיקות מנעול על A
  - סוג המנעול
  - מצביע לתור של בקשות למנעולים על A
- קבלה של מנעול ושחרור של מנעול צריכות להתבצע כפעולות אטומיות
- ניתן לשדרג מנעול משותף למנעול בלעדי

מצב של קיפאון
Deadlock

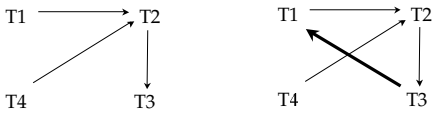
- מצב של קיפאון מתרחש כאשר קיים מעגל של עסקאות, כך שכל עסקה במעגל מחכה למנעול שכרגע מוחזק ע"י העסקה הבאה במעגל
- שתי דרכים לטפל במצבי קיפאון:
  - מניעת מצבי קיפאון
  - גילוי מצבי קיפאון וביטולם

גילוי מצב קיפאון

- בונים גרף של "מחכה ל-" (waits-for graph)
  - צומת לכל עסקה
  - צלע מ- T_1 ל- T_2 אם T_1 מחכה למנעול שכרגע T_2 מחזיקה
- מפעם לפעם בודקים אם יש בגרף מעגל (שפירושו מצב של קיפאון)
- אם יש מעגל, מבטלים את אחת העסקאות שנמצאת על המעגל, ומתחילים אותה מחדש

דוגמה

T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C) X(A)
T4: X(B)



כאמור, T_1 מחכה למנעול ש-T_2 מחזיקה

- Wait-Die – עסקה צעירה שמבקשת מנעול עלולה להתבטל
  - אם ל-T_1 עדיפות גבוהה יותר, אז T_1 מחכה לשחרור המנעול ע"י T_2; אחרת מבטלים את T_1 (כלומר, מבצעים לה abort) ומתחילים אותה מחדש
- Wound-Wait – עסקה צעירה שמחזיקה מנעול עלולה להתבטל
  - אם ל-T_1 עדיפות גבוהה יותר, אז מבטלים את T_2; אחרת T_1 מחכה ל-T_2

מניעת מצבי קיפאון

- נותנים לכל עסקה *חותמת זמן* (שמציינת את זמן ההתחלה של העסקה)
- זמן מוקדם יותר עדיף על זמן מאוחר יותר
- נניח ש-T_1 מחכה למנעול ש-T_2 מחזיקה
- שתי דרכים לפיהן ניתן לפעול:
  - Wait-Die
  - Wound-Wait

Wait-Die

- עסקה ותיקה שצריכה מנעול מחכה לעסקה צעירה שמחזיקה מנעול זה
- עסקה שהשיגה את כל המנעולים שהיא צריכה יכולה לסיים
- עסקה צעירה עלולה להתבטל שוב ושוב אם בכל פעם שהיא צריכה מנעול, אזי מנעול זה מוחזק ע"י עסקאות ותיקות יותר

Wound-Wait

- עסקה צעירה מתבטלת אם היא מחזיקה מנעול שצריכה עסקה ותיקה יותר
- עסקה עלולה להתבטל גם אם היא כבר השיגה את כל המנעולים שהיא צריכה, ולכן יכולה לסיים

סיכום לגבי שתי הדרכים למניעת מצבי קיפאון

- בכל אחת משתי הדרכים אין אפשרות לקיפאון
  - העסקה הותיקה ביותר במערכת אינה מתבטלת
- *הרעבה* (starvation) פירושה שעסקה לא מצליחה אף פעם להשיג את כל המנעולים שהיא צריכה
  - כלומר העסקה מתבטלת שוב ושוב ולכן אינה מסתיימת
- כדי למנוע הרעבה, כשמתחילים עסקה מחדש, צריך להשאיר לה את חותמת הזמן שהייתה לה קודם

התאוששות מנפילות

זוהי רק הקדמה קצרה ואינטואיטיבית
של המושגים הבסיסיים

55

דוגמה: העברת כסף מחשבון לחשבון

- חשבונות בנק A, B
- נעביר 100 ₪ מ- A ל- B
- התוכנית: A:=A-100
  B:=B+100
- התוכנית נכונה, אבל מה קורה אם המחשב נופל אחרי ביצוע הפקודה הראשונה?
  - הורדנו כסף מ- A ולא הוספנו ל- B
  - התוכנית הסתיימה בצורה שגויה

56

ביצוע אטומי של תוכנית

- תוכנית הפועלת על מסד נתונים צריכה להתבצע באופן *אטומי*, כלומר
  - התוכנית מתבצעת בשלמותה, או
  - שאינה מתבצעת כלל
- תוכנית *מתחייבת* (מבצעת *commit*) לאחר שביצעה את כל פעולותיה
  - אם התוכנית התחייבה, אז כל פעולותיה נשמרות
  - אחרת, שום פעולה שביצעה לא נשמרת במסד

57

מנגנון התאוששות (Recovery)

- לכל מערכת מסד נתונים מנגנון התאוששות שאחראי על הדברים הבאים:
  - לבטל את כל הפעולות של התוכנית אם היא נפלה (abort) לפני שביצעה commit
  - לוודא שכל הפעולות של התוכנית נשמרות במסד אם התוכנית ביצעה commit
- מאוחר יותר נסביר איך פועל מנגנון ההתאוששות

58

תזמון שמאפשר התאוששות

59

קריאה מלוכלכת Dirty Read

- נניח שעסקה T_1 קוראת ערך של פריט A שנכתב ע"י עסקה T_2
- בהמשך עסקה T_2 מתבטלת
  - צריך לבטל גם את T_1! (כי קראה ערך שלמעשה לא קיים)
  - נוצר מפל הפלות (cascading aborts)
  - אם T_1 התחייבה (עשתה commit) לפני שמספיקים לבטל אותה, אז התזמון אינו מאפשר התאוששות (schedule is not recoverable), כי אין דרך לסיים את T_2 ואין דרך לבטלה (כי ביטולה מחייב גם את הביטול של T_1)

60

תזמון שמאפשר התאוששות (Recoverable Schedule)

- תזמון מאפשר התאוששות אם עסקה מתחייבת רק אחרי (ובתנאי ש-) כל העסקאות שאת ערכיהן היא קראה התחייבו
- בפועל מקפידים על דרישה מחמירה יותר
  - עסקה רשאית לקרוא רק ערכים שנכתבו ע"י עסקאות שכבר התחייבו
- תזמון שמקיים תנאי זה מאפשר התאוששות (recoverable) ומונע מפל הפלות

פרוטוקול 2PL מחמיר Strict 2PL

- כמו 2PL הרגיל ובתוספת הדרישה הבאה
  - עסקה משחררת המנעולים שהיא מחזיקה רק אחרי שהתחייבה
- הפרוטוקול המחמיר מונע מפל הפלות ומאפשר התאוששות, כאשר צריך לבטל עסקה כלשהי

בעיית הפאנטומים

עסקאות במסד דינמי: בעיית הפאנטומים

- EMPS(Name,Dept,Age)
- T_1 אמורה למצוא את העובד המבוגר ביותר ב- CS וב- EE
- T_2 מוסיפה עובד חדש ל- CS שהוא המבוגר ביותר, ומוחקת מ- EE את העובד המבוגר ביותר

תזמון אפשרי שמקיים 2PL

- T_1 נועלת את כל הרשומות של CS ומוצאת את העובד המבוגר ביותר (שגילו 71)
- T_2 מוסיפה עובד ל- CS שגילו 75
  - אפשר לעשות זאת בזמן ש- T_1 מחזיקה מנעולים על כל הרשומות של CS שקיימות כרגע במסד
- T_2 נועלת את כל הרשומות של EE ומוחקת מ- EE את העובד המבוגר ביותר (שגילו 80)
  - T_2 משחררת את כל המנעולים שהיא מחזיקה
- T_1 נועלת את כל הרשומות של EE ומוצאת העובד המבוגר ביותר (שגילו 63)

אין תזמון סדרתי שקול

- אם T_1 רצה ראשונה, אז היא צריכה להחזיר גיל 71 ל- CS וגיל 80 ל- EE
- אם T_1 רצה שניה, אז היא צריכה להחזיר גיל 75 ל- CS וגיל 63 ל- EE
- הבעיה נוצרה מכיוון ש- T_1 מחזיקה מנעולים על הרשומות שהן כרגע ב- CS, אבל לא על הרשומות שמתווספות ל- CS תוך כדי פעולתה

מסקנה

- בר-סדרתיות קונפליקטית מבטיחה סדרתיות רק אם קבוצת הפריטים שפועלים עליה הנה קבועה

67

פתרון אפשרי: נעילת פרדיקטים Predicate Locking

- עסקה צריכה לקבל מנעול על כל הרשומות שמקיימות תנאי (פרדיקט) מסוים, למשל
  - AGE > 50 או DEPT = CS
- אם אין נעילת פרדיקטים במקרה הכללי (זה תהליך יקר),
  - אז קל יותר לנעול את כל הרשומות שמקיימות את התנאי DEPT = CS אם יש אינדקס על DEPT
    - ←נועלים את כל הבלוקים שבהם נמצאים (או צריכים להימצא) נתוני הכניסה עבור DEPT=CS
  - אם אין אינדקס, צריך מנגנון שמודא שלא מתווספות רשומות ל- EMPS, כאשר עסקה נועלת רשומות של EMPS שמקיימות את התנאי DEPT = CS

68