חישוב ואופטימיזציה
של שאילתות
חלק 1
Query Evaluation
and Optimization
Part 1

1

הקדמה:
הצגת הבעיה
והגורמים המשפיעים

2

המטרה: חישוב שאילתות בצורה יעילה ככל האפשר

- הבעיה: כמות הנתונים גדולה מאוד
  - אי אפשר לקרוא את כל הנתונים בבת אחת לזיכרון הפנימי ואז להתחיל בחישוב
  - צריך לחלק את החישוב לשלבים – בכל שלב:
    - קוראים חלק מהנתונים לזיכרון הפנימי
    - מחשבים חלק מהתוצאה לפי הנתונים בזיכרון הפנימי
    - כותבים לדיסק את החלק של התוצאה שחושב
    - עוברים לשלב הבא

3

הזמן הנדרש ל- I/O הנו הגורם העיקרי המשפיע על זמן החישוב של שאילתה

- החישוב המתבצע ע"י ה- CPU על הנתונים הנמצאים בזיכרון הפנימי אינו מסובך
- הקריאה מהדיסק והכתיבה לדיסק הנם איטיים (לפחות) פי אלף מהפעולות המתבצעות בזיכרון הפנימי
- לפיכך, הזמן הנדרש לקריאה מהדיסק ולכתיבה על הדיסק הנו הגורם העיקרי המשפיע על זמן החישוב של שאילתות

4

הגורמים העיקריים המשפיעים על הזמן הנדרש ל- I/O

- ארגון הנתונים על הדיסק
- השיטה בה מבצעים כל אחת מהפעולות
  - קיימות מספר שיטות שונות לביצוע פעולת הצירוף (כנ"ל לגבי פעולת הבחירה)
    - הזמן הנדרש ל- I/O תלוי בשיטה
- התוכנית שנבחרה לחישוב השאילתה כולה, קרי
  - השיטה שנבחרה לביצוע כל אחת מהפעולות
  - הסדר שנבחר לביצוע הפעולות

5

הנושאים העיקריים שנלמד

- כיצד מאורגנים נתונים על הדיסק וכיצד מבצעים פעולות קלט-פלט (I/O)
- שיטות לחישוב כל אחת מהפעולות האלגבריות
  - דגש מיוחד על שיטות לחישוב הצירוף, שהיא הפעולה היקרה ביותר והמורכבת ביותר
- אלגוריתם לאופטימיזציה של שאילתה, שמוצא את
  - השיטה האופטימלית לביצוע כל אחת מהפעולות, ואת
  - הסדר המהיר ביותר לביצוע הפעולות

6

דוגמה

- $R(A,B) \bowtie S(B,C) \bowtie T(C,D)$
- צירוף הוא אסוציאטיבי, לכן יש שתי אפשרויות לחישוב הביטוי הנ"ל:
  - $R(A,B) \bowtie (S(B,C) \bowtie T(C,D))$
  - $(R(A,B) \bowtie S(B,C)) \bowtie T(C,D)$
- צירוף הוא גם קומוטטיבי, לכן יש אפשרות נוספת
  - $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$
- מהי האפשרות המהירה ביותר?

המשך הדוגמה

- כאמור יש 3 אפשרויות לסידור 2 פעולות צירוף
  - $R(A,B) \bowtie (S(B,C) \bowtie T(C,D))$
  - $(R(A,B) \bowtie S(B,C)) \bowtie T(C,D)$
  - $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$
- חלק מהגורמים המשפיעים על הסדר האופטימלי
  - ארגון היחסים על הדיסק
  - גודל תוצאת הביניים
    - ← עדיף לבצע תחילה צירוף שהתוצאה שלו היא יחס קטן ככל האפשר

שיטות עיקריות לחישוב הצירוף של שני יחסים

- צירוף היא הפעולה האלגברית היקרה ביותר וקיימות עבורה מספר שיטות חישוב שונות:
  - Block Nested-Loops Join
  - Index Nested-Loops Join
  - Sort-Merge Join
  - Hash Join
- בהמשך, נתאר כל אחת מהשיטות הללו
- כמו כן, נתאר שיטות לחישוב הפעולות האלגבריות האחרות

דוגמה לבעיית האופטימיזציה

- עבור $R(A,B) \bowtie S(B,C) \bowtie T(B,D)$ צריך לבחור
  - שיטה מתאימה לכל אחת משתי פעולות הצירוף
    - ← אין צורך לבחור אותה שיטה לשתי הפעולות
  - סדר מתאים לביצוע הפעולות
- יש מספר רב של אפשרויות לבחור מתוכן
- הבחירה של השיטה לכל פעולה עשויה להיות תלויה בסדר הפעולות
  - השיטה האופטימלית לחישוב הצירוף בין R לבין S עשויה להיות תלויה בשאלה האם צירוף זה מתבצע ראשון או שני

מדוע אופטימיזציה היא חשובה?

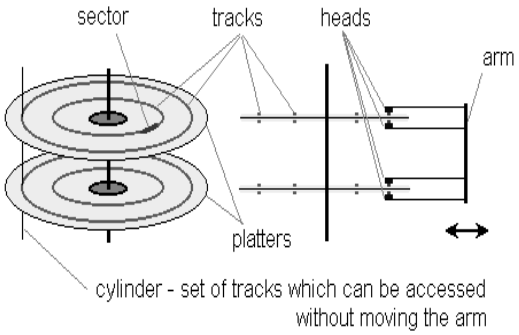
- בין התוכנית האופטימלית לחישוב שאילתה לבין תוכנית גרועה לחישוב אותה שאילתה יכול להיות הבדל עצום
  - שעות לעומת שניות ואולי אפילו הבדל גדול יותר
- לא כל כך חשוב לבחור את התוכנית המהירה ביותר – חשוב להימנע מבחירת תוכנית גרועה

ארגון הנתונים על הדיסק, מבנה הדיסק ואופן הביצוע של פעולות I/O

אחסון יחסים

- יחסים מאוחסנים בקבצים על דיסקים
- דיסק מורכב מבלוקים (דפים) בעלי גודל אחיד
- בלוק חייב להיקרא או להיכתב בשלמותו
- הזמן לקריאת או כתיבת בלוק אינו אחיד ותלוי במיקום הבלוק על הדיסק ובמצב הדיסק
  - כלומר, היכן נמצאת הזרועה יחסית למיקום הבלוק
- הקריאה והכתיבה מדיסק הנם איטיים לפחות פי אלף מקריאה וכתיבה מהזיכרון הפנימי

13



14

הנחות מפשטות

- מניחים זמן קבוע לקריאה או כתיבה של בלוק
  - בלוק טיפוסי מכיל בין 1K ל- 4K בתים (bytes)
- כאמור, המחיר של ביצוע שאילתה הנו הזמן הנדרש לקריאה וכתיבה מהדיסק, תוך התעלמות מהזמן שלוקחות פעולות ה- CPU
  - זאת הנחה סבירה, כי פעולות הדיסק לוקחות הרבה יותר זמן וניתנות לביצוע במקביל לפעולות ה- CPU

15

אחסון יחס כקובץ

- כל יחס מאוחסן כקובץ נפרד
- קובץ בנוי מבלוקים (הנקראים גם דפים)
- בכל בלוק רשומות רבות
  - עשרות רשומות בבלוק בהחלט אפשרי
- פעולה על רשומה מתבצעת ע"י קריאת הבלוק המכיל את הרשומה לזיכרון הפנימי, ביצוע הפעולה, וכתיבת הבלוק בחזרה לדיסק

16

אחסון רשומות בבלוק

- רשומות יכולות להיות באורך קבוע או באורך משתנה
- שיטת האחסון צריכה לתמוך בביצוע יעיל של
  - מציאת רשומה בתוך בלוק
  - ניצול מקום המתפנה ממחיקת רשומות

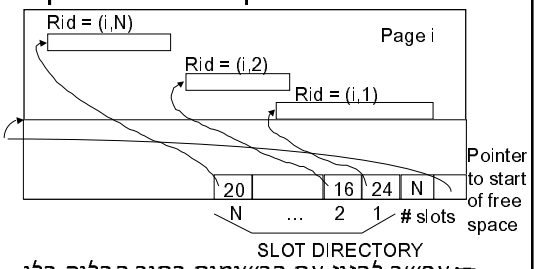
17

הרעיון העיקרי של ארגון רשומות בבלוק

- רשומה מזוהה ע"י RID (Record ID), המורכב ממספר הבלוק וממספר הרשומה בתוך הבלוק
- בסוף הבלוק יש מערך האומר היכן מתחילה כל רשומה הנמצאת בבלוק
- *הערה*: ה- RID של רשומה יכול להופיע במקומות רבים במסד – בכל מקום שבו יש הפניה או התייחסות לרשומה
  - לכן, אי אפשר לשנות את ה- RID (אלא במחיר מאוד יקר)

18

הדגמה של ארגון רשומות בבלוק



☛ *אפשר להזיז את הרשומות בתוך הבלוק בלי לשנות את ה- RID שלהן*

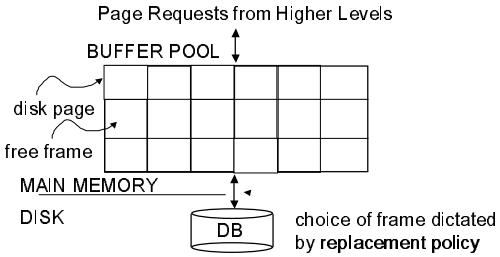
19

חוצץ (Buffer)

- חלק מהזיכרון הפנימי מוקצה לחוצץ, שלתוכו נקראים בלוקים מהדיסק
- תוכניות האפליקציה פועלות על הבלוקים (דפים) הנמצאים בחוצץ
- כאשר מסתיים השימוש בבלוק הנמצא בחוצץ, ניתן לנצל את מקומו לטובת בלוק אחר
  - אם הבלוק השתנה בזמן שהיה בחוצץ, צריך לכתוב אותו לדיסק לפני שמפנים את מקומו לטובת בלוק אחר

20

ניהול חוצץ



- *Data must be in RAM for DBMS to operate on it*
- *Table of <frame#, pageid> pairs is maintained*

21

כיצד מחליטים איזה בלוק יפנה את מקומו?
(מדיניות החלפה – Replacement Policy)

- LRU (Least Recently Used) – מחליפים את הבלוק שנעשה בו שימוש הכי רחוק בעבר
  - מדיניות מקובלת בניהול זיכרון וירטואלי ע"י ה- OS
- MRU (Most Recently Used)
  - לעיתים מדיניות זאת טובה יותר, למשל במעבר סדרתי על קובץ, כאשר
    - ←צריך לעבור על הקובץ באופן סדרתי מספר פעמים
    - ←מספר הבלוקים בחוצץ קטן ממספר הבלוקים בקובץ

22

ניתן להשתמש במערכת ההפעלה לניהול החוצצים, אבל

- מערכות לניהול מסדי נתונים מעדיפות לנהל את החוצצים בעצמן, כי
  - כך קל יותר להשיג תאימות של ה- DBMS למספר מערכות הפעלה שונות
  - יש היבטים שלא קיימים במערכות הפעלה ומצריכים טיפול מיוחד
    - ←תוכניות שרצות במקביל ומשתמשות באותו בלוק
    - ←מדיניות החלפה שונה מזו שמתאימה לזיכרון וירטואלי
    - ←קריאה מקדימה של בלוקים לתוך החוצץ

23

קריאה מקדימה של בלוקים לתוך החוצץ

- כאמור, ניתן לבצע פעולות I/O במקביל לחישובים המבוצעים ע"י ה- CPU
- אם בתוכנית לחישוב שאילתה הוחלט שיש לעבור על קובץ באופן סדרתי, אז אפשר לקרוא בכל שלב מספר בלוקים לתוך החוצץ (בהתאם לגודל החוצץ), כאשר באותו הזמן ה- CPU פועל על הבלוקים שנקראו בשלב הקודם

24

סוגי קבצים

25

סוגי קבצים לאחסון יחסים

- קובץ ערמה (Heap file)
  - הרשומות מאוחסנות (ללא מיון) בבלוקים, שמשורשרים זה לזה
- קובץ ממוין (Sorted file)
  - טוב לשליפת כל הרשומות לפי סדר המיון או למציאת הרשומות בטווח מסוים של ערכים
- קובץ ערבול (Hash file)
  - פונקצית הערבול מקבלת ערכים עבור המפתח ומוצאת רשומות עם ערכים אלה

26

הפעולות על קובץ

- הוספת או מחיקת רשומה בודדת
- חיפוש לפי ערך עבור המפתח
- קובץ ערמה
  - מוסיפים בסוף הקובץ
- קובץ ממוין
  - הקובץ ממוין לפי המפתח
  - מכווצים את הקובץ לאחר מחיקה
- קובץ ערבול
  - אין overflow, 80% תפוסה

27

המאפיינים של קובץ שקובעים את הזמן הנדרש לביצוע הפעולות

- מספר הבלוקים של הקובץ – B
- מספר הרשומות בבלוק – R
- הזמן (הממוצע) לקרוא בלוק – D
- בעזרת הפרמטרים האלה נקבע את הזמן (של פעולות ה- I/O) הנדרש לביצוע הפעולות בסוגי הקבצים השונים

28

הערה

- מתעלמים מקריאה מקדימה (pre-fetching) של בלוקים
  - קריאה מקדימה חוסכת זמן, כי היא מתבצעת במקביל לעיבוד בלוקים שכבר נמצאים בזיכרון
  - אבל קשה להביא אותה בחשבון באופן מדויק
  - במקרה הגרוע ביותר אי אפשר לבצע קריאה מקדימה

29

הזמן הנדרש לביצוע הפעולות

	ערמה	ממוין	ערבול
מעבר על כל הרשומות	BD	BD	1.25BD
חיפוש רשומה לפי מפתח	0.5BD	$D\log_2 B$	D
חיפוש כל הרשומות בטווח נתון	BD	$D(\log_2 B$ + # of pages with matches)	1.25BD
הוספת רשומה	2D	Search + BD	2D
מחיקת רשומה	Search + D	Search + BD	2D

אינדקסים

- אינדקס מעל קובץ (קרי, יחס) נבנה עבור מפתח חיפוש (search key)
- מפתח חיפוש הוא אוסף כלשהו של שדות, שאיננו בהכרח מפתח של היחס
- בהינתן ערך עבור מפתח החיפוש, האינדקס מאפשר להגיע ישירות לכל הרשומות בעלות הערך הנתון (יכולה להיות יותר מרשומה אחת עם אותו ערך עבור מפתח החיפוש)

שתי האפשרויות למבנה הרשומות של אינדקס

- רשומות רגילות של יחס, כלומר האינדקס הוא חלק מקובץ שמאחסן רשומות של יחס
  - במקרה זה האינדקס חייב להיות על המפתח הראשי
  - לכל היותר אינדקס אחד יכול להיות כזה
- נתוני כניסה (data entries) לקובץ אחר
  - נתוני כניסה הן רשומות שמאפשרות להגיע במהירות לכל הרשומות, בקובץ האחר, שיש להן את הערך המבוקש עבור מפתח החיפוש
  - במקרה זה האינדקס הוא קובץ נפרד

האפשרויות למבנה של נתוני כניסה

- נתוני כניסה מהצורה (k, rid), כאשר k הוא ערך ו- rid (record identifier) הוא מצביע לרשומה של היחס עם הערך k
  - הרשומה (k, rid) היא מאורך קבוע ויכולות להיות מספר רשומות עבור ערך נתון של k
- נתוני כניסה מהצורה (k, list of rids)
  - זוהי רשומה מאורך משתנה ויש רק רשומה אחת עבור ערך נתון של k

סיווג אינדקסים: אינדקס ראשי לעומת אינדקס משני

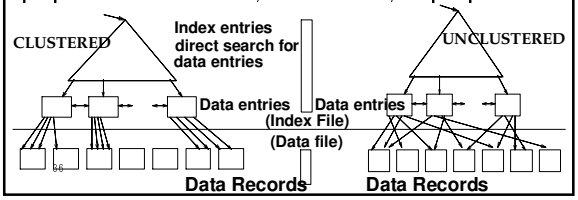
- אינדקס ראשי – מפתח החיפוש מכיל מפתח ראשי של היחס
- אינדקס משני – אחרת
- פירוש אחר: אינדקס ראשי מכיל רשומות של קובץ, בעוד שאינדקס משני מכיל נתוני כניסה

סיווג אינדקסים: אינדקס מקובץ לעומת אינדקס שאינו מקובץ

- אינדקס מקובץ (clustered index) – סדר הרשומות באינדקס זהה או "קרוב" לסדר הרשומות בקובץ
- אינדקס לא מקובץ (unclustered index) – אחרת
- אינדקס שמכיל רשומות של קובץ הוא תמיד מקובץ, אבל אינדקס מקובץ יכול להיות בנוי גם מנתוני כניסה
- לכל היותר אינדקס אחד יכול להיות מקובץ
- שליפת רשומות לפי אינדקס לא מקובץ לוקחת יותר זמן
- אינדקס מקובץ, שהנו נפרד מהקובץ עצמו, דורש תהליך יקר למדי של עדכון, כדי שיישאר מקובץ גם לאחר עדכונים תכופים של הקובץ עצמו

בניית אינדקס מקובץ עבור קובץ ערמה

- מיין את הערמה והשאר בכל בלוק שטח פנוי להוספת רשומות בעתיד
- בעתיד יתכן ויהיה צורך בבלוקים של overflow לצורך הוספת רשומות – לכן הסדר של רשומות האינדקס קרוב, אבל לא זהה, לזה של רשומות הקובץ



אינדקס צפוף ואינדקס דליל

- אינדקס דליל שומר רק חלק מהערכים של מפתח החיפוש
  - חייב להיות אינדקס מקובץ
- כדי למצוא את הרשומות של הקובץ עם מפתח k מחפשים ערך b כך ש-
  - b הערך הגדול ביותר באינדקס שעדיין אינו גדול מ- k
- ממשיכים לחפש בקובץ בסדר עולה מהרשומה הראשונה עם הערך b

מפתח חיפוש מורכב

- כאשר מפתח חיפוש כולל כמה שדות, רצוי למיין את השדות השונים לפי מיון לקסיקוגרפי (או לבחור בשיטת אכסון מתוחכמת יותר), כדי שניתן יהיה לבצע ביעילות חיפוש טווח על חלק מהשדות
- לדוגמה, אינדקס על (age, sal)
  - חפש רשומות עם age=20
  - חפש רשומות עם age=20 and sal>1000
  - ← מצא כל הרשומות עם sal=20 ובחר מתוכן את אלה עם המשכורות הגדולות מ- 1000

סוגי מבנים של אינדקסים

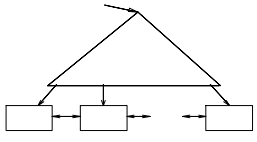
- **אינדקס מבוסס על עץ חיפוש מאוזן**
  - עץ B+ הוא הנפוץ ביותר
  - מאפשר חיפוש לפי טווח
- **אינדקס מבוסס על ערבול**
  - מחייב ערבול בר-הרחבה (extendible hashing), כלומר אפשרות להגדיל את מספר הדליים (ולשנות בהתאם את פונקצית הערבול) כאשר מתווספות רשומות
  - מאפשר חיפוש רק לפי שוויון

שקפים על עץ B+

- להלן שלושה שקפים מפרק 9 בספר של Ramakrishnan על עץ B+

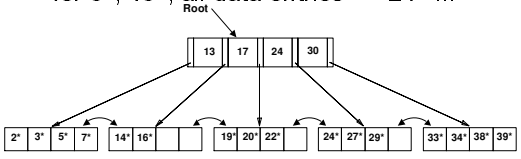
B+ Tree: The Most Widely Used Index

- Insert/delete at $\log_F N$ cost; keep tree *height-balanced*. (F = fanout, N = # leaf pages)
- Minimum 50% occupancy (except for root). Each node contains **d** <= *m* <= 2**d** entries. The parameter **d** is called the *order* of the tree.
- Supports equality and range-searches efficiently.



Example B+ Tree

- Search begins at root, and key comparisons direct it to a leaf Search for 5*, 15*, all data entries >= 24* ...



☛ *Based on the search for 15*, we know it is not in the tree!*

B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

43

נושאים נוספים על עצי +B בפרק 9

- הוספה ומחיקה של רשומות
  - צריך לדאוג שכל צומת יישאר לפחות חצי מלא
  - צריך לדאוג שהעץ יישאר מאוזן
- בנייה ראשונית של עץ +B עבור קובץ נתון
- דחיסת מפתחות בעלים הפנימיים
  - מגדיל את ה- fan-out ולכן מקטין את עומק העץ

44

נושאים נוספים שלא נכסה

- פרק 10 עוסק באינדקסים הבנויים על ערבול
- פרק 11 עוסק במיון חיצוני
  - כלומר, שיטות מיון כאשר אי אפשר לקרוא בבת אחת את כל הנתונים לתוך הזיכרון הפנימי
  - מיון חיצוני נחוץ
    - כאשר רוצים תוצאה ממוינת
    - למחיקת עותקים כפולים של רשומות (DISTINCT)
    - לצורך ביצוע group by
    - לביצוע sort-merge join (אחת השיטות של צירוף)

45