סיכום DB

29 ביולי 2022

1 שבוע 1:

1.1 הרצאה 1:

- **ארבעת השלבים בבניית מסד נתונים:**
  1 ניתוח הצרכים: מהו המידע שנרצה לשמור, איך נשתמש בו ואלו אילוצים יש על המידע.
  2 ניתוח קונספטואלי: נתרגם את השלב הקודם לדיאגרמה.
  3 תכנון לוגי: נחליט אלו טבלאות אנו רוצים לשמור ומה האילוצים שלהם.
  4 תכנון פיזי: נגדיר איפה כל פריט מידע יישב וכיצד ניגש אליו.

1.1.1 ישויות, תכונות וקשרים:

- **ישות** $entity$**:** אובייקט שניתן להבדיל בינו לבין אובייקטים אחרים בעזרת התכונות שלהם.
- **קבוצת ישויות:** קבוצת ישויות מאותו הסוג, **תסומן בעזרת מלבן** .
- **תכונות** $attributes$**:** מתארות את הישויות בתוך הקבוצה. תכונה צריכה להיות ערך יחיד, שונה מ $null$, צריכה להיות קיימת אצל כל הישויות. **נסמן בעיגול**.
- **מפתח:** קבוצה **מינימלית** של תכונות שמזהות באופן יחיד את הישויות בתוך הקבוצה.
  לכל קבוצת ישויות חייב להיות מפתח. **נסמנו בעזרת קו תחתון** מתחתיו.
- **קשר** $relationship$**:** אסוסיאציה בין 2 ישויות או יותר.
- **קבוצת קשרים:** קבוצת קשרים מאותו הסוג, קבוצת קשרים R מורכבת ממכפלה קרטזית של כל הישויות. נסמן בעזרת מעויין.
  הערה: לקבוצת קשרים גם יכולות להיות תכונות.
- **קבוצת קשרים רקורסיבית:** קבוצה שמקשרת את אותה הישות לעצמה.

- **כפילויות בקשרים:** אם אין אילוצים אזי כל אילוץ יכול להופיע מספר פעמים.
  כשנרצה לסמן שכל ישות מופיעה לכל היותר פעם אחת נסמן זאת **בעזרת חץ** $\Rightarrow$.
  כשנרצה לסמן כל ישות מופיעה בדיוק פעם אחת: נסמן **בעזרת חץ עגול**.

- **ירושה: נסמן בעזרת משולש** שבתוכו כתוב ISA. ומסמן כי הישויות יורשות את כל התכונות מהאב (כולל מפתחות).
  למעשה אם B, C יורשות מ A אזי $B, C \subseteq A$.

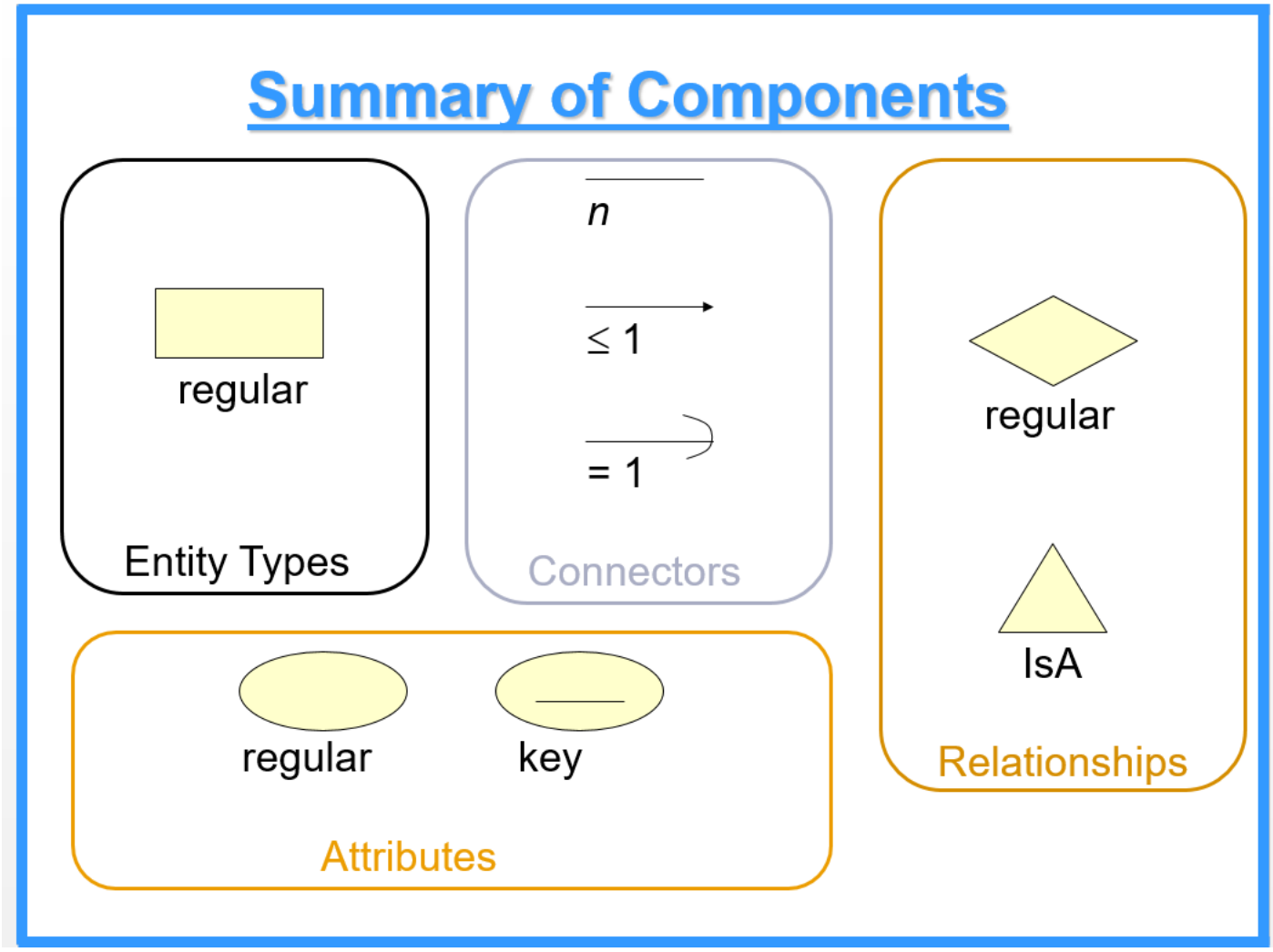
- **קבוצת ישויות חלשה:** קבוצה שהמפתח שלה בלבד לא מספיק בכדי לזהות אותה, אלא אנו צריכים מפתח של קבוצות ישויות נוספת בכדי זהות אותה.
  את קבוצת הישויות נסמן בעזרת ריבוע כפול, בנוסף נסמן את קבוצת הקשרים במעויין כפול, כך נסמן כי המפתח נמצא בקבוצת הקשרים הזו.

1.1.2 מעבר מדיאגרמת ישויות לבסיס נתונים:

- **תכנון לוגי:** תחילה נצייר דיארמה לאחר מכן נבנה מודל יחסי (רלציות).

- **עקרונות לתרגום אלמנטים בדיאגרמה:**
  ישויות: נקח את שם קבוצת הישויות כשם היחס, את קבוצת התכונות כתכונות היחס, המפתח נשאר אותו הדבר.
  קבוצת קשרים: תהפוך ליחס במודל הרלציוני, נתרגם באותו האופן של יחס (כל קבוצת ישויות תוגדר כתכונה).
  המפתח יהיה המפתחות של כל קבוצות הישויות שקשורות לקבוצת הקשרים.
  קבוצת קשרים עם חץ: נתרגם באותו האופן, אך כעת המפתח יהיה המפתח של קבוצות הישויות אליהן החץ **לא** מצביע (ולא כל המפתחות מכל קבוצות הקשרים).
  קבוצת קשרים עם חץ מעוגל: נתרגם באותו האופן, אם יש קשר בינארי - נוכל להכניס את קבוצת הקשרים בתור תכונה של קבוצת הקשרים שהחץ **לא** פונה אליה.
  אם יותר מבינארי - נבנה אותו מספר של יחסים כפי שמופיע בדיאגרמה (א"א לצמצם).
  קבוצת ישויות חלשה: מפתח היחס - הוא המפתח עצמו ובנוסף המפתח של קבוצת הישויות שדרכו אנו מזהים את הישויות.
  ירושה: יש שלש דרכים שונות לתרגם ירושה - **1:** יצירת יחס עבור כ"א מהישויות (נשים לב להעביר את ירושת התכונות והמפתחות). **2:** ניצור יחס עבור כל קומבינציה אפשרית של ישויות. **3:** נייצר יחס אחד בלבד, ואם יש ישות שחסרות לה חלק מהתכונות, נכניס ערכי $null$ לתכונות אלו.

- **לסיכום:**



1.2 תרגול 1:

- **הערה:** עבור n ישויות, כאשר יש לנו יחס בין ישויות $e_1...e_n$, ויש חץ מהקשר בניהן לישות e_n, אזי החץ מתייחס לכל הישויות $e_1...e_{n-1}$ **יחד** ולא לכל ישות בנפרד. כלומר - היחס צריך להתקיים על כל $n-1$ הישויות יחד.
- **הערה:** אם יש קשר בין כמה ישויות ויש חץ עגול הנכנס לאחת מהישויות, אזי ככל הנראה המודל לא שלם.
- **הערה:** אם יש קשר בין כמה ישויות ויש חץ הנכנס לאחת מהישויות - E, אזי E תהיה ריקה, רק אם לפחות קבוצה אחת משאר הישויות היא קבוצה ריקה.

2 שבוע 2:

2.1 הרצאה 2 - איך נייצר טבלאות ב SQL:

- $create\ table$: כשנגדיר טבלה חדשה, נציין את שמה, ובסוגריים נשים את תיאור העמודות ואת האילוצים על הטבלה.

- **אילוצים:** נחוצים בכדי לשמור על נכונות הטבלה, לכן נוסיף תמיד כמה שיותר אילוצים.

- **סוגי אילוצים:**

1 $not\ null$**:** אומר לנו שהשדה הזה לא יכול לקבל ערך $null$ - ריק.

2 ערך דיפולטיבי: שדה מסויים יקבל ערך דיפולטיבי.

3 אילוצי בדיקה: אילוצי בדיקה על שדה מסויים או על הטבלה כולה. תנאי בוליאני שבודק אם התנאי מתקיים. נממש בעזרת $CHECK()$.

4 $UNIQUE$ **- יחודיות:** שלא יכולות להיות שתי שורות בטבלה עם אותו ערך (לדוגמה - תז).

5 אילוץ מפתח ראשי: עמודה שתסומן כך תהיה יחודית - $UNIQUE$ וגם $not\ null$. ניתן לסמן פעם אחת בטבלה בלבד.

6 אילוץ מפתח זר $FOREING\ KEY$**:** מאלץ את המערכת שתבדוק שקיום הערך בטבלה אחרת מתקיים גם כן. ניתן להגדיר זאת רק אם השדה שאנו מצביעים עליו הוא $unique$ או מפתח ראשי.
לדוגמה - אם עובד מסויים עובד במחלקה 1 אזי לא ניתן להוסיף את העובד למחלקה 1 בלי לבדוק שהיא קיימת בטבלת המחלקות.

- $on\ delete\ cascade$**:** נוסיף בסוף האילוץ של $FOREING\ KEY$, במידה שנמחק מחלקה מסויימת זה יכריח את ממסד הנתונים למחוק את כל העובדים במחלקה שנמחקה (במקום לזרוק שגיאה שאי אפשר למחוק את המחלקה הזו).

- **אילוץ מפתח זר:** אם נרצה לממש טבלה של מחלקה שיש לה מנהל, אך המנהל הוא גם עובד שקשור לטבלת מועסקים, לכן בטבלת המחלקה נסמן את המנהל במפתח זר.

- **מחיקת טבלה:** נשתמש בפקודה $drop\ table\ tableName$. אם יש לנו אילוצי מפתח זר אנו צריכים לשים לב לסדר מחיקת הטבלאות.
$cascade$**:** פקודה שאומרת כי בזמן מחיקת טבלה כל אילוץ מפתח זר שהצביע על הטבלה ימחק גם כן.

- **הכנסת שורות לטבלה:** $insert\ into\ tableName$, נוסיף את שמות העמודות והערכים.
ניתן להכניס בלי לציין את שמות העמודות אם הכנסת הערכים נעשית לפי סדר העמודות.

- **המודל הרלציוני:** לטבלה נקרא יחס או רלציה, יש לו אינסטנס - אוסף השורות ביחס, יש לו סכמה - שם הטבלה עם שמות הישויות.
הערות: אותה שורה לא יכולה להופיע כמה פעמים ביחס, בנוסף אין ערכי $null$.

2.1.1 פעולות:

- **פעולת הטלה -** π**:** מקבלת יחס עם n עמודות, לייד פעולת ההטלה נציין את שמות העמודות עליהן אנו רוצים להטיל. ומחזירה את היחס שמכיל את העמודות שהזכרנו. למעשה היא מחזירה את **העמודות** שהזכרנו שמופיעות ביחס.

- **פעולת הבחירה** σ_C**:** מקבלת יחס, לייד פעולת הבחירה נתאר תנאי בוליאני C מעל שורה בודדת, התוצאה היא **השורות** שמקיימות את התנאי.

- **איחוד** $\cup$: מקבלת שני יחסים שחייבים להיות מתאימים (עם אותה הסכימה), ומחזירה את איחוד השורות (ללא כפילויות).

- **חיסור** $-$ ": מקבלת שני יחסים שחייבים להיות מתאימים (עם אותה הסכימה) ומחזירה את חיסור השורות.

- **מכפלה קרטזית** $\times$: מקבלת שני יחסים ומחזירה מכפלה קרטזית בניהם (**משרשרת שורות** של שתי טבלאות אחת לייד השניה).

- **שינוי שם** ρ: מקבל יחס, ומחזיר לו שם חדש.

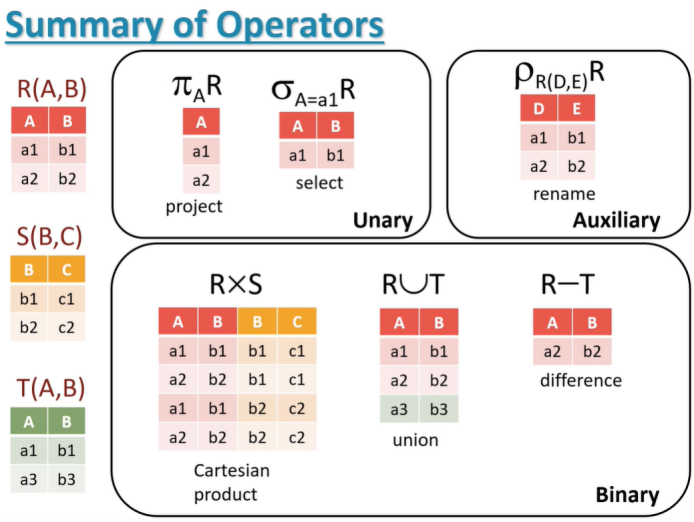
- **חיתוך** $\cap$: עבור שתי יחסים R, S מתקיים $R \cap S = R - (R - S)$.

- **צירוף על תנאי** $join = \bowtie_C$: מכפלה קרטזית של שני יחסים ותנאי על תוצאת ההכפלה, $\sigma_C(R \times S) = R \bowtie_C S$.

- **צירוף טבעי** $\bowtie$: כפל של שני יחסים, ותנאי שדורש שוויון על כל האטרביוטים (עמודות) המשותפים (מופיעים בשני היחסים).

- **חילוק** $\div$: נחלק יחס R ביחס S, נוכל לעשות זאת **רק אם** $S \subseteq R$ (הסכימה של S מוכלת ב R). מה שנקבל בתוצאה הם האטריביוטים שנמצאים ב R ולא ב S.
  השורות שיהיו בתוצאה הן - תתי שורות של R עם העמודות הלא משותפות, כך שלכל שורה מהעמודות המשותפות יש טאפל של העמודות שלא משותפות ב R.
  כלל אצבע: לרוב כשנראה שאילתה של "כל" נצטרך להשתמש בפעולת חילוק והטלה.

- **סיכום הפעולות:**



- **משרשרת שורות**

- **שקילות:** נאמר ששני ביטויים $\mathrm{E}_1, \mathrm{E}_2$ הם שקולים, ונסמן $\mathrm{E}_1 \equiv \mathrm{E}_2$ אמ"מ הם תמיד מחזירים את אותה התוצאה.
  הוכחת שקילות: נעשה באמצעות כלים מתמטיים וכללי האופרטורים.

- **ייעול החישוב:** SQL ממזער את התוצאות ע"י הפעלת שקילויות. בנוסף נפעיל תחילה פעולות בחירה והטלה, בנוסף נשים לב לסדר ביצוע הצירופים. כך נחסוך בזמן ופעולות.

2.2 תרגול 2 - תרגום דיאגרמות לטבלאות:

- **קשר בינארי עם חץ עגול:** אם יש חץ עגול מישות A לישות B אזי ניתן להכניס את הנתונים (כולל היחס בניהם) של הישות B לטבלה של A (בקשר שאינו בינארי זה לא נכון).

3 שבוע 3:

3.1 הרצאה 3:

- **חמשת הפעולות הבסיסיות באלגברה רלציונית:** $\sigma, \pi, \times, -, \cup$
- **אי תלות של האופרטורים הבסיסיים באלגברה רלציונית:** חמשת הפעולות הבסיסיות לא ניתנות לביטוי אחת באמצעות האחרת - כלומר הן ב"ת, ואנו צריכים את כולן.

3.1.1 SQL:

- **שאילתה:** מקבלת טבלאות, ומחזירה טבלה כפלט. הוצאה לא נשמרת באף מקום אלא אם ביקשנו זאת מפורשות.
- **ההיררכיה:** שאילתה $\Leftarrow$ אלגברה רלציונית $\Leftarrow$ ביטוי שקול יעיל $\Leftarrow$ חישוב בעזרת Query Evaluator $\Leftarrow$ טבלה.
- **אתר לניסוי על נתונים ושאילתות:** http://sqlfiddle.com/!17/4ccef/2
- **הגדרה של שאילתה:** הגדרת שאילה תכלול את שלשת הרכיבים הבאים $SELECT = \pi, FROM = \times, WHERE = \sigma$.

ם בסיסיים שחייבים להופיע בכל שאילתה. רכיבים אופציונלים נוספים הם $WHWRW, GROUP\ BY, HAVING\ ORDER\ BY$, אם הם יופיעו אזי הם יופיעו בסדר הזה.

- **תכונה:** השפה תבדיל בין אותיות גדולות לקטנות רק כאשר המילה תופיע בתוך "".
- **הורדת שורות כפולות - DISTINCT :** כשנטיל עם המילה DISTINCT בשדה $SELECT$ התוצאה שתחזור תמחק שורות כפולות.
- **ביטול הטלה "*":** כשנרצה לבצע בחירה בלי הטלה נוסיף אחרי $SELECT$ את הסימן $*$ במקום שמות עמודות.
- **שם לעמודה חדשה:** כשנרצה לחשב יחס בין שתי עמודות כך שהשאילתה תחזיר עמודה שלא קיימת, אזי שם העמודה החדשה יהיה $?colom?$. כדי לבחור שם נוסיף $AS\ name$.
- **הפעלת פונקציות:** בשורת ה $SELECT$ ניתן דהפעיל פונקציות על ערכים (כפל בקבוע ועוד).
- $WHERE$: נוכל לשים שם תנאים מורכבים כגון השוואות וחלקים לוגים.
- $LIKE$: נוסיף בשורת ה $WHERE$, השוואה בין סטרינגים - דומה לשימוש ברג'קסים.

- $ORDER\ BY\ row$: ניתן לציין באיזה סדר אנו רוצים למיין את התוצאה (לפי סדר גילאים, תז ועוד). בנוסף ניתן להגדיר אם אנו רוצים למיין בסדר עולה ASC או יורד $DESC$.

- $FROM = \times$: תסמן מכפלה קרטזית - אלו שמות של טבלאות אנו רוצים להכפיל.
הערה: כשנכתוב כמה טבלאות ב $FROM$, אנו נצטרך להשתמש ב $WHERE$ כך: $tablename.colum = ...$ אם יש אותם שדות בטבלאות שונות.

- $Aliasing$: קיצור שמות של טבלאות, נעשה זאת כך: $FROM\ tabelName\ A$ כאשר A תסמן את הקיצור שנבחר.

- **צירוף טבעי:** ניתן לכתוב בשדה $FROM$ את התנאי שנרצה בין הטבלאות כך - A Natural Join B, במקום לכתוב זאת בשדה $WHERE$.

3.1.2 פעולות על קבוצות:

- **פקודות של פעולות על קבוצות:** $UNION = \cup,\ EXEPT = -,\ INTERSECT = \cap$.

- **כיצד נממש:** נרשום שאילתה שלמה, אחכ את שם הפעולה ואחכ שאילתה נוספת.

- **הערה:** שמות העמודות יקבעו לפי השאילתה הראשונה.

- **כפילויות:** בדיפולט שורות כפולות ימחקו, אם נרצה שהן לא ימחקו נוסיף לייד הפקודה את הפקודה $UNION\ ALL$.

3.1.3 תתי שאילתות:

- **תתי שאילתות:** $SELECT, FROM, WHERE, HAVING$ כולן פקודות שיכולות להכיל תתי שאילתות.

- **תתי שאילתות מתואמות:** תת שאילתה יכול הלהתייחס לתת שאילתה אחרת.

- **כיצד נכתוב תת שאילתה:** בשדה $WHERE$ נוסיף את תת השאילתה שתופיע כחלק מביטוי בוליאני. לכן השימוש בתת שאילתה יעשה כדי לייצג ביטוי בוליאני מורכב.
נשתמש בפקודות $IN\backslash NOT\ IN$, ואח"כ תת שאילתה שתופיע בתוך סוגריים.
למעשה אנו בודקים אם תת השאילתה נמצאת או לא נמצאת בתוך השאילתה הראשית.

- **אינטואיציה:** נתחיל תמיד לפענח את השאילתה הפנימית ביותר, ונעלה בהיררכיה.

- **סינטקס נוסף - אופרטור אריתמטי:** נשתמש באופרטור, אחכ בפקודות $ANY\backslash ALL$ ואחכ נכתוב את תת השאילתה.

- **דרך נוספת -** $EXISTS$: מחזיר ערך אמת אם תת השאילתה לא ריקה, עבור $NOT\ EXISTS$ יוחזר ערך אמת אם תוצאת השאילתה ריקה.

- **הערה:** תת שאילתה שמופיעה ב $FROM$ צריכה לקבל שם - $alias$.

3.2 תרגול 3:

- **שינוי של של טבלה או עמודה (באלגברה):** נשתמש ב ρ שתסמן שינוי שם, ונכתוב את השם החדש של הטבלה ואת השדות נשים בסוגריים.
- **הערה:** כשמבצעים חיסור צריך לשים לב שהוא מתבצע על מפתח. אחרת נצטרך לבדוק מקרי קצה.

3.2.1 SQL:

- **איחוד:** נשתמש בפקודה $UNION$ ונחבר בין שתי שאילתות (אם נרצה כסילויות נוסיף את הפקודה ALL).
- **שינוי שם של טבלה**: נכתוב בשדה $FROM : TABLE\ new\ name$.

4 שבוע 4:

4.1 הרצאה 4:

4.1.1 פונקציות הקבצה:

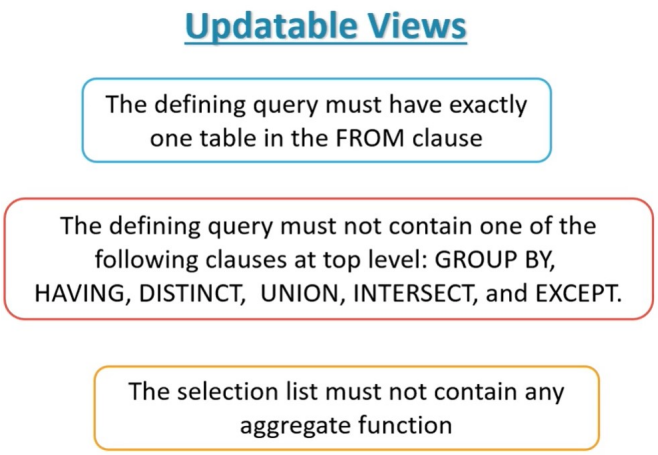
- **פונקציית הקבצה:** פונקציה שמקבלת סט של ערכים ומחזירה ערך בודד. לרוב הן יופיעו בשדה $SELECT$.
- $DISTINCT$**:** ניתן להוסיף בתוך הפונקציה את המילה $DISTINCT$ כך היא תספור שורות **שונות** בלבד.
- sum**:** תקבל סט ותתחזיר את הסכום.
- $count(*)$**:** סופרת את מספר השורות.
- $count(A)$**:** עבור אטריביוט A- סופר את מספר הערכים ב A (ללא ערכי $null$, אם כל הערכים $null$ - תחזיר 0).
- $avg(A)$**:** יחזיר את הממוצע של A.
- $GROUP\ BY\ (A_1, A_2...)$**:** כשנרצה לחשב פונקציות הקבצה לקבוצות שונות של ערכים (שורות עם אותן ערכים יכנסו לאותה קבוצה), ולקבל את התוצאה עבור כל קבוצה.
- **השדה** $HAVING$**:** שדה שמייצג תנאי בוליאני שמופעל מעל הקבוצות שיצרנו בשדה $GROUP\ BY$.
- **הערה:** כשבשדה $SELECT$ יש פונקצית הקבצה שמופעלת על שדה אחד, ויש שדה נוסף. אזי השדה הנוסף צריך להופיע בשדה $GROUP\ BY$.
- **הערה:** כל אטריביוט שמופיע בשדה $HAVING$ מחוץ לפונקציית הקבצה , צריך להופיע בשדה $GROUP\ BY$.
- **הערה:** לא ניתן להפעיל פונקציות הקבצה אחת על השניה.
- **כתיבת תת שאילתה בשדה** $SELECT$**:** מותרת רק אם היא מחזירה ערך בודד.

4.1.2 התמודדות עם ערכי $null$:

- **טבלאות אמת:** כשניתקל בערך $null$ נחזיר $unknown$
- **ביטויים:** כל ביטוי מתמטי עם $null$ יחזיר $null$.
- **שויונות:** שוויון בין ערכים כשאחד מהם $null$ יחזיר $null$.
- **בדיקה:** ניתן לבדוק $x\ is\ null$.
  דרך נוספת: נוסיף בשדה $WHERE$ ביטוי או אי שוויון
- $NATURAL\ LEFT\ OUTER\ JOIN$**:** בדומה לצירוף טבעי, אך בשונה הוא חיצוני - על הצירוף השמאלי. כל ערך שמצטרף יקבל שורה, שאר הערכים יקבלו ערך $null$.
- $NATURAL\ RIGHT\ OUTER\ JOIN$**:** באותו האון רק עבור הטבלה הימנית.
- $FULL\ OUTER\ JOIN$**:** משאירים כל מה שבצירוף הטבעי, ושאר השורות שלא הצטרפו **בשתי הטבאות**, יקבלו ערכי $null$.

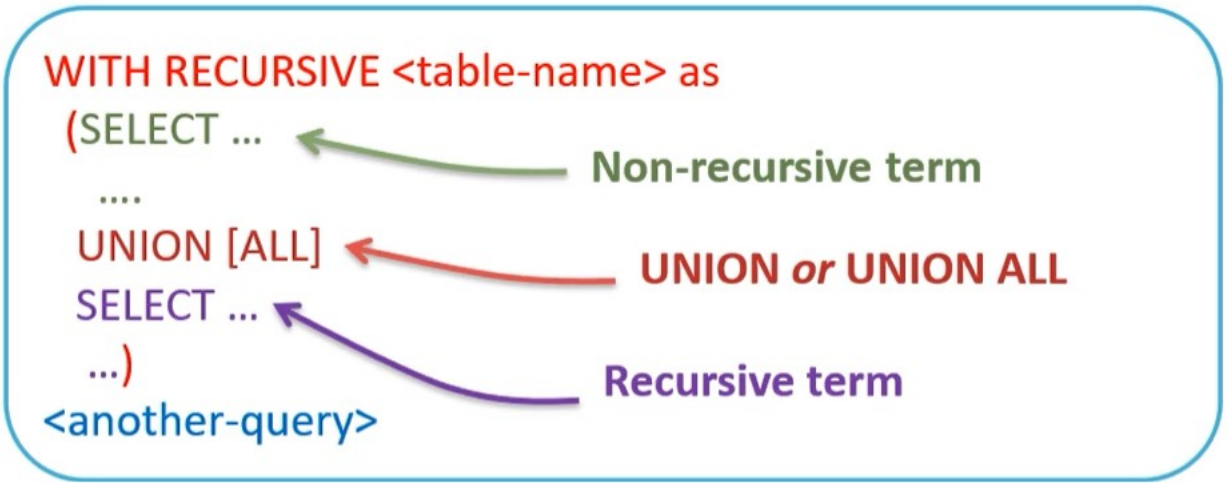
4.1.3 פיצ'רים נוספים:

- **שינויים בשורות:** עדכון נתונים שנמצאים בטבלאות נעשה באופן הבא -
  הכנסת שורות חדשות: נעשה באמצעות $insert$.
  מחיקת עמודה מטבלה: נעשה באמצעות $DELETE\ FROM\ table$, לאחר מכן בשדה $WHERE$ נוסיף את העמודות שאנו רוצים למחוק (ניתן להוסיף תנאי שימחק שורות שלא מקיימות את התנאי, או להשאיר ריק וכל העמודה תימחק).
  עדכון עמודה: נעשה באמצעות $UPDATE\ table$ לאחר מכן נוסיף שדה SET ונכתוב בו את העדכון.
- $view$**:** טבלה וירטואלית המוגדרת ע"י שאילתה (בדומה למאקרו).
  נגדיר כך: $CREATEVIEW\ name$ והתוכן של הטבלה הוא תוצאת השאילתה. לאחר מכן נוכל להשתמש בטבלה בתור טבלה רגילה.
- **מתי נשתמש ב** $view$**:** אם אנו משמשים הרבה פעמים בשאילתה מסויימת. סיבה נוספת - אבטחת מידע, באופן זה ניתן להסתיר מידע מהמשתמשים וניתן להם גישה רק לטבלה הוירטואלית.
- **הרשאות:** נשתמש בפקודה $grant$ בכדי לתת הרשאות למשתמש מסויים.
- **עדכון** $view$**:** ניתן להשתמש ב $insert, update$ ולעדכן את ה $view$.
  תנאים ש $view$ **צריך לקיים כדי שנוכל לערוך אותו:**



בלה זמנית: טבלה שמוגדרת לצורך חישוב שאילתה מסויימת ולאחר כן היא נמחקת. נשתמש בפקודה $WITH\ name\ AS\ (quwry)$.

- **רקורסיה:** נשתמש בפקודה $WITHRECURSIVE < table - name > as$



- **מימוש:** מאחורי הקלעים מוגדרות שתי טבלאות - טבלת עבודה וטבלת ערכים סופיים. שלבי הרקורסיה מחושבים בטבלת העבודה והתוצאה נכנסת לטבלה הסופית. החישוב יסתיים כשטבלת העבודה תהיה ריקה.

4.2 תרגול 4:

- all: ניתן לכתוב את התנאי בתת שאילתה של $WHERE$ ולחפש את כל השורות שלא מקיימות את התנאי.
- $not\ exist$: ניתן לכתוב את התנאי בתת שאילתה ולמצוא את כל השורות שמקיימות - לא קיים A כך ש B.
- $not\ in$: ניתן לכתוב את התנאי בתת שאילתה ולמצוא את כל השורות שמקיימות A לא נמצא בתוך הרשימה B.
- צירוף תוצאות - $agrigation$: נוסיף את השדה $GROUP\ BY$ שם נשים את התנאי לפיו אנחנו רוצים לקבץ.

5 שבוע 5:

5.1 הרצאה 5 - אינדקסים:

5.1.1 אחסון המידע:

- **איך הדאטה מאוחסן:** הוא נשמר על זיכרון הדיסק - שם הוא לא יכול להימחק. אך כדי לעבד אותו אנחנו צריכים להזיז אותו לזיכרון המרכזי כדי להשתמש בו. נעביר אותו ביחידות שיקראו דפים או בלוקים, ניתן להעביר רק ביחידות של כפולה טבעית של דפים.

- **פעולת הדיסק:** יש ראש קורא ודיסקיות, כשאנו רוצים לקרוא מידע צריך לסובב את הדיסק ולהזיז את הראש הקורא למקום הרצוי.
  יש שני סוגי זכרון דיסק: SSD, HDD.

- **הזכרון הראשי:** מנוהל ע"י $buffer\ manager$ והוא מחליט היכן לאחסן את הדאטה שנכנס לזכרון הראשי, המידע נשמר בקבצים - קבוצת בלוקים.

- **חישוב יעילות -** $I\backslash O\ complexity$**:** נחשב באמצעות מספר הבלוקים שקראנו וכתבנו לדיסק (לא כולל כתיבת התוצאה הסופית), משום שעלות העברת הבלוקים מהדיסק יותר גבוהה מעלות החישוב.

- **מיון כל המידע שעל הדיסק:** היא בעיה שאנחנו לא יודעים מה העלות שלה, משום שצריך להעביר את המידע לזכרון בחלקים ולמיין אותו כך.

5.1.2 גישה לטבלאות:

- **שמירת הטבלאות:** הטבלאות שמורות בערימה - אוסף של בלוקים שההוספה מתבצע לסופו.

- **מציאת שורה:** אנו צריכים לעבור ולקרוא n שורות בכדי למצוא שורה אחת.

- **הוספת שורה חדשה:** אנו מוסיפים לסוף הזיכרון המרכזי, וכותבים לדיסק. העלות היא 2.
  עם מפתח: אם הטבלה עם מפתח וצאיך לבדוק שלא נסתר אילוץ - צריך לקרוא את כל הבלוקים ולכן העלות היא $n+1$.

- **מחיקת שורה:** אנו צריכים לחפש את השורה, להעלות לזכרון ולמחוק אותה - לכתוב מחדש את הבלוק רק בלי השורה שנמחקה. העלות היא $n+1$.

- **מיון הדאטה:** מכיוון שניתן להעביר מידע רק בבלוקים וצריך להעביר את כל הטבלה מהערימה, אזי העלות היא יקרה. מיון יכול לייעל לנו מציאת שורה אך במקרה של הכנסת שורה נצטרך למיין את כל הטבלה מחדש.
  מתי נשמור את הדאטה ממויין:
  1: יש טבלאות שרק מוסיפים להן מידע ויש מיון טבעי שנשמר לפי סדר ההכנסה.
  2: $index\ organized\ files$.

5.1.3 $Index$:

- **מבנה** $index$: מבנה נתונים שמוגדר מעל הטבלה ומייעל מציאת שורות. למעשה זה מבנה נתונים ממויין מעל עמודה אחת. ניתן לממש אותו בעזרת עץ או טבלת האש. לכל שורה יהיה מפתח, ומצביע לשורה ב $index$, וכך נוכל למצוא את המידע ביעילות מבלי לעבור על כל הערימה.

- $B+\ Tree$: עץ עם שורש, עלים וקודקודים פנימיים. בנוסף הוא עץ מאוזן (גובה אחיד) והעלים הם רשימה משורשרת דו כיוונית כך שניתן להגיע מכל עלה לכל עלה. בנוסף הערכים בעלים ממויינים.
יש לנו מפתח חיפוש ושורות שעליהן אנו מצביעים, לכל שורה בטבלה יש עלה שמתאים לה.

- **כיצד נאחסן מידע בעץ:** כל עלה יחזיק שני מפתחות a, b כך כל שורה שערכה קטן מ a תאוחסן בצד שמאל, כל שורה שערכה בין $a \leq r \leq b$ תאוחסן באמצע. ושורות שערכן גדול מ b יאוחסנו בימין.

- **עלות החיפוש בעץ:** גובה העץ. עלות $I\backslash O$ היא עומק העץ + טיול על העלים.

- **מספר הבנים של כל קודקוד:** לכל קודקוד יש לכל יותר d ילדים עבור d דרגת פיצול של העץ ויהיה לכל קודקוד (מלבד השורש) לפחות $\lceil \frac{d}{2} \rceil$ילדים. בנוסף כל קודקוד שאינו עלה שיש לו k ילדים יהיו לו $k-1$ ערכי חיפוש (כי הערכים ממויינים לפי טווח).

- **חישוב עומק העץ:** $log_{\lceil \frac{d}{2} \rceil} n$, עבור n מספר הערכים המאוחסנים (מספר הערכים בעלים).

- **כיצד נקבעת דרגת הפיצול:** המערכת בוחרת את דרגת הפיצול ואין לנו שליטה על זה. והיא שואפת לשמור כמה שיותר מידע בכל קודקוד בודד, כך שהוא יכיל כמה שיותר מידע כדי שלא נבצע קריאות מיותרות (כל בלוק יאחסן קודקוד שלם).
הנוסחה היא: עבור מספר בייטים נסמן - $v - value, p - pointer, b - block$

$$d \leq \left\lfloor \frac{b+v}{v+p} \right\rfloor$$

למעשה אנו מחשבים שבכל בלוק בגודל b יכנסו לכל היותר $d \cdot p + (d-1)v$ בייטים שזהו גודל של קודקוד יחיד.

- **עלות חישוב שאילתה עם אינדקס:**
דרך ראשונה $index\ unique\ scan$: אנו צריכים לעשות לכל היותר מעבר 1 משורש האינדקס עד לעלה. **העלות:** h .
דרך שניה $index\ range\ scan$: אנו צריכים עבור על עוד עלים אחרי המעבר על האינדקס. **עלות:** $h+leavs$ ($leaves$ הוא המספר המקסימלי של העלים בהם נמצא המידע).
דרך שלישית $table\ access\ by\ rowid$: נשתמש בה יחד עם אחת מהשיטות הראשונות, כאשר אנו רוצים למצוא על השורה מידע שלא שמור באינדקס. מבנה העץ והטבלה שמורים בקבצים נפרדים, ונצטרך לגשת לטבלה כדי להוציא את המידע (דרך הפויינטר).

- **מתי נשתמש בכל שיטה:**
1: אם השאילתה דורשת מעבר אחד - חיפוש שורה אחת, נשתמש בשיטה הראשונה.
2: אם אנו מבקשים כמה שורות שמקיימות את התנאי נשתמש ב $range$.

3: אם צריך למצוא מידע על שורה שלא כתוב בעץ נצטרך לגשת לטבלה, לכן נשתמש ב $table\ access\ by\ rowid$.
הערה: אם צריך לעבור על רוב הטבלה נעדיף לעבוד בלי אינדקס.

- **נוסחה לחישוב המספר המקסימלי של העלים בהם נמצא המידע:** עבור m שורות (בהן נמצא המידע) ו d דרגת פיצול.

$$leaves = \left\lceil \frac{m}{\lceil \frac{d}{2} \rceil - 1} \right\rceil$$

- **כיצד נייצר אידקס:** נשתמש בפקודה כך - CREATE INDEX ON tableNme ({ columnName, ... }) .
ניתן להוסיף פקודות כמו $unique, name$. בנוסף ניתן לייצר אינדקס על ביטוי, ולהגדיר סדר.
הערה: ניתן ליצור אינדקס רק על מספר שורות בטבלה.

- **הפקודה** $Explain$**:** נכתוב אותה לפני השאילתה והיא תחזיר לנו באיזו שיטה המערכת **זמן המעבר:** המעבר מהדיסק לזכרון הראשי הוא ארוך, בעוד החישוב בזכרון הראשי יכול להיות קטן משמעותתחשב את השאילתה.

- **הפקודה** $Explain\ analize$**:** המערכת מוציאה תכנית ביצוע, מחשבת את השאילתה ומחזירה לנו מה היית העלות בפועל.

- **הערה:** מבנה האינדקס נבנה כשאנו מחליטים, והמערכת בוחרת אם להשתמש בו. כל אינדקס נוסף צורך זיכרון נוסף, ויש לו עלויות תחזוק עבור כל עדכון של הטבלה.
לכן נבנה אינדקסים לשאילתות נפוצות וחשובות.

- **אינדקסים מעל יותר מעמודה בודדת:** בכל קודקוד יהיו מספר ערכים כמספר העמודות וגם בעלים, בנוסף יש מיון ראשוני לפי השורה הראשונה שכתבנו באינדקס וכן הלאה עבור כל עמודה.
אם התנאי הוא מעל עמודה שאינה ראשונה: לא נוכל למצוא את התשובה לשאילתה ביעילות, משום שהיא ממויינת מיון ראשוני לפי העמודה הראשונה.
היעילות המרבית: תהיה כאשר יש תנאי שוויון על השדה הראשון ותנאי אי שוויון על השדה השני.

- **שיטה נוספת - הפקודה** $include$**:** בניית אינדקס מעל שדה מסויים כך שרק בעלים יתווסף שדה נוסף. נעשה זאת עם $include(colName)$

6 שבוע 6:

6.1 הרצאה 6 - יעילות פעולת $join$:

- **הגדרה - אלגוריתם חיצוני:** אלגוריתם שמבצע את החישובים הנדרשים, תחת ההנחה שלא כל הקלט נכנס לזיכרון המרכזי.

- **איך מערכת** DB **מחשבת שאילתות** $join$ **ביעילות:** בעזרת אלגוריתם חיצוני. עבור צירוף של טבלאות A, B האלגוריתם יכול לשמור בלוק אחד בלבד לכל טבלה ובלוק נוסף לפלט. נבצע את הצירוף בכל פעם על הבלוקים ששמרנו מהטבלאות

- נחשב לבלוק פלט, לאחר מכן נייצא את הפלט למסך או לדיסק (במידה וצריך לשמור את הטבלאות שצירפנו). לאחר מכן האלגוריתם ינקה את הבלוקים ויטען את הבלוקים הבאים בטבלה.

- **כיצד המערכת בוחרת את האלגוריתם:** קיימים ארבעה אלגוריתמים לייעול שאילתות $join$. בכל פעם שיש שאילתת $join$ מערכת ה DB מבצעת חישוב של העלות עם כל אחד מהאלגוריתמים ומחשבת לפי האלגוריתם הטוב ביותר.

- **נגדיר:** עבור היחסים R, S נסמן את מספר **הבלוקים** ביחס באמצעות: $B(R), B(S)$, ואת מספר הבלוקים בזכרון המרכזי נסמן ב M. את כמות **השורות** ביחס נסמן ב $T(R)$.

- **הערה:** בכל אחד מארבעת האלגוריתמים הבאים אם יש לנו פעולת בחירה נעדיף לבצע אותה קודם. כך נוכל לצמצם את מספר פעולות ה $I\backslash O$.

כיצד נחשב את העלות: עבור הטבלה שעליה אנו מפעילים את פעולת הבחירה - הקריאה הראשונה תמיד תהיה על כל הטבלה, ואח"כ הקריאות יהיו על הטבלה בגודל המצומצם (לאחר פעולת הבחירה).

6.1.1 האלגוריתם - $BlockNestedLoopsJoin$:

- **האלגוריתם - $BlockNestedLoopsJoin$:** האלגוריתם פועל בעזרת לולאות, הוא צריך לבחור מי יהיה היחס החיצוני ומי הפנימי. לאחר מכן נקרא את היחס החיצוני ביחידות של $M-2$ בלוקים, ואת היחס הפנימי נקרא בלוק בלוק. את התוצאה נכתוב לבלוק הפלט.

- **העלות:** אם היחס החיצוני הוא R, אזי: $B(R) + B(S)\left\lceil \frac{B(R)}{M-2} \right\rceil$.

- **הערה:** נעדיף לבחור את היחס הקטן יותר שיהיה היחס החיצוני בכדי לחסוך בפעולות $I\backslash O$.

6.1.2 האלגוריתם - $Index\ Nested\ Loops\ Join$:

- **האלגוריתם - $Index\ Nested\ Loops\ Join$:** אלגוריתם שעובד עם לולאה מקוננת המניחה קיום של אינדקס (על היחס הפנימי). האלגוריתם משתמש בארבעה בלוקים בזכרון המרכזי - בלוק לכל טבלה + בלוק לחישוב הפלט + בלוק למבנה האינדקס.

נגדיר יחס חיצוני ופנימי, נעבור על היחס החיצוני ונסתכל עליו שורה שורה. עבור כל שורה ביחס הפנימי, נשתמש באינדקס על השדה עליו אנו עושים $join$ כדי למצוא שורות מתאימות, ונצרף בניהן.

- **העלות:** אם היחס החיצוני הוא R - $B(R) + T(R) \cdot index\ search$.

6.1.3 האלגוריתם - $Hash\ Join$:

- **האלגוריתם - $Hash\ Join$:** משתמש בטבלאות האש בכדי למצוא שורות תואמות. נפעיל פונקציית האש על כל אחת מהטבלאות, ונעשה צירופים בין כל שני דליים תואמים בשתי הטבלאות.

- **המימוש:** נחלק כל אחד מהטבלאות ל $M-1$ דליים ונעבוד בכל שלב על טבלה אחת, בבלוק הנוסף נשמור את הקלט. בכל שלב נעלה חלק מהטבלה לזכרון, נחשב את פונקציית האש ונמיין לדליים. כאשר זכרון של דלי מתמלא - נעביר את הדלי לדיסק ונמלא מחדש במקום שהתפנה.

- **צירוף הדליים:** נקרא דליים תואמים משתי הטבלאות ונצרף בניהם, את הפלט נכתוב בבלוק נוסף שנעשה לו ניקוי בכל שלב.
- **העלות:** ייצירת האש - עבור יחס S העלות היא $2B(S)$. עלות קריאת הדליים $B(S)$. סה"כ $3B(R)+3B(S)$.
- **הערה:** בכדי שנצליח לקרוא את הדליים בזמן לינארי (מעבר בודד על כל דלי), נדאג שאחד מהתנאים הבאים יתקיים:

$$\left\lceil \frac{B(R)}{M-1} \right\rceil \leq M-2 \quad \text{or} \quad \left\lceil \frac{B(S)}{M-1} \right\rceil \leq M-2$$

- **גודל של כל דלי:** $\left\lceil \frac{B(S)}{M-1} \right\rceil$.

6.1.4 מיון נתונים במסד נתונים:

- **מיון נתונים במסד נתונים:** לא תמיד במיון תתבצע הפעולה $sort$, לדוגמא אם יש אינדקס לא נצטרך למיין.
- **שימוש במיון:** כל פעולה שדורשת מחיקה או צירוף של שורות תבצע יותר טוב אם הטבלה תהיה ממויינת,לכן נעדיף למיין לפני פעולות אלו.
  $DICTICT, GROUB\ BY, UNION, MINUS, CUT$.
- **כיצד נמיין:** בעזרת אלגוריתם חיצוני, כי לא תמיד כל הטבלה תוכל להיכנס לזיכרון.
- **מתי נוכל למיין:** רק אם מתקיים התנאי - $\left\lceil \frac{B(R)}{M} \right\rceil < M$ כלומר - כמות הסדרות הממויינות קטן מהזכרון M, כך שישאר בלוק פנוי לפלט.
- **מימוש:** נמיין בעזרת האלגוריתם $merge\ sort$ חיצוני בשני שלבים -
  1: נייצר סדרת ממויינות קצרות בעזרת אלגוריתם שממיין במקום.
  2: נמזג את הסדרות כך - נעלה את האיבר הראשון מכל הסדרות לזכרון ונכתוב לבלוק הפלט את המינימלי מבין כולם.
- **העלות:** קריאה ייצור הסדרות $2B(R)$, מיזוג הסדרות $B(R)$. סה"כ $3B(R)$.

6.1.5 האלגוריתם $Sort\ Marge\ Join$:

- **האלגוריתם** $Sort\ Marge\ Join$**:** נמיין כל טבלה לפי השדה שאותו אנו אמורים לצרף. לאחר מכן נקרא את הטבלאות במקביל ונצרף את השורות.
- **מספר הבלוקים:** האלגוריתם משתמש בשלשה בלוקים - אחד לכל טבלה + בלוק לפלט.
- **מתי נשתמש באלגוריתם:** נשתמש באלגוריתם אם התנאי הבא מתקיים:

$$\left\lceil \frac{B(R)}{M} \right\rceil < M \ and \ \left\lceil \frac{B(S)}{M} \right\rceil < M$$

העלות היא: $5B(R)+5B(S)$

נוכל למיין באופן יעיל יותר: רק אם מתקיימים שני התנאים הבאים (כל טבלה שמורה ב M_i בלוקים ממויינים. ומתקיים $\sum_i M_i < M$) -

$$\left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil < M$$

- **העלות:** מיון של שתי הטבלאות - $2B(R)+2B(S)$, מיזוג $B(R)+B(S)$. סה"כ $3B(R)+3B(S)$.

- **הבעיה:** לא תמיד הצירוף יעלה לנו $B(R)+B(S)$.
  הפתרון: נניח שהבעיה הזו לא מתרחשת אצלנו.

6.2 תרגול 6 - יעילות פעולת $join$:

- ראינו

7 שבוע 7:

7.1 הרצאה 7 - כיצד המערכת מחשבת שאילתה:

- **סטטיסטיקה:** בכדי שהמערכת תוכל לבחור איזה דרך טובה יותר לחישוב השאילתה, היא מחשבת סטטיסטיקה על הנתונים.
  לדוגמה עבור פעולת בחירה על טבלה היא תבצע קודם את הבחירה שמורידה יוצר שורות.

- **הפקודות** $ANALIZE, VACUUM$: פקודות שכשנריץ אותן המערכת תחשב את הסטטיסטיקה של הנתונים.

- **מתי מתרחש חישוב סטטיסטי:** כאשר נריץ את הפקודות $ANALIZE, VACUUM$, מאחורי הקלעים פקודות אלו רצות באופן עצמאי מידיי פעם. עבור אינדקס הסטטיסטיקה תחושב בעת יצירת האינדקס.

- **הטבלה** $pgstats$: היא טבלה בה נשמר המידה הסטטיסטי על הטבלה.

- **נסמן:** $V(R,A)$ כמות הערכים השונים שיש בשדה A של הטבלה R. **נשים לב** אם A מפתח אזי $V(R,A)=T(R)$

7.1.1 חישוב כמות השורות בתוצאה:

- **עבור** $A='a'$: עבור טבלה R ושדה A, ובהנחה כי האיברים בשדה זה מתפלגים אחיד אזי מספר השורות בתוצאה הוא: $\frac{T(R)}{V(R,A)}$.

- **עבור תנאי השוואה** $A \leq x$: אם אנו יודעים שטווח השדה A הוא $[y,z]$, אזי אחוז השורות בתוצאה הוא: $T(R)\frac{x-y+1}{z-y+1}$.

- **עבור תנאי השוואה** $A < x$**:** כשאנו לא יודעים מה הטווח של A: נניח כי שליש מהשורות מקיימות את התנאי $\frac{T(R)}{3}$.
- **עבור שני תנאים** $A =' a' \ and \ B =' b'$**:** נניח כי אין תלות בין התנאים, ולכן כמות השורות התוצאה שווה ל: $\frac{T(R)}{V(R,A)\cdot V(R,B)}$.
- **עבור שני תנאים כך ש** $A =' a' \ and \ B <' b'$**:** אם אין לנו מידע על הטווח של B, נניח ששליש מהשורות של B עוהות על התנאי, ולכן מספר השורות בתוצאה שווה ל $\frac{T(R)}{3\cdot V(R,A)}$.
- **עבור שתי השוואות** $A <' a' \ and \ B <' b'$**:** אם אין לנו מידע על הטווח שלהן, אזי התוצאה שווה ל $\frac{T(R)}{9}$.
- **עבור שאילתה עם תנאי** $A =' a' \ or \ B =' b'$ **-** or**:** נניח אי תלות, ומספר השורות בתוצאה שווה ל

$$T(R)\left(1-\left(1-\frac{1}{V(R,A)}\right)\cdot\left(1-\frac{1}{V(R,B)}\right)\right)$$

- **עבור פעולת צירוף ותנאי** $R.A = S.A$**:** נניח התפלגות אחידה, מספר השורות בתוצאה שווה

$$\frac{T(R)\cdot T(S)}{\max\{V(R,A),V(S,A)\}}$$

נשים לב כי: עבור פעולת צירוף ותנאי $R.A = S.A$ כאשר A מפתח ב R ומפתח זר ב S: גודל הטבלה בתוצאה שווה ל $T(S)$.

- **עבור תנאי של צירוף בתוספת תנאי בחירה** $R.A = S.A$ **and** $R.B =' b'$**:** נניח אי תלות, מספר השורות בתוצאה שווה ל

$$\frac{T(R)\cdot T(S)}{\max\{V(R,A),V(S,A)\}\cdot V(R,B)}$$

- **עבור תנאי צירוף על שתי עמודות:** נניח אי תלות, מספר השורות בתוצאה שווה ל

$$\frac{T(R)\cdot T(S)}{\max\{V(R,A),V(S,A)\}\cdot \max\{V(R,B),V(S,B)\}}$$

- **עבור פעולת הטלה:**

  עם כפילויות: $T(R)$.

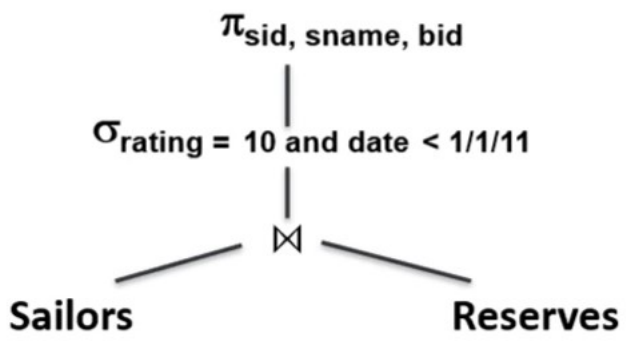
  ללא כפילויות: $V(R,A)$.

7.1.2 תכנית ביצוע שאילתה - *Query Plans*:

- **תכנית ביצוע שאילתה:** היא תיאור מדוייק כיצד אנו נחשב א השאילתה.
  1: נתרגם את השאילתה לאלגברה רלציונית.
  2: נבחר עבור כל אחת מהפעולות האלגבריות תכנית ביצוע (אלגוריתם) לפעולה.
  3: נשים פעולות בחירה והטלה בתחילת השאילתה - כמה שיותר מוקדם.

- **שלב ראשון - עץ ביצוע:** נתרגם את הביטוי הרלציוני לעץ שמייצג את הביטוי כך שהחלק הפנימי של הביטוי נמצא בעלים.
  הערה: כך נראה העץ הראשוני לפני שלב 3.

$$\pi_{\text{sid, sname, bid}}\left(\sigma_{\text{rating = 10 and date < 1/1/11}}\left(\text{Sailors} \bowtie \text{Reserves}\right)\right)$$



- **שלב שני - התאמת אלגוריתמים:** לייד כל אופרטור אלגברי נכתוב את האלגוריתם שאנו רוצים להשתמש בו לחישוב השאילתה.
  מוסכמה: אם אנו משתמשים באלגוריתם שמצריך יחס חיתוני ופנימי, אזי היחס השמאלי הוא היחס החיצוני.

- **הגדרה -** *pipeslined*: אם לא כתבנו אלגוריתם לייד פעולות הבחירה וההטלה, אזי ההנחה היא כי המידע עובר אליהן ישירות שורה שורה מבלי לעבור בדיסק.

- **שלב שלישי - מיקום פעולות הבחירה:** אם נחליט שאנו משנים את מיקום פעולות הבחירה, אזי אם יש לנו אינדקס נרשום *index scan* אחרת נכתוב *full table*.
  הערה: פעולות אלו הן *pipeslined*.

- **כמה תכניות שונות יש עבור אותה שאילתה:** עבור צירוף על שתי טבלאות, ובהנחה שביצענו את שלב שלש - פעולות הבחירה מתרחשות בהתחלה. אם יש לנו אינדקס על כל היחסים שאנו צריכים להטיל עליהם יכולות להיות לנו 18 דרכים שונות.

- **כיצד נבחר את התכנית הטובה ביותר:**

1: נמצא את השיטה הזולה ביותר לחישוב פעולות הבחירה - $full\ table \setminus index$.

2: נמצא את אלגוריתם ה $join$ הטוב ביותר עבורנו.

3: נניח דחיפה מקסימלית של פעולות בחירה והטלה - לא נתעסק בתכנונים ללא דחיפה.

- **נגדיר:** עבור ביטוי E, $Read(E)$ - עלות הקריאה של הביטוי מהדיסק. $B(E)$ - כמות הבלוקים בביטוי. $T(E)$ - כמות השורות בביטוי.

- **חישוב עלות צירוף על ביטויים:** כשחישבנו עלות שאילתה עם ארבעת אלגוריתמי הצירוף, הנוסחאות כללו את הגודל של הטבלה + קריאת הטבלה ($B(S)$).

אך כאשר אנו משתמשים בחישוב העלות על ביטוי, יכול להיות שגודל הביטוי וקריאת הביטוי שונים, ולכן $B(S)$ לא תהיה העלות בפועל. לכן נרצה להבדיל בין גודל תוצאת הביטוי לבין עלות הקריאה.

- **עלות האלגוריתם** BNL**:** אם היחס החיצוני הוא R -

$$read(E_R) + read(E_S) \cdot \left\lceil \frac{B(E_R)}{M-2} \right\rceil$$

- **עלות האלגוריתם** INL**:** אם היחס החיצוני הוא R -

$$read(E_R) + T(E_R) \cdot cost\ of\ select$$

- **עלות האלגוריתם** $HASH\ JOIN$**:** אם היחס החיצוני הוא R ומתקיים התנאי על הדליים -

$$read(E_R) + read(E_s) + 2B(E_R) + 2B(E_S)$$

- **עלות האלגוריתם** $SM\ join$**:** אם היחס החיצוני הוא R ומתקיים התנאי על הדליים -

$$read(E_R) + read(E_s) + 2B(E_R) + 2B(E_S)$$

אם התנאי על הדליים לא מתקיים:

$$read(E_R) + read(E_s) + 4B(E_R) + 4B(E_S)$$

8 שבוע 8:

8.1 הרצאה 8 - תיאוריות תכנון ונירמול:

- **יתרונות של נרמול:** יותר מהיר לבצע פעולות עדכון והכנסה. הטבלאות יהיו קטנות יותר ונכונות המידע צישמר.
- **חסרונות של נרמול:** חישוב שאילתות ידרשו יותר פעולות צירוף.
- **כלל האצבע:** תמיד ננרמל תחילה את מסדבהנתונים, אם נראה שאחרי הנרמול מתרחשוב בעיות נוריד את הנרמול.
- **טיפוס של עמודה:** ניתן להגדיר את ה $type$ של עמודה להיות מערך או קבוצה וכך להכניס מערך של נתונים בערך של העמודה.
- **הגדרה - צורה נורמלית ראשונה** $(Firs\ normal\ fotm)$**:** נאמר שטבלה היא מ $Firs\ normal\ fotm$ אם כל אטריביוט מכיל ערכים אטומים (לא קבוצות או מערכים) ואין עמודות שמכילות את אותה קבוצת ערכים (לדוגמה אין שלש עמודות כשכל עמודה מייצגת לקוח אחר של העסק אלא טבלת לקוחות).
- **חסרונות של כפל מידע:** אם נשמור מידע שמשותף לכמה שורות בכל אחת מהשורות ה יכול להוביל למספר בעיות. אחת הבעיות היא במקרה של שינוי נצטרך לעדכן בכל המקומות. בנוסף זה תופס לנו זיכרון לשווא.
- **הגדרה - אנומליות עדכון:** כשנרצה לעדכן ערכים כפולים נצטרך לשים לב שאנו מעדכנים את כולם, מקרה של טעות נקרא אנומליות עדכון.
- **הגדרה - אנומליות הוספה:** אם נרצה להוסיף מקום חדש לטבלה אך הוספתו תלויה בהוספת ערך אחר (לדוגמה הוספת מספר טלפון למשרד חדש אך עדיין אין שם עובד, כך לא נוכל להוסיף את המספר ללא פרטי העובד).
- **הגדרה - אנומליות מחיקה:** אם נרצה למחוק ערך מהטבלה אך ערך זה תלוי בערך אחר (לדוגמה הסרת ערך של עובד שפוטר אך הסרתו תגרום להסרת מספר הטלפון של המשרד).

8.1.1 תלות פונקציונלית:

- **הגדרה - תלות פונקציונלית:** בהינתן סכמה של יחס $R(A_1...A_n)$ וקבוצות אטרביוטים X, Y שמוכלות בקבוצת האטרביוטים של R, בנוסף נתונות מספר שורות מהיחס R שנסמנן ב r.
  נאמר שמקיימת תלות פונקציונלית $X \Rightarrow R$ אם לכל שתי שורות $s, t \in r$ אם s, t שוות ב X אזי הן שוות גם ב Y.
  דוגמה - כל שתי שורות עם שם משרד (אטרביוט X) דומה, יכילו גם מספר טלפון (אטרביוט Y) דומה.
- **הערה:** מתכנן הסכמה הוא זה שאמור להגדיר אלו תלויות צריכות להתקיים בסכמה. והוא יתן לנו קבוצה F עם התלויות הטרוויאליות בסכמה.
- **הערה:** אם כל השורות ביחס השמאלי שונות, אזי יש תלות פונקציונלית באופן ריק (גם אם השורות ביחס הימני דומות).

- **הגדרה - הסקת תלות:** עבור קבוצת תלויות פונקצינליות F, נאמר שהתלות $X \Rightarrow Y$ נובעת ממנה אם עבור כל מופע r של R אם התלויות ב F מתקיימות אזי גם $X \Rightarrow Y$ מתקיים. ונאמר ש $X \Rightarrow Y$ נובע מ F.

- **טרנזטביות:** אם $X \Rightarrow Y$ וגם $Y \Rightarrow Z$ אזי מתקיים $X \Rightarrow Z$.

- **הגדרה - תלות טרוויאלית:** אם $Y \subseteq X$ אזי נגיד ש $X \Rightarrow Y$ היא תלות טרוויאלית.
  למעשה: אם כל אטרביות שמופיע בצד ימין מופיע גם מצד שמאל אזי התלות טרוויאלית.

- **כיצד נוכיח שתלות לא מתקיימת:** נייצר דוגמה נגדית שעומדת בכל התלויות הנתונות, אך סותרת את התלות שרצינו להוכיח.

- **הגדרה - סגור** $(closer)$**:** בהינתן קבוצת תלויות F וקבוצת אטרבריוטים X, נגדיר את הסגור של X - X_F^+ היא קבוצת כל האטרבריוטים A כך ש $X \Rightarrow A$.

- **אלגוריתם לחישוב הסגור של** X**:**

Algorithm that Computes X_F^+

Closure(X, F)

V:=X

While there is a Y → Z in F such that

1. Y is contained in V and
2. Z is not contained in V

do add Z to V

Return V

- **למה:** $X \Rightarrow Y$ נובע מ F, אמ"מ $Y \subseteq X$.

- **למה:** $X \Rightarrow Y$ נובע מ F, אמ"מ $Y \subseteq Closer\,(X, F)$.
  כדי לבדוק אם $X \Rightarrow Y$ נובע מ F, נחשב את הסגור של X ונבדוק אם Y נמצא בתוצאה.

- **הגדרה - מפתח על:** קבוצת אטרביוטים X ב R היא **מפתח על** אם $X^+ = R$ (כל האטרביוטים ב R מופיעים בסגור של X).

- **הגדרה - מפתח:** נאמר ש X **מפתח** ב R אם $X^+ = R$. וגם, לכל $Y \subset X$ מתקיים $Y^+ \subset R$ (X קבוצה מינימלית, הורדת אטרביוט מהקבוצה תגרום לה לא להיות מפתח על).

- **הערה:** יכולות להיות מספר קבוצות בגדלים שונים ושתיהן יהיו מפתח כי כל אחת מינימלית.

- **אלגוריתם למציאת מפתח:**

```
Minimize(X, F):
for each A∈X
        if A∈{X-A}+
                then X = X – {A}
Return X

FindKey(R,F):
Return Minimize(R,F)
```

9 שבוע 9:

9.1 הרצאה 9:

9.1.1 אלגוריתם למציאת כל המפתחות:

- **הרעיון:** נמצא מפתח אחד - k, ואחכ נסתכל על כל התלויות שגוררות את המפתח $k \Rightarrow$. אנו יוצאים מנקודת הנחה שאם k הוא מפתח, אזי כל אטרביוט שיגרור אותו גם הוא יהיה מפתח.
נבדוק עבור כל אטרביוט שגורר את k אם ניתן להחליף את k באטרביוט שגורר אותו.

- **אלגוריתם למציאת כל המפתחות:**
נתחיל ממפתח מסויים, ונסתכל על כל הגרירות, כל גרירה שמכילה את המפתח בצד ימין תתווסף לתור ונבדוק אותה

Finding all Keys

```
All Keys(R,F):
K := FindKey(R,F)
KeyQueue := {K}
Keys := {K}
While KeyQueue.isNotEmpty()
    K := KeyQueue.dequeue()
    Foreach X→Y ∈ F for which Y ∩ K is not empty do
        S:=K-{Y} ∪ X          //S is a superkey!
        If S does not contain any J ∈ Keys then
            S' := Minimize (S,F) //S' is a new key
            Add S' to Keys and to KeyQueue
Return Keys
```

לפי הכלל של החיתוך.

- **הערה:** כמות המפתחות יכולה להיות אקספוננציאלית ולכן האלגוריתם לא יהיה פולינומיאלי.
- **זמן הריצה של האלגוריתם:** פולינומיאלי בפלט ובקלט.
- **הערה:** אטרביוט שלא מופיע כלל בצד ימין של התלויות הפונקציונליות, יש לו סיכוי גבוה יותר להימצא במפתח.

9.1.2 הצורות הנורמליות $BCNF,\ 3NF$:

- **צורות נורמליות:** צורות שבהן הטבלאות שלנו צריכות להיבנות בכדי שנהיה בטוחים שהן בנויות בצורה הגיונית. באה למנוע את הבעיה שיש כפילויות בערכים שבטבלאות.
- **כלל אצבע:** אם צד שמאל של התלויות הפונקציונליות הנתונות לנו הוא מפתח - לא תהיה יתירות, אך אם הוא לא מפתח יכולה להיות יתירות.
- **הצורה הנורמלית** $BCNF$**:** נאמר ש R הוא בצורה נורמלית $BCNF$ אם לכל תלות $X \Rightarrow Y$ **שנובעת מ** F, מתקיימת אחת משתי הדרישות הבאות:

  1: $Y \subseteq X$ - התלות טרוויאלית, צד ימין מוכל בצד שמאל.

2: X הוא **מפתח על** של R.

כלומר: נרצה שרק מפתחות יקבעו לנו את התלויות הפונקציונליות.

- **הגדרה שקולה ל** $BCNF$**:** נאמר ש R הוא בצורה נורמלית $BCNF$ אמ"מ לכל תלות $X \Rightarrow Y$ **ב** F, מתקיימת אחת משתי הדרישות הבאות:

  1: $Y \subseteq X$ - התלות טרוויאלית, צד ימין מוכל בצד שמאל.

  2: X הוא **מפתח על** של R.

- **עלות הבדיקה:** אם יחס הוא ב $BCNF$ היא פולינומאלית.

- **הצורה הנורמלית** $3NF$**:** נאמר ש R הוא בצורה נורמלית $3NF$ אם לכל תלות $X \Rightarrow Y$ **שנובעת מ** F, מתקיימת אחת משתי הדרישות הבאות:

  1: X הוא מפתח על של R.

  2: לכל אטרביוט $A \in Y$ (בצד ימין) מתקיים כי - $A \in X$ (שהוא כבר היה בצד שמאל), או שהוא מופיע במפתח.

  כלומר: נרצה שהאטרביוטים יהיו מפתחות או יופיעו במפתח.

- **הגדרה שקולה ל** $3NF$: נאמר ש R הוא בצורה נורמלית $3NF$ אם לכל תלות $X \Rightarrow Y$ **ב** F, מתקיימת אחת משתי הדרישות הבאות:

  1: X הוא מפתח על של R.

  2: לכל אטרביוט $A \in Y$ (בצד ימין) מתקיים כי - $A \in X$ (שהוא כבר היה בצד שמאל), או שהוא מופיע במפתח.

- **עלות הבדיקה:** אם יחס הוא ב $3NF$ היא בעיית $NP - hard$.

- **הערה:** אם יחס הוא ב $BCNF$ הוא גם ב $3NF$, אך ההפך לא מתקיים.

- **הערה:** אם קבוצת התלויות הפונקציונליות ריקה היחס שייך ל $BCNF$ כי התנאים מתקיימים באופן ריק.

- **טענה:** כל יחס עם שני אטרביוטים הוא תמיד ב $BCNF$.

9.1.3 פירוקים:

- **מתי נשתמש בפירוק:** אם היחס שלנו לא ב $3NF$ ולא ב $BCNF$ נרצה להשתמש בפירוק.

- **הרעיון:** נרצה לפרק את היחס לתתי יחסים כדי להקטין את כמות היתירות של המידע שיש לנו בטבלה.

- **הגדרה - פירוק:** פירוק של R היא קבוצה של יחסים $R_1...R_n$ כך שסך האטרביוטים ב $R_1...R_n$ הוא אותו דבר כמו ב R.

- **מה נרצה שהפירוק יקיים:**

  1: הפירוק יהיה **ללא אובדן** - נוכל לשחזר את היחס המקורי ע"י צירוף טבעי בין תתי הטבלאות.

  2: נרצה שיתקיים **שימור תלויות** - כל אחת מהתלויות תתקיים על כל תת טבלה לאחר הפירוק, וגם על הטבלה הגדולה לאחר צירוף.

  3: נרצה שתתי הטבלאות יקיימו את $3NF$ או $BCNF$.

- **הערה:** עבור יחס r יתקיים תמיד כי $r \subseteq \pi_{R_1} r \bowtie \ldots \bowtie \pi_{R_n} r$. כלומר r יהיה מוכל בצירוף ההטלות על r.

- **הגדרה - פירוק ללא אובדן לשני תתי יחסים:** נאמר שפירוק היחס R לשני יחסים R_1, R_2 הוא ללא אובדן ביחס ל F אם לפחות אחד משני התנאים הבאים מתקיים:
  1: $R_1 \subseteq (R_1 \cap R_2)^+$.
  2: $R_2 \subseteq (R_1 \cap R_2)^+$.

- **הגדרה - פירוק ללא אובדן ליותר משני תתי יחסים:**
עבור יחס R עם k אטרביוטים ופירוק ל n תתי יחסים, נבדוק אם הפירוק ללא אובדן באופן הבא:
נייצר טבלה:

```
CreateTable(R=(A1,...,Ak),R1,...,Rn):
T := new Table[n,k]
for i = 1 to n
    for j = 1 to k
        if Aj∈Ri
            then T[i,j] = aj
            else T[i,j] = bij
return T
```

לאחר מכן נבדוק אם יש בטבלה סתירה לתלויות, אם כן **נעדכן** את הטבלה:

```
ChaseTable(T,F):
While there are rows t, s in T and X→Y in F such that
            t[X]=s[X] and t[Y]≠s[Y]
    for each Ai in Y do
        if t[Ai]=ai then replace s[Ai] with ai
        else if s[Ai]=ai then replace t[Ai] with ai
        else replace t[Ai] with s[Ai]
```

כדי להכריע אם הפירוק ללא אובדן **נבדוק** אם יש שורה שיש בה רק ערכי a, אם כן - הפירוק ללא אובדן:

```
TestDecomposition(R,R1,...,Rn,F):
T = CreateTable(R,R1,...,Rn)
ChaseTable(T,F)
If T contains a row with only "a" values
    return "lossless"
else return "not lossless"
```

- **הערה:** אם אין אף שורה שמכילה רק a־ים, אזי היא דוגמה נגדית לכך שהפירוק הוא עם אובדן. כלומר - לאחר $join$ נוצרה שורה שלא הייתה בטבלה המקורית.

9.2 תרגול 9 - צורות נורמליות:

- **היררכיה של צורות נורמליות:** $BCNF \subseteq 3NF \subseteq NOT-3NF$.
- **טענה:** יהי יחס R עם תלויות פונקציונליות F, וכל מפתח ב F הוא אטרביוט בודד. אזי R הוא ב $BCNF$ אמ"מ R הוא ב $3NF$.

10 שבוע 10:

10.1 הרצאה 10 - פירוקים:

- **הגדרה - הטלה** $(projection)$**:** הטלה של F על תת סכמה R_i, היא אוסף התלויות הפונקציונליות שנובעות מ F ורלוונטיות לאטרביוטים ב R_i (כל התלויות שרק אטרביוטים מ R_1 משתתפים בהן). ובאופן פורמלי:

$$\mathrm{F_{R_i} = \{V \to W \text{ s.t. } V, W \subseteq R_i \text{ and } V \to W \text{ follows from } F\}}$$

הערה: הקבוצה הזו יכולה להגיע לגודל אקספוננציאלי במספר האטרביוטים.

- **הגדרה - שימור תלויות בפירוק:** נרצה לוודא שהתלויות מתקיימות בתתי הסכמות מבלי שנצטרך לצרפן כדי לוודא שימור תלויות.
באופן פורמלי: נאמר שפירוק משמר תלויות, אם לכל תלות $X \Rightarrow Y$, התלות נובעת מ F אמ"מ היא נובעת מאיחוד ההטלות F_{R_i}.
- **אלגוריתם פולינומאלי לחישוב שימור תלויות:**

```
IsDependencyPreserving(R,R1,..,Rn,F):
For each X→Y in F do:
        Z:= X
        repeat
                Z' := Z
                for i = 1 to n do
                        Z := Z ∪ ((Z ∩ Ri)+ ∩ Ri)
        until Z = Z'
        if Y is not contained in Z then return "NO"
Return "YES"
```

- **הערה:** במהלך ריצת האלגוריתם ניתן להשתמש באותה הסכמה כמה פעמים כדי להגדיל את החיתוך.
- **הערה:** אין צורך לבדוק שימור תלויות עבור תלויות של אטרביוטים שמופיעים כולם באותה הטבלה.
- **בדיקת הצורה הנורמלית של תתי היחסים:** בכדי להגדיר מה הצורה הנורמלית של כל תת יחס R_i נרצה לדעת מה התלויות מעל R_i. נעשה זאת ע"י חישוב חלקי של ההטלה של F על R_iבעזרת האלגוריתם האקספונציאלי הבא:

Computing a set of Dependencies Equivalent to F_{R_i}

```
ComputeDependenciesInProjection (R,Ri,F):
G := ∅
For each X ⊆ Ri do:
        Add the dependency X→(X+ ∩ Ri)  to G
```

לאחר מכן נעבור על כל אחת מהתלויות שהגדרנו בתת היחס ונבדוק איזו צורה נורמלית היא מקיימת.

- **הגדרה - כיסוי מינימלי:** קבוצה שמכילה תלויות פונקציונליות כך שאף אחת מהן לא נובעת מאחרת.
  פורמלית: כיסוי מינימלי לקבוצת תלויות פונקציונליות F היא קבוצת תלויות פונקציונליות G כך ש:
  צד ימין קטן - לכל תלות ב G יש רק אטרביוט אחד בצד ימין.
  הקבוצה G צריכה להיות שקולה ל F- כל תלות ב F נובעת מ G.
  G צריכה להיות מינימלית - לא ניתן להוריד ממנה תלויות מיותרות.
- **הערה:** יכול להיות שיהיו לנו כמה כיסויים מינימלים שונים. כתלות בסדר המעבר על התלויות.
- **כלל:** אם יש לנו תלות $X \Rightarrow Y$:
  אם האטרביוט X לא מופיע בצד שמאל של אף תלות אחרת, אזי התלות בהכרח נמצאת בכיסוי המינימלי.
  אם האטרביוט Y לא מופיע בצד ימין של אף תלות אחרת, אזי התלות בהכרח נמצאת בכיסוי המינימלי.

- **אלגוריתם פולינומיאלי למציאת כיסוי מינימלי:**

Finding a Minimal Cover

```
ComputeMinimalCover(F):
G := ∅
for each X→Y in F
    for each A in Y
        add X→ A to G
for each X → A in G
    for each B in X
        if A ∈ (X-B)⁺ then  remove B from X → A
for each X → A in G
        if X → A follows from the other dependencies
        then remove X → A
```

- **אלגוריתם למציאת פירוק ל** $3NF$**:**

```
Find3NFDecomposition(R, F):
G := ComputeMinimalCover(F)
for each X→A in G
    add the schema XA
If no schema created contains a key, add a key as a schema
Remove schemas that are contained in other schemas
```

- **אלגוריתם למציאת פירוק ל** $BCNF$**:** לא נוכל להבטיח שימור תלויות.

```
FindBCNFDecomposition(R, F):
If R is in BCNF
    then return R
    else let X→Y be a BCNF violation
        R1 = X⁺
        R2 = X ∪ (R-X⁺)
        return FindBNCFDecomposition(R1,FR1) ∪
                FindBNCFDecomposition(R2,FR2)
```

הערה: האלגוריתם לא פולינומיאלי, אך קיים אלגוריתם פולינומיאלי.

11 שבוע 11:

11.1 הרצאה 11 - ניהול טרנזקציות:

- **הרעיון:** נרצה לאפשר הרצת מספר תכניות במקביל על אותו מסד נתונים, כך שלר יתרחשו יתנגשויות. בהרצאה זו נתמקד בבעיות שיכולות להיווצר מהרצת כמה תכניות במקביל.

- **בעיות שיכולות להיווצר בהרצת תכנית:**
חישוב חלקי של התכנית: התכנית רצה, אך בגלל מקרה בלתי צפוי היא תעצר באמצע.
באג בתכנית: התכנית לא מסיימת את הפעולה הנדרשת עקב באג.
הרצת תכניות במקביל: מספר תכניות רצות במקביל וסותרות אחת את השניה.
תקלת מערכת: המערכת נפלה ולכן התכנית לא בוצעה עד הסוף.

- **תכונות** $ACID$**: תכונות שנרצה שיתקיימו במערכת.**
אטומיות - $Atomicity$**:** כל תכנית תתבצע בצורה אטומית - לא יהיה חישוב חלקי, אם הן לא יסתיימו כולן בהצלחה - יתרחש $rollback$.
עקביות - $Consistency$**:** המתכנת צריך לדאוג שהקוד יהיה קונסיסטנטי ולא יהיו בו באגים.
בידוד - $Isolation$**:** כל תכנית תחשוב שהיא רצה לבד ותכניות אחרות לא יפריעו לה.
עמידות - $Durability$**:** השינויים ישמרו גם במקרה של נפילת המערכת, ואם בוצע $commit$ המידע ישמר למסד הנתונים.

- **מנהל הטרנזקציות:** החלק במערכת שאחראי על תזמון והרצת תכניות במקביל. אחראי על $Atomicity, Isolation$.

11.1.1 טרנזקציות:

- **הגדרה - טרנזקציה:** כשיש לנו אוסף של פעולות שמבחינה לוגית צריכות להתבצע יחד, נשתמש במנגנון הטרנזקציה. כך במקרה שאחת הפעולות לא יכולה להתבצע כולן לא יתבצעו. (למשל העברת כסף מחשבון אחד לאחר, צריך להוריד מאחד ולהוסיף לאחר, נרצה שזה יתרחש כפעולה אטומית).

- **כתיבת טרנזקציה:** נתחיל עם הפקודה $BEGIN\ TRANSACTION$, לאחר מכן נכתוב פקודות עדכון\הכנסה...
לבסוף יש לנו כמה אפשרויות: $commit$- שמירת השינויים. $rollback$ - אם יש ערך לא תקין נוכל לבטל את השינויים (במקרה של חריגה, המערכת תבצע פקודה זו לבד). $END\ TRANSACTION$ - כמו הפקודה $commit$.

11.1.2 בעיות מקביליות - $ConcurrencyProblems$:

- **כתיבה מלוכלכת:** מתרחשת כאשר טרנזקציה A כותבת מידע חדש על מידע שנכתב ע"י טרנזקציה B שעדיין לא ביצעה $commit$.

- **קריאה מלוכלכת:** מתרחשת כאשר טרנזקציה A קוראת ערך שנכתב ע"י טרנזקציה B שעדיין לא ביצעה $commit$.

- **קריאה שאינה ניתנת לשחזור -** $nonreteatable\ read$**:** מתרחשת כאשר טרנזקציה קוראת שוב ערך שהיא כבר קראה, אך הוא שונה בקריאה השניה למרות שהיא לא שינתה אותו. מצב זה בעייתי לתכנית כי היא לא תדע איך להתקדם.

- $Phantom\ read$**:** מתרחשת כאשר טרנזקציה מקבלת שתי קבוצות ערכים שונות עבור אותה שאילתה (לדוגמה התווספו שורות חדשות). הבעיה הזו דומה ל $nonreteatable\ read$ ומשנה את הנחות העבודה של התכנית.

- **אנומליה סדרתית -** $Serialization\ anomaly$**:** מתרחשת כאשר קיים שוני בין הרצת הפקודת במקביל, לבין הרצתן כסדרת פעולות.

11.1.3 כיצד נדאג ל $Isolation$:

- **הפתרון:** נשתמש במנגנון בקרת מקביליות שידאג לכך שהטרנזקציות לא יראו אחת את השניה אך ירוצו במקביל.

- **שיטה ראשונה -** $Single\ version$**:** ערך של כל שורה יישמר פעם אחת ויעודכן.
  שיטה שניה - $multi\ version$**:** נשמור הרבה העתקים לכל שורה, וכל טרנזקציה תסתכל על עותק אחר, בנוסף נדאג שכולם יתעכנו כמו שצריך.

- **דרגות בידוד של טרנזקציות:** קיימות 4 רמות בידוד שונות שמגדירות עד כמה כל טרנזקציה מבודדת מהאחרות.

$$read\ uncommited < read\ commited < repeatable\ read < serializable.$$

  מערכות שונות, יממשו את הרמות הללו בצורות שונות.

- **הערה:** כשנדבר על בידוד נתמקדת ברמת ה $serializable$.
  הערה: המצב הדיפולטיבי בכתיבת תכנית ב $Postgres$ היא $read\ commited$.

- **כיצד נגדיר דרגת בידוד:** לאחר פקודת התחלת טרנזקציה, נוסיף שורה של $ISOLATION\ LEVEL$.
  בנוסף ניתן להגדיר פקודת $SET\ TRANSACTION$ ולאחריה לכתוב את רמת הבידוד, וזה ישרה את הרמה על כל הטרנזקציות שיבואו אח"כ.

11.1.4 הגדרות:

- **הגדרה - טרנזקציה:** ביצוע של תכנית אחת במערכת מסד הנתונים, נסתכל עליה כאושף של קריאת וכתיבת אובייקטים במסד הנתונים.

- **הנחות על טרנזקציות:**
  1 - טרנזקציות לא מתקשרות אחת עם השניה, אלא יכולות לראות את התוצאות של הכתיבות אחת של השניה.
  2 - נניח כי מסד הנתוני הוא קבוצה קבועה של אובייקטים בת"ל.
  3 - נניח כי יש רק העתק אחד לכל אובייקט במסד הנתונים (יותר מאוחר ניפתר מהנחה זו).

- **הגדרה - תזמון** $(Schedule)$**:** תזמון קבוצת טרנזקציות $T_1...T_n$, הוא סידור הטרנזקציות כך שיהיו קונסיסטנטיות.

- **הגדרה - תזמון מלא:** אם כל הטרנזקציות עושות $commit$ או $anbort$ בסוף הפעולה, נאמר כי התזמון הוא מלא.

- **הגדרה - תזמון סדרתי:** אם הפעולות מתרחשות אחת אחרי השניה ולא אחת תוך כדי השניה, התזמון ייקרא סדרתי.

- **הנחת הריצה הסדרתית:** כל תזמון סדרתי מעל בסיס נתונים קונסיסטנטי, תשאיר אותו קונסיסטנטי.

- **הגדרה - תזמון בר סידור** $(Serializable)$**:** נאמר כי תזמון של טרנזקציות שמבצעות $commit$ הוא בר סידור, אם האפקט שלו על מסד הנתונים דומה לאפקט של תזמון סדרתי (גם אם הוא אינו סדרתי).
אם חלק מהטרנזקציות מבצעות $commit$ **וחלק מבצעות** $abort$**:** נאמר כי התזמון הוא בר סידור, אם האפקט שלו על מסד הנתונים זהה לריצה סדרתית של הטרנזקציות שביצעו $commit$.

- **מה הופך תזמון ללא בר סידור - קונפליקטים:**
1 - אם הטרנזקציות קוראות או כותבות קבוצות זרות של אובייקטים - אין בעיה.
2 - אם יש טרנזקציות שרק קוראת אך לא כותבת - אין בעיה.
3 - תתרחש בעיה כאשר טרנזקציה אחת כותבת את אובייקט A,וטרנזקציה אחרת קוראת או דורסת אותו.

- **סוגי קונפליקטים:**
WR: כאשר טרנזקציה אחת כותבת אובייקט A, וטרנזקציה אחרת קוראת אותו (לא בהכרח תתרחש בעיה).
RW: כאשר טרנזקציה אחת קוראת אובייקט A, וטרנזקציה אחרת כותבת אותו (לא בהכרח תתרחש בעיה).
RR: כאשר טרנזקציה אחת כותבת אובייקט A, וטרנזקציה אחרת דורסת אותו (לא בהכרח תתרחש בעיה).

- **הגדרה - תזמון בר התאוששות** $(recoverable)$**:** נאמר כי תזמון הוא בר התאוששות, אם כל הטרנזקציות מבצעות $commit$ רק לאחר שכל הטרנזקציות שביצעו שינויים לפניהן עשו גם כן $commit$.

- **הגדרה -** $cascading\ aborts$**:** פעולות $abort$ שמתרחשות בשרשרת. ברגע שיש קריאה מלוכלכת יכול להתבצע לנו פעולות $abort$ אחת אחרי השניה. משום שאם הטרנזקציה הראשונה מבטלת את פעולתה, כל הטרנזקציות שקראו את מה שהיא כתבה יצטרכו לבטל את פעולתן גם כן.
הגדרה - $avoids\ cascading\ aborts$**:** נאמר שתזמון נמנע מביטולים בשרשרת, אם טרנזקציות קוראות רק שינויים של טרנזקציות שביצעו $commit$.

11.1.5 מה נרצה שיקרה:התאוששות:

- **השאלות שנשאל:**
1 - נרצה שתהיה לנו יכולת לזהות האם תזמון הוא בר סידור.
2 - איך מסד הנתונים יכול לדאוג שהתזמונים שיווצרו יהיו ברי סידור.

- **הגדרה - קונפליקט בין פעולות:** שתי פעולות המקיימות את התנאים הבאים - מתבצעות על אותו אובייקט. מבוצעות ע"י שתי טרנזקציות שונות. לפחות אחד מהן היא פעולת כתיבה. יוגדרו כקונפליקט (משום שהן עלולות לגרום לבעיה).

- **הגדרה - תזמונים שקולי קונפליקט:** נאמר כי שני תזמונים מעל אותן טרנזקציות הם שקולי קונפליקט, אם כל זוג פעולות הנמצאות בקונפליקט, מופיעות באותו הסדר בשני התזמונים.

- **טענה:** לתזמונים המוגדרים כשקולי קונפליקט, יהיה את אותו אפקט סופי על מסד הנתונים.

- **הגדרה - תזמון בר סידור קונפליקטים** $(conflict\ serializable)$**:** נאמר כי תזמון הוא בר סידור קונפליקטים, אם הוא שקול קונפליקטים לתזמון סדרתי.

- **בדיקה האם תזמון הינו בר סידור קונפליקטים:** בהינתן תזמון S, ניצור גרף קדימויות מכוון של התזמון. בגרף יש קדקוד לכל אחת מהטרנזקציות, וצלעות בין כל שתי טרנזקציות T_i, T_j כך שאם יש שתי פעולות קונפליקט בתזמון שמערבות את שתי הטרנזקציות, וגם T_i היא הראשונה שמתבצעת.
  נאמר כי S הוא בר תזמון אם אין מעגל בגרף.
  כדי למצוא את התזמון הסדרתי אליו הוא שקול, נפעיל אלגוריתם מיון טופולוגי.

11.2 תרגול 11 - תיאוריות עיצוב וטרנזקציות:

- **טריק ל** $BCNF$**:** אם יש לנו n אטרביוטים, אזי תלות פונקציונלית מקבוצה בגודל $n-1$ בהכרח תקיים $BCNF$ משום ש: או שהיא תגרור את כל האטרביוטים ותהיה מפתח, או שהיא תגרור את עצמה ותהיה טרוויאלית.

11.2.1 טרנזקציות:

- **הרעיון:** נרצה שכמה מתמשים יוכלו לגשת אל מסד הנתונים ולקרוא או לכתוב ממנו מבלי שיהיו התנגשויות. לכן קיימת מערכת - מנהל הטרנזקציות, שמקבלת את כל הפעולות ומחליטה מתי הן יתבצעו.

- **הגדרה - טרנזקציה:**
  מבחינת המשתמש: רצף פעולות של משתמש בודד שמבחינתו הן צריכות להתבצע ברצף אטומי.
  מבחינת מסד הנתונים: מנהל טרנזקציות שצריך לנהל את הפעולות כך שלא תתרחש התנגשות.

- **הגדרה - תזמון:** רצף פעולות שהגיעו מטרנזקציות שונות, והן נוגעות באותם אובייקטים.

- **פעולות של טרנזקציות:** עבור אובייקט A, טרנזקציה יכולה לקרוא את A, לכתוב את A, לשמור את הפעולות שבוצעו - $commit$, ולבטל את הפעולות שבוצעו - $abort$.

- **הגדרה - קריאה או כתיבה מלוכלכת:** מצב שנרצה להימנע ממנו. כתיבות או קריאות שמתבצעות לפני שהטרנזקציה הקודמת ביצעה פעולת $commit$. נרצה להימנע מזה משום שאם הטרנזקציה הקודמת תבצע פעולת $abort$ אנו לא נדע מה נשמר.

- **הגדרה - קריאה שאינה ניתנת לשחזור -** $nonreteatable\ read$**:** מצב שנרצה להימנע ממנו. טרנזקציה שקוראת אובייקט מסויים, ובפעם השניה שהירא ניגשת לקרוא אותו הוא השתנה ע"י טרנזקציה אחרת.

- **הגדרה - בר התאוששות -** $recoverable$**:** אם טרנזקציה מסויימת מבצעת פעולת $abort$ זה לא יגרום לבעיה. ע"י כך שהטרנזקציה הבאה תעשה גם פעולת $abort$. (המצב הזה לא מתקיים תמיד והוא תלוי בסדר הפעולות).

- **הגדרה - $cascading\ aborts$:** פעולות $abort$ שמתרחשות בשרשרת. ברגע שיש קריאה מלוכלכת יכול להתבצע לנו פעולות $abort$ אחת אחרי השניה. משום שאם הטרנזקציה הראשונה מבטלת את פעולתה, כל הטרנזקציות שקראו את מה שהיא כתבה יצטרכו לבטל את פעולתן גם כן.

12 שבוע 12:

12.1 הרצאה 12 - פרוטוקולים:

- **מהם פרוטוקולים:** נרצה שהתזמונים שלנו יהיו ברי סידור, לכן מערכת מסד הנתונים תצטרך לעקוב אחרי פרוטוקולים מסויימים כדי שתעמוד בתקנים.
  קיימים שני סוגי פרוטורקהתאוששות:ולים: פרוטוקול מבוסס נעילה, ופרוטוקול חותמת זמן פשוט.

- **מנעולים:** לכל אובייקט במסד הנתונים יש מנעול שהוא שייך אליו. קיימים שני סוגי מנעולים.
  מנעול משותף - $Shared\ lock$: מנעול המשמש לקריאה
  מנעול אקסלוסיבי - $Exlusive\ lock$: מנעול המשמש לקריאה (מי שיש לו מנעול זה יכול גם לקרוא את האובייקט).

- **מנעול אקסלוסיבי:** טרנזקציה יכולה הלהחזיק במנעול אקסלוסיבי על אובייקט A, רק אם אין לאף טרנזקציה אחרת אף מנעול על האובייקט.
  מנעול משותף: כמה טרנזקציות יכולות להחזיק בו.

- **שימוש בפרוטוקול נעילה:** בכל פעם שטרנזקציה תרצה לגשת לאובייקט היא תצטרך לבקש את המנעול. מנעול משותף במקרה של קריאה, ומנעול אקסלוסיבי במקרה של כתיבה. אם המנעול תפוס, הטרנזקציה תחכה שהוא יתפנה.
  ניהול המנעולים יתבצע ע"י מנהל המנעולים.

12.1.1 פרוטוקול מבוסס מנעולים - $2PL\ Protocol$:

- **קריאה מאובייקט:** טרנזקציה שרוצה לקרוא מאובייקט תצטרך לבקש **מנעול משותף** על האובייקט.

- **כתיבה לאובייקט:** טרנזקציה שרוצה לכתוב תצטרך לבקש **מנעול אקסלוסיבי** על האובייקט.
  הערה: אם טרנזקציה רוצה לקרוא ואחכ לכתוב, היא יכולה לבקש מראש מנעול אקסלוסיבי.

- **שלבי הגדילה והכיווץ:** טרנזקציה יכולה לבקש כמה מנעולים שהיא רוצה - שלב הגדילה. אך ברגע שטרנזקציה משחררת מנעול כלשהו, היא לא יכולה לבקש מנעולים נוספים - שלב הכיווץ.

- **מה קורה כשהמנעול תפוס:** כשהמנעול תפוס מנהל המנעולים יכול לבצע אחד משני דברים: או להכניס את הטרנזקציה תור של טרנזקציות שמחכות. או לבצע פעולת $abort$ על הטרנזקציה, ולאחר מכן הרצתה באופן אוטומטי שוב עד שתצליח.

- **הגדרה - תזמון ניתן להשגה ע"י $2PL$:** נאמר כי תזמון ניתן להשגה $2PL$, אם הנעילות והשחרורים של המנעולים מתבצעות בסדר שבו הן כתובות בתזמון, ואינן סותרות את הכללים.

- **נגדיר:** $S(A)$ - מנעול משותף על A. $X(A)$ - מנעול אקסלוסיבי על A. $U(A)$ - שחרור מנעול על A.
- **משפט:** כל תזמון שניתן להשגה ע"י $2PL$, הוא בר סידור קונפליקטים (אין מעגל בגרף הקדימויות).
- **טענה:** תזמון שניתן להשגה ע"י $2PL$, **לא בהכרח** מקיים בר התאוששות.

12.1.2 פרוטוקול $Strict\ 2PL$:

- $Strict\ 2PL$: פרוטוקול מחמיר יותר, שדואג לכך שכל תזמון שמקיים את $2PL$ יהיה גם בר התאוששות.
הגדרה: נאמר שתזמון הוא $Strict$ אם כל ערך שנכתב ע"י טרנזקציה T, לא נדרס ע"י טרנזקציה אחרת עד ש T מבצעת פעלת $commit$ או $abort$.
- **כיצד נגדיר** $Strict\ 2PL$: נחליף את התנאי השלישי - טרנזקציה לא יכולה לבקש מנעול לאחר שהתחילה לשחרר מנעולים, בתנאי הבא - טרנזקציה משחררת את המנעולים שלה **רק בזמן הסיום** - פעולת $commit/abort$.
- **משפט:** תזמון המושג ע"י $Strict\ 2PL$, אזי בהכרח הוא בר סידור קונפליקטים, בר התאוששות, ונמנע מ $cascading\ aborts$.

12.1.3 $Deadlock$:

- **הגדרה** - $Deadlock$: מצב שיכול לקרות כאשאנו משתמשים בפרוטוקול $2PL$,מתרחש כששתי טרנזקציות או יותר מחכות אחת לפעולה של השניה (שחרור מנעול), כך שהן תקועות ולא יכולות להתקדם.
- **הפתרון:** קיימות שתי גישות לטיפול בבעיה. **מניעה** $prevention$ - נדאג שאף פעם לא נגיע למצב של $deadlock$. **גילוי** $Detection$ - נאפשר מצב של $deadlock$ אך נטפל בו כשהוא מתרחש.
- **מניעה** $prevention$: ניתן לטרנזקציה זמן התחלתי - מתי היא מתחילה לפעול, שמגדיר את העדיפות שלה.
שיטת $wait-die$: נעדיף את הטרנזקציה המוקדמת. אם הטרנזקציה שמבקשת את המנעול התחילה לפני הטרנזקציה שמחזיקה את המנעול - הטרנזקציה המבקשת תחכה. אחרת - נעשה לטרנזקציה המבקשת $abort$, ונאתחל אותה מחדש עם הזמן ההתחלתי הראשון.

שיטת $wound\ wait$: גם כאן נעדיף את הטרנזקציה המוקדמת. אם הטרנזקציה שמבקשת את המנעול התחילה לפני הטרנזקציה שמחזיקה את המנעול - נבטל את הטרנזקציה שמחזיקה את המנעול, ונפעיל אותה מחדש עם הזמן ההתחלתי הראשון. אחרת - הטרנזקציה המבקשת תחכה.

- $waits\ for\ graph$: גרף שעוזר בהתמודדות עם $deadlock$. עבור כל טרנזקציה נייצר קדקוד, ונייצר צלע בין $T_i \Rightarrow T_j$ אם T_i מחכה ל T_jשישחרר מנעול. מעגל בגרף מציין קיום מצב של $dedlock$.
שיטות המניעה משתמשות בגרף זה כדי לבדוק אם מתרחש $deadlock$.
- **שיטות לבחירת טרנזקציה לביטול:** כאשר מתרחש מצב של $deadlock$ מנהל הטרנזקציות יצטרך לבחור את הטרנזקציה שתתבטל בכדי שנצא ממצב $deadlock$. יש כמה שיטות:
1 - הטרנזקציה שיש לה הכי קצת מנעולים.

2 - הטרנזקציה שעשתה הכי פחות עבודה.

3 - הטרנזקציה החדשה ביותר.

12.1.4 פרוטוקול חותמת זמן - $Simple\ Timestamp$:

- **הרעיון:** נרצה תזמון ששקול קונפליקטים לתזמון סדרתי שמסדר את הטרנזקציות לפי זמן ההתחלה שלהן.

- **השיטה:** נגדיר זמן התחלתי לכל טרנזקציה $TS(T)$. אם חותמת הזמן של T_i קטנה מחותמת הזמן של T_k, אזי נרצה להיות שקולים לתזמון בו T_i הופעלה לפני T_k.
בפרט, אם שתיהן כותבות לאותו אובייקט נצפה לכך שהערך יהיה הערך ש T_kכתבה.

- **הגדרות נוספות:** נגדיר לכל אובייקט חותמת זמן קריאה - $RTS(A)$, וחותמת זמן כתיבה - $WTS(A)$. הערך יהיה הכתיבה או הקריאה של הטרנזקציה האחרונה.

- **ניהול הגישות לאובייקטים:** כשטרנזקציה T תרצה לגשת לאובייקט A, היא תבדוק את חותמת הזמן שלו.
אם חותמת הזמן של האובייקט מאוחרת יותר מזמן הטרנזקציה: (הטרנזקציה לא צריכה להיחשף אליו), - נבצע $abort$ לטרנזקציה ונפעיל אותה מחדש עם זמן TS חדש.
אחרת - נפצל למקרים של קריאה וכתיבה:
1 אם T רוצה לקרוא את A: נאפשר לטרנזקציה לקרוא את A, נעדכן את חותמת הזמן של A להיות - $RTS(\mathrm{A}) := \max(RTS(\mathrm{A}), \mathrm{TS}(\mathrm{T_i}))$. בנוסף, נעשה העתק לוקאלי של A, כדי שישמר למקרה ש T תרצה לקרוא אותו שוב.

2 אם T רוצה לכתוב את A: נחלק למקרים:
אם $TS(T) < RTS(A)$ **או** $TS(T) < WTS(A)$**:** כלומר - הפעם האחרונה שקראו או כתבו את A הייתה אחרי ש T נוצרה. נבצע $abort$ ל T ונאתחל אותה עם זמן חדש.
אחרת: נאפשר כתיבה, נעדכן $TS(T) < WTS(A)$ ונשמור העתק לוקאלי של A.

- **כלל הכתיבה של תומס:** בדומה לפרוטוקול המקורי, שני הכללים הראשונים נשארים אותו הדבר, אך השוני הוא בכתיבה לאובייקט אחרי שכבר כתבו אליו.
אם T רוצה לכתוב את A ו $TS(T) < WTS(A)$**:** כלומר - הפעם האחרונה שכתבו את A הייתה אחרי ש T נוצרה. לא נבצע $abort$ אלא נבצע את הכתיבה, אך רק **על העתק** לוקאלי וא נשנה את מסד הנתונים.

- **משפט:** הפרוטוקול ללא כלל הכתיבה של תומס, שקול קונפלקטים לתזמון סדרתי המסודר לפי זמן.
עם כלל הכתיבה של תומס: אין שקילות קונפליקטים. אך יש שקילות לריצה סדרתית. (ההשפעה על מסד הנתונים שקולה בין שתי הגרסאות).

12.1.5 פרוטוקול $Multi\ version(MMVC)$:

- זה הפרוטוקול שמערכת $postgres$ משתמשת בו.

- **הרעיון:** לכל טרנזקציה תהיה תמונת מצב של בסיס הנתונים (עותק) בהתאם לזמן בו היא התחילה לרוץ. כל העדכונים נעשים לתמונת המצב, ברגע של ביצוע $commit$ הפעולה תתבצע רק אם היא יכולה להסתיים בלי לפגוע באף פעולה אחרת.

- **יתרונות:** כל טרנזקציה יכולה לרוץ מבלי לחכות לאחרות.

- **טבלאות בבסיס נתונים:** הטבלאות מכילות עמודות נסתרות, כגון $xmin, xmax$ שמייצות ID של טרנזקציות שייצרו את השורה או מחקו את השורה בהתאמה.
למעשה אף שורה לא נמחקת מהטבלה אלא רק הערכים $xmin, xmax$ משתנים.

- **הפקודה** $vacum$**:** ניתן להשתמש בה, בנוסף היא רצה מידיי פעם מאחורי הקלעים, היא מוחקת את השורות שלא רלוונטיות לאף טרנזקציה.

- **הפקודה** **SELECT *, Xmin, xmax FROM :** תחזיר את השורות $xmin, xmax$ בנוסף לשורות הטבלה.

- **העמודה** $ctid$**:** מייצגת את מיקום השורה בזכרון - בלוק ושורה.

13 שבוע 13:

13.1 הרצאה 13 - התאוששות:

- **התאוששות:** אמרנו כי מסד הנתונים צריך לקיים כמה תכונות, שתיים מתוכן הן אטומיות - שהפעולות יתרחשו בצורה אטומים, ואם לא אז הן יתבטלו ויתרחש $rollback$. עמידות - הפעולות יישמרו לאר ביצוע $commit$.
התאוששות באה להבטיח ששתי פעולות אלו אכן יקרו, שתהיה עמידות, וכיצד נבצע $rollback$.

- **גיבוי ושחזור:** נרצה שיתבצע גיבוי בכל שלב ולכן נצטרך לבצע מספר פעולות שימרו את המידע. בנוסף נרצה שתהיה לנו היכולת לשחזר את המידע שנשמר במקרה שתהיה נפילה של מסד הנתונים או ביצוע פעולת $rollback$.

- **עם איזה סוגי נפילות המערכת צריכה להתמודד:** יש שלש סוגי נפילות - ברמת המערכת, הטרנזקציה והדיסק.
נפילת טרנזקציה: אם התרחשה שגיאה לוגית, או פעולה שהטרנזקציה לא יכולה לבצע בגלל סתירה לאילוצים, הטרנזקציה תרצה לבצע $rolllbak$. בנוסף היא יכולה לבצע $abort$ כדי למנוע דדלוק.
נפילת מערכת: אם יש באג במסד הנתונים, או המחשב שעליונ\ נמצא מסד הנתונים נופל. נשים לב כי נפילות מערכת הן בלתי צפוייות.
נפילת הדיסק: אם הדיסק מקולקל ולא עובד לא ניתן לשחזר את מסד הנתונים. הפתרון הוא לשמור גיבוי של מסד הנתונים על כמה דיסקים.

- **ניהול הזיכרון המרכזי -** $buffer\ pool$**:** כשטרנזקציה מבצעת שינוי על אובייקט, מנהל הזיכרון המרכזי צריך להחליט על פוליסה וכיצד לעדכן את הדיסק במקרה של שינויים.

- **פוליסת** $Steal$**:** מסד הנתונים יכול לקחת את הבלוקים מהזכרון ולכתוב אותם לדיסק גם אם הטרנזקציה עדיין לא ביצעה $commit$.

- **פוליסת** $no\ Steal$**:** רק ערכים שבוצעו עליהם $commit$ יכתבו לדיסק.
עם פוליסה זו קל יותר לבצע $rollback$. אך לא בטוח שתמיד יהיה לנו מספיק מקום בזכרון המרכזי כדי לשמור את כל השינויים עד ביצוע $commit$.

- **פוליסת** $Force$**:** ברגע שהתבצעה פעולת $commit$ המערכת תכריח את הערכים שהשתנו להיכתב לדיסק.
פוליסה זו מקלה על התאוששות כי השינויים נמצאים כבר על הדיסק. אך היא איטית יותר כי אנו מחוייבים לכתוב לדיסק בכל פעם שמתרחשת פעולת $commit$.

- **פוליסת** $no\ Force$**:** לא נכתוב לדיסק כל שינוי ישירות לאחר ביצוע פעול $commit$.

- **הפוליסה הנפוצה:** במערכות מסדי נתונים היא $Steal + no\ Force$, משום שבמסדי נתונים גדולים הן יעילות יותר. אך נצטרך לדאוג לגיבוי ושחזור.

- log**:** תיעוד שנמצא בזכרון המרכזי, בו אנו כותבים בו את הפעולות שביצענו על השורות של מסד הנתונים, נכתוב מתי טרנזקציה מתחילה, מתי היא כותבת, ופעולות $abotr, commit$. אין צורך לכתוב ללוג קריאות של טרנזקציות.

- **שיטת** $Write\ Ahead\ Logging(WAL)$**:** שיטה שמערכת בסיס הנתונים משתמשת בה לשחזור המידע.
ניהול הבאפר נעשה בפוליסה של $Steal + no\ Force$. המערכת תייצר log של כל השינויים שנעשו על מסד הנתונים, ותכתוב אותו לדיסק. נשים לב כי המערכת צריכה לכתוב ל log את השינויים ולכתוב את ה log לדיסק **לפני** שהשינויים מתבצעים בפועל על מסד הנתונים. כשמתרחשת פעולת $commit$ המערכת תכתוב את ה log לדיסק.
במקרה של נפילת המערכת נוכל לשחזר את המידע מה log.

- **מתי המערכת תכתובת את ה** log **לדיסק:** יש 4 מקרים שונים בהם נכתוב את ה log לדיסק.
1: במקרה של ביצוע פקודת $commit$, כך נוכל לשחזר את המידע במקרה של נפילה.
2: אם אנו רוצים לבצע $Steal$, ורוצים לכתוב לדיסק בלוק שנעשו בו שינויים. נכתוב תחילה את ה log לדיסק ואחכ נכתוב את הבלוק.
3: אם המקום שהקצנו ל log מתמלא, נעביר אותו לדיסק.
4: מידי פען כשמסד הנתונים לא בשימוש נעביר את ה log לדיסק.

- **מה נשמר ב** log**:** בכל כניסה ללוג נשמור
1: $Log\ Sequence\ Number(LSN)$ שהוא מעין מספר מזהה לפעולה.
2: בנוסף, כשנכתוב את הלוג לדיסק נשמור בזכרון המרכזי את מספר הכניסה האחרונה שבוצעה ללוג - $flushedLSN$.
3: המערכת תשמור גם $prevLSN$ - שמציין את מספר שורה הקודמת בה הטרנזקציה ביצעה שינוי (במקרה שזה השינוי הראשון נשמור בשדה זה $null$).
4: המערכת תשמור גם CLR - מידע על $undo$ הכולל את: מספר השורה שעליה ביצענו $undo$, מהו הערך ששונה, מהי פעולת ה $undo$ הבאה, ומהי השורה הקודמת עליה ביצענו $undo$.

- **מה קורה כשטרנזקציה עושה** $commit$**:** המערכת כותבת את כל השורות שהטרנזקציה ביצעה עליהן שינויים (מה שמופיע בלוג) לדיסק.
המערכת תשמור עבור כל טרנזקציה את ה $last\ LSN$ - השינוי האחרון של הטרנזציה, כך היא תעקוב האם שורה

זו נכתבה כבר לדיסק או שצריך לכתוב אותה. כך אם $last\ LSN > flushedLSN$ (השורה האחרונה בלוג שבה הטרנזקציה ביצעה שינוי, גדולה מהשורה האחרונה שנכתבה), אם כן - נכתוב לדיסק את הלוג עד ה $last\ LSN$, ונעדכן את $flushedLSN = last\ LSN$.

- **מה קורה כשנרצה לכתוב דף מלוכלך לדיסק:** בגלל פוליסת $Steal$, נרצה לכתוב דפים שהשתנו אך טרם בוצעה עליהם פעולת $commit$. המערכת תכתוב לדיסק את כל החלקים של הלוג עד המקום בו ביצענו שינו בדף. המערכת תשמור $pageLSN$ שמסמן את השורה האחרונה בלוג בה בוצע השינוי בדף.
בבואנו לבצע $Stel$ המערכת תבדוק אם $pageLSN > flushedLSN$ (האם השורה האחרונה שבה בוצע השינוי, גדולה מהשורה האחרונה שנכתבה לדיסק), אם כן - נכתוב לדיסק את הלוג עד ה $pageLSN$, ונעדכן את $pageLSN = last\ LSN$.
בנוסף המערכת תשמור את $recLSN$ שתסמן את השורה הראשונה שהפכה את הדף למלוכלך.

- **האלגוריתם $ARIES$ להתאוששות:** נשתמש במדיניות של $Steal + no\ Force$. נשחזר את ההסטוריה בזמן של ה $redo$, כלומר - כשאנו רוצים לבצע התאוששות מנפילה נעבור על כל ההסטוריה לפי ה log כדי להגיע למצב שהיה לפני ההתרסקות. לאחר מכן נעשה $undo$ לכל הטרנזקציות שעדיין לא ביצעו $commit$, ונכתוב ל log את כל הפעולות של ה $undo$.
1 נשמור טבלה של הטרנזקציות הפועלות ATT: עבור כל טרנזקציה נשמור את - ID, סטטוס הטרנזקציה (בתהליך, מבצעת $commit$, בוטלה וכעת מתבצע שחזור), ו $last\ LSN$.
2 נשמור טבלת DPT: טבלה שמייצגת את כל הדפים המלוכלכים, עבור כל דף נשמור את ה - $page\ number, recLSN, pageLSN$.

- **מה קורה כשטרנזקציה מבצעת $abort$:** במקרה שלא הייתה נפילת מערכת נצטרך לבצע $undo$. נעדכן שפעולת ה $abort$ הצליחה, ונוסיף את השינוי ל log.
לאחר מכן נצטרך לדעת איזה שינויים הטרנזקציה ביצעה. נסתכל על ה $last\ LSN$ של הטרנזקציה כך נדע מהו השינוי האחרון שהטרנזקציה ביצעה, בנוסף נוכל לעקוב אחר כל השינויים שהטרנזקציה ביצעה עם ה $prevLSN$.
עבור כל $undo$ שבוצע נכתוב ללוג CLR, כדי שנדע שבוצע שינוי בשורה זו במקרה שהמערכת תיפול תוך ביצוע $undo$.

13.1.1 שחזור מערכת:

- **התאוששות מנפעילת המערכת כולה:** עד עכשיו התעסקנו בנפילת טרנזקציה בודדת, כעת נראה כיצד נתאושש מנ0ילת המערכת.
אנליזה: ננסה לשחזר את טבלאות ה ATT, DPT למצב שהן היו לסני הנפילה.
$Redo$: חזור על כל ההיסטוריה החל מ $recLSN$ הקטן ביותר בטבלת הדפים המלוכלים.
$Undo$: נבצע $rollback$ על כל טרנזקציה שלא ביצעה $commit$ לפני הנפילה.

- $Checkpoint$: נקודה בה אנו יודעים שכל המידע נכתב לדיסק, נרצה לשמור את הנקודה הזו כדי לשחזר החל ממקום מסויים ולא מה log הראשון.
כדי לשמור את הנקודה הזו - לא נאפשר לטרנזקציות חדשות להתחיל, ונחכה שכל הטרנזקציות שרצות יסיימו. לאחר מכן נכתוב לדיסק את כל ה log, נכתוב את כל הדפים המלוכלים, נוסיף ללוג $Checkpoint$, ונכתוב את הלוג לדיסק.

- **שלב האנליזה:** נלך אל נקודת ה $Checkpoint$ האחרונה, אם אין לנו כזו נלך ללוג הראשון. נמצא את נקודת הזמן ממנה נעשה $redo$, ונשחזר את ה DPT - טבלת דפים מלוכלים. למעשה לא נוכל לשחזר את כל ה DPT באופן יעיל, אך נמצא קבוצה המכילה את DPT.
בנוסף נצטרך לשחזר את ATT - טבלת הטרנזקציות הפעילות. גם כאן ייתכן שנשחזר קבוצה גדולה יותר מהקבוצה בפועל.

- **שלב ה** $Redo$**:** לאחר ששחזרנו את הטבלאות, נלך לערך $pageLSN$ הקטן ביותר (הדף המלוכלך הראשון), בטבלה DPT, ומשם נתחיל את ה $Redo$.
נעשה זאת כך - נבדוק עבור כל דף מלוכלך בטבלה אם $pageLSN < curLSN$, אם כן - עדכון הדף לא נכון ולכן נצטרך לעשות לו $Redo$ לפי מה שמופיע בלוג.

- **שלב ה** $Undo$**:** לאחר ששחזרנו את הטבלה ATT, נעשה $abort$ לכל הטרנזקציות שלא סיימו באותו האופן שמבצעים $rollback$. נתחיל מהטרנזקציה האחרונה בלוג.

14 שבוע 14 :

14.1 הרצאה 14 - $NoSQL$:

- $NoSQL$**:** מסד נתונים שאינו רלציוני, ולא מבוסס על טבלאות. לרוב לא תהיה תמיכה בפעולת צירוף, אלא הפעולה מתבצע ע"י שפה עילית אחרת שעובדת מעל המסד ותומכת בפעולה. בנוסף הם אינם מנורמלים ויש ייתירות של מידע, כמו כן אין קונסיסטנטיות.

- **מעבר בין מסדי נתונים:** במעבר בין DB שאינן SQL, נצטרך לכתוב את כל המסד מחדש.

- $scale\ up$**:** פעולה בה אנו משדרגים את המחשב, ומעבירים את מסד הנתונים למחשב עם כח חישוב גדול יותר.

- $scale\ out$**:** כשאנו רוצים לפצל את מסד הנתונים שלנו לכמה מחשבים שונים מפאת גודלו.

- **חסרונות של מסד נתונים רלציוני (**SQL**):** מסד הנתונים שומר את כל הדאטה בטבלאות, במידה ויהיה לנו דאטה כדוגמת גרף או מסמכים יכול להיות שנעדיף לשמור אותם בצורה אחרת ולא בטבלאות.
החיסרון השני הוא שמסדים רלציונים מתקשים עם פעולת $scale\ out$, בנוסף הם יקרים יותר.

- **מודלי נתונים:** המודל של מסד רלציוני הוא טבלה שטוחה - כך שכל המידע מאוחסן בשלש רמות: טבלה, שורה, אובייקט אטומי. למסדי נתוהים שאינם רלציונים יש מודלים אחרים.

- $schema-less$**:** בחלק ממסדי הנתונים שאינם רלציונים אנו לא צריכים לתת סכמה בבניית הטבלה ולהתחייב אליה לכל אורך הדרך. אלא אנו יכולים להוסיף ולהוריד מידע כרצונינו.
החסרון: מסד הנתונים אינו מסודר ולכן יכול להיות שנרצה מידע מסויים אך הוא כלל לא נמצא שם ואנו לא יכולים לעקוב אחרי זה.

14.1.1 $Data\ Models$:

- **המודל** $key - value\ store$**:** טבלת האש מבוזרת, כך שניתן לשמור אותו על מספר מחשבים שונים בו זמנית. יש מפתחות כך שכל מפתח מייצג את האטרביוט. ניתן לעדכן ערכים של מפתחות כך:

$$SET\ person : name\ "Ronel"$$

הפעולות הן: $SET, GET, DEL, INCR$. הפעולה $INCR$ היא אטומית (קריאה ועדכון יחד).

- **המודל** $Document\ Store$**:** כאן הערכים אינם אובייקטים כליים, אלא מסמכי $json$.

- **המודל** $Column\ Store$**:** בדומה ל SQL גם הוא מאוחסן במעין טבלאות. אך הוא מתאים לטבלאות שיש להם הרבה ערכי $null$. ניתן לחשוב על המידע כאילו הוא יושב בטבלת האש ואנו מחלצים מידע כך: $f : (Row, Col) \Rightarrow val$ (נותנים מססר שורה ועמודה ומקבלים ערך).
השפה מאוד דומה ל SQL אך אין פעולת צירוף, העדכונים מבוססים רק על המפתח הראשי, אין פעולת or, אין תמיכה בטרנזקציות ועוד.

- **המודל** $Graph\ Databases$**:** המודל הוא אוסף של קודקודים וצלעות, עם תוויות על הקודקודים ועל הצלעות. בנוסף ניתן לשים משקל על הצלעות.

14.1.2 מודל ביזור - $Disrtibution\ Model$:

- **מודל ביזור:** הינו מודל שמכריע איך המידע מפוזר בין מחשבים שונים, כיצד הוא מפולג, והאם הוא משוכפל. הפתרון הפשוט הוא להניח שאין ביזור.

- **חלוקה אנכית:** נפריד את הטבלאות בין מחשבים שונים.

- **חלוקה אופקית:** נחלק את כל הטבלאות בין כמה מחשבים, כלומר כל מחשב יכיל את כל הטבלאות אך רק אחוז מסויים מכל טבלה.
מסד נתונים המבוסס על חלוקה זו מוגדר כ $sharded$, ומסדי נתונים שאינם SQL לרוב מחולקים בצורה זו.

- **כיצד נחלק את הטבלאות -** $Sharding$**:** נרצה לדעת מהי הדרך היעילה ביותר לחלק את הטבלאות בין מחשבים.
מיקום גיאוגרפי: נרצה לאחסן את המידע הכי קרוב למיקום הגאגרבי בו הוא בשימוש.
חלוקת עומס: נחלק פריטים פופולארים בין מחשבים שונים.
אגריגציה של נתונים: פריטי מידע שניגשים אליהם יחד בד"כ נשים באותו המחשב, כך נוכל לחסוך בבקשות..

- **איך נשכפל את הנתונים:** נרצה שכפול כדי שנוכל לגשת אל המידע ביעילות.
שיטת $Master - slave$**:** מחשב אחד יוגדר כ $master$, וכל השאר יוגדרו כ $slave$. ניתן לקרוא מכולם, אך ניתן לכתוב רק אל המאסטר.
חסרונות: אם המאסטר נופל לא ניתן לעדכן את המידע - כל הביצים בסל אחד. חיסרון נוסף הוא חוסר עקביות, ייתכן שנעדכן ערך אצל המאסטר, אך הוא עדיין לא התעדכן אצל ה $slave$.

שיטת $peer\ to\ peer$**:** כל המחשבים יחזיקו העתקים של כל הנתונים, ניתן לכתוב ולקרוא מכולם.
חסרונות: בעיות עקביות ועדכון, בנוסף אם שני משתמשים יכתבו לאותו הערך למחשבים שונים אחד ידרוס את השני בעדכון.

- **כיצד נעשה** $Sharding$ **ושכפול:** נוכל להשתמש בפרדיגמה של $Master - slave$. כך ניתן לשים כל פריט בכמה מחשבים, אך רק אחד יוגדר כמאסטר עבור הפריט הזה.

- $Sharding$ **עם** $peer\ to\ peer$**:** חלק מהמידע יישמר על כל אחד מהמחשבים וניתן לקרוא ולכתוב לכולם, והם יעדכנו אחד את השני.

14.1.3 איך נדאג לעקביות:

- **הסיבות לחוסר עקביות:** משתמשים שונים שמעדכנים את אותם ערכים יכולים לגרום לחוסר עקביות. בנוסף העתקי מידע לא מעודכנים יכולים לגרום לחוסר עקביות.

- **נרצה שיתקיימו תכונות** CAP**:**
$consistency$ **- עקביות:** אם אנו מבקשים מידע - נקבל את המידע האחרון שעודכן או הודעת שגיאה.
$Availability$ **- נגישות:** כל בקשה צריכה להיענות ע"י המסד, (ערך ולא הודעת שגיאה), אך אין כל דרישה שהיא תהיה עדכנית או נכונה.
$Partition\ Tolerance$**:** גם אם הרשת מחולקת לכמה מחשבים וחלק מהמחשבים אינם זמינים (נפלו), עדיין המערכת תמשיך לעבוד.

- **משפט** CAP**:** אף מערכת אינה יכולה לתת בו זמנית את שלשה התכונות יחד. לכן נצטרך לוותר על אחת מהתכונות הבאות - $consistency, Availability$.

- **מסד נתונים התומך ב** $Partition\ Tolerance$ **ו** $consistency$**:** לא תהיה לנו תמיכה ב $Availability$, כלומר לפעמים לא נקבל תשובה קונקרטית. נעדיף שחוסר זה יקרה בכתיבה, רק כשכל המחשבים במערכת עודכנו בשינויים העידכון ייעקה, אחרת - תתרחש חוסר $Availability$.

- **מסד נתונים התומך ב** $Partition\ Tolerance$ **ו** $Availability$**:** כל משתמש יקבל תמיד מידע, אך מכיוון שאין $consistency$ הם יכולים לקבל תשובות שונות לאותן השאלות.

פרוטוקול הסכמה מבוזר:

- **מוטיבציה:** לפעמים כשטרנזקציה תרצה לשנות ערך, היא לא תוכל לעשות זאת בכל המחשבים, וזה יפגע בעקביות. לכן נרצה פרוטוקול שיגדיר לנו באיזה מצבים טרנקציה תבצע $abort$ ולא תעדכן.

- **הפתרון הנאיבי:** המחשב שמנהל את הטרנזקציה ישלח הודעה לכל המחשבים שצריכים לעדכן את המידע, והם יחזירו לו הודעה כשהמידע עודכן.
החסרון: יכול להיות שתיפול הרשת ולא כולם יעודכנו, או שאין אופציה לעדכן את הנתונים (לדוגמה העברת כסף מלקוח A ל B).

- **מה אנו רוצים:** נרצה שיתקיימו התכונות הבאות:
בטיחות - $Safety$: או שהשינויים יבוצעו בכל המחשבים או שאף אחד מהם לא יבוצע.
דרישה שניה היא $Liveness$: אם כל המחשבים יכולים לבצע את הששינוי - הטרנזקציה תצליח לבצע $commit$, אחרת - נרצה לדעת על התקלה כמה שיותר מוקדם.

- **פרוטוקול $2PC - 2\ phase\ commit$:** יהיה מחשב אחד שינהל את הטרנזקציות (TC) בין המחשבים, והעדכון יבוצע בשני שלבים. נדגים על העברת כסף בין שתי חשבונות.
1: TC יגיד ל Aשיתכונן להעביר את הכסף (שיבדוק האם החשבון זמין ויש בו כסף), ויעדכן את B שהוא עציד לקבל כסף.
2: אם שני המחשבים ענו שהם יכולים לבצע את הפעולה - נעביר את הכסף. ואח"כ יודיע למחשב שביקש, שהפעולה אכן בוצעה.
אחרת - אם לא חזרה תשובה תוך $time\ out$ זמן, הוא יודיע לכולם לבצע $abort$.
הערה: המחשבים יכולים לתקשר ביניהם ללא ה TC ולהסיק האם לבצע את ההעברה או לא.
מה קורה במצב של נפילה: נשתמש ב log כדי שיהי תיעוד במקרה של נפילה.

14.1.4 כיצד נבחר מסד נתונים:

- **בחירת סוג מסד שאינו SQL:** נצטרך לבדוק מבו מודל הנתונים, ביזור הנתונים, הבטחות העקביות שנרצה ועוד.

- **כיצד נבחר בין SQL ל $noSQL$:** בבואנו לבחור מסד נתונים נסתכל על חמשת התכונות ונחליט לפי מה שחשוב לנו.
בטיחות - $integrety$: אם נרצה מידע שאינו משתנה נעדיף את - SQL שהוא יוצר אמין.
$scalability$: אם נרצה פיזור של המידע בי כמה מחשבים ככל שהקאטה גדל נעדף את $noSQL$
$queering$**:** אם נרצה מסד שתומך בשאילתות רבות ביעילות נעדיף את SQL
$unstructed\ data$**:** אם המידע שלנו לא מאורגן בטבלאות ואין לו מבנה ברור נעדיף את $noSQL$.
עלות - $cost$: מבחינת העלות ועבור כמויות גדולות של דאטה, העלות יותר נמוכה כאשר אנו מבזרים מערכות $noSQL$.

- **שילוב מסדי נתונים:** חברות גדולות מחזיקות כמה סוגים של מסדי נתונים, ומידע שונה הם יחזיקו בסוגי מסדי נתונים שונים לפי הצורך.

- **מערכות $newSQL$:** מערכות חדשות שהן רלציוניות, וגם תומכות בסקלביליות.

15 חזרה למבחן:

15.1 דיאגרמות:

- **חץ רגיל:** מסמן כי הישות יכולה להשתתף **לכל היותר פעם אחת** ביחס.
נקרא את החץ כך: לכל ישות שהחץ יוצא ממנה יש לכל היותר ישות אחת שהחץ נכנס אליה.

- **חץ עגול:** מסמן כי הישות יכולה להשתתף **בדיוק פעם אחת** ביחס.
נקרא את החץ כך: לכל ישות שהחץ יוצא ממנה יש בדיוק ישות אחת שהחץ נכנס אליה.

- **כמות הזוגות בקשר:**

עבור שני יחסים בלי חצים כלל: כמות הזוגות בקשר תהיה בין 0 למכפלת גודל הישויות.

עבור חץ רגיל: כמות הזוגות בקשר תהיה בין 0 לגודל הישות שהחץ יוצא ממנה. אם יש n ישויות שמשתתפות ביחס - אזי כל חץ שנכנס לישות ה n יחשב כאילו הוא יוצא מ $n-1$ הישויות האחרות והגודל המקסימלי יהיה מכפלת $n-1$ הקבוצות.

עבור חץ רגיל דו"צ: כמות הזוגות בקשר תהיה בין 0 למינימום גודל שתי קבוצות הישויות. זה נכון גם למספר ישויות שמשתתפות בקשר וחץ נכנס רק לחלקן, אזי נתייחס לכל חץ בנפרד כך שכל חץ מגדיר קבוצה.

עבור חץ עגול: אותו הדבר רק בלי הטווח מ 0, אלא רק המקסימלי.

- **קשר** $many\ to\ many$: אם יש לנו שלש ישויות - A, B, C, ויש חץ שנכנס ל A וחץ שנכנס ל B. אזי - B, C ביחס עם A, וגם - A, C ביחס עם B.

- **קבוצת ישויות חלשה:** היא קבוצה שלא ניתן לזהות רק באמצעות המפתח שלה. נסמן באמצעות ריבוע כפול, אם היא נמצאת בקשר עם קבוצה אחרת, אזי הקשר גם יסומן במעויין כפול.

15.2 תרגום דיאגרמות ליחסים:

- **נתרגם את הדיאגרמה לטבלאות \ יחסים כך:**

טבלה \ יחס: כל ישות תקבל טבלה, כך שכל תכונה של ישות תהיה עמודה בטבלה.

קבוצת קשרים: תהפוך גם היא ליחס עם התכונות שלה, בנוסף היא תקבל את המפתחות של הישויות שאיתה ביחס.

קבוצת קשרים רקורסיבית: תקבל את המפתחות של כל האפשרויות. לדוגמה - $managerID, workerID$

תרגום של חצים רגילים: עבור כל ישות שנכנס אליה חץ - נוסיף לטבלת **היחס** את המפתחות של שאר הישויות (שהחץ יוצא מהן). **אם יש כמה חצים** ניתן לבחור אחת מבין כל קבוצות הישויות שהחץ יוצא מהן.

תרגום של חץ עגול בודד: בקשר בינארי - אין צורך להגדיר טבלה חדשה ליחס, אלא נכניס את תכונות היחס אל הטבלה של הישות ממנה יוצא החץ. **אם יש כמה ישויות** - ניצור טבלה גם לקשר, כמו בחץ רגיל.

מפתח של טבלה: באותו האופן של דיאגרמה - מזהה מינימלי ייחודי.

קבוצת ישויות חלשה: ניצור יחס כרגיל, אך נוסיף את המפתחות של הקבוצות איתן היא בקבוצת הקשרים החלשה.

ירושה: יש כמה אפשרויות - **1** - $E \backslash R$: ניצור טבלת יחס עבור כל ישות, בקבוצות היורשות נשים גם את המפתחות של המורישות. **2** - $object\ orientrd$: ניצור טבלה עבור כל קומבינציה אפשרית בין ביחסים. היחס המוריש יקבל גם טבלה לבד. **3** - **ערכי** $null$: ניצור יחס בודד עבור כל הישיות, ונשים ערכי $null$ אם אין ערך מתאים לתכונה הזאת בישות.

15.3 במודל הרלציוני:

- **איחוד:** נאחד את **השורות** של שני היחסים.

בחירה σ: יחזרו **השורות** שמקיימות את התנאיץ

הטלה π: יחזרו **העמודות** שהטלנו עליהן.

חיסור: נחסר את השורות בין היחסים. עבור $A-B$ יחזרו כל **השורות** של A שלא נמצאות ב B.

מכפלה קרטזית: כל הקומבינציות האפשרויות בין **עמודות ושורות**. עבור $A \times B$ מספר השורות והעמודות בטבלה החדשה יהיו $|A| \cdot |B|$.

חיתוך: נחזיר את החיתוך של **העמודות** של שני היחסים.

צירוף טבעי $\bowtie$**:** מכפלה קרטזית על היחסים רק בשורות שיש להן את אותם ערכים בשתי הטבאות.

חילוק: נפעיל את האופרטור רק אם עמודות היחס השני B מוכלות בראשון A. עבור $A \div B$ יחזרו **העמודות** של A שלא נמצאות ב B עם השורות שנמצאות בשתיהם.

- **טענות על שקילות:**

איחוד והטלה π**:** הטלה ואח"כ איחוד = איחוד ואח"כ הטלה.

חיתוך והטלה: חיתוך והטלה לאחר מכן $\neq$ להטלה וחיתוך לאחר מכן.

חיסור ובחירה σ**:** בחירה ואח"כ חיסור = לחיסור ואח"כ הפעלת פעולת בחירה.

חילוק והטלה: חילוק ואח"כ הטלה $\neq$ להטלה על היחס וחלוקה לאחר מכן, אלא גדול ממנו.

צירוף טבעי והטלה: הטלה על עמודות שונות של יחס וצירוף שלהם $\neq$ ליחס, אלא גדול ממנו.

- **כשנרצה להוכיח שקילות:** נראה כי יש שוויון בין שני המקרים.

כשנרצה לסתור שקילות: נראה דוגמה נגדית.

15.4 SQL:

15.4.1 תתי שאילתות:

- **תתי שאילתות ב** $WHERE$**:**

תוך תתי השאילתות בשדה $WHERE$ ניתן להכניס את התנאים הבאים: $IN, NOT\ IN, ANY, ALL, EXIST, NOT\ EXIST$.

- **תתי שאילתות בשדה** $FROM$**:** חייבות להכיל $alias$.

- **פונקציות אגריגציה:** לרוב, נשים אותן בתוך השדה $SELECT$, מקבלות סט של ערכים ומחזירות ערך בודד. לדוגמה - $count, sum, avg, min, max$ כל הפונקציות הללו יכולות לקבל גם ערך בודד ולחשב עבורו, וגם את הטבלה כולה.

- **השדה** $HAVING$**:** פועל כמו השדה $WHERE$, אך בניגוד אליו הוא פועל על קבוצות.

- **השדה** $GROUP\ BY$**:** יחזיר את פונקציית האגריגציה עבור קבוצות של נתונים בעמודה מסויימת.

- **הערה חשובה:** כל שדה ב $SELECT$ או ב $HAVING$ שלא נמצא בתוך פונקציית אגריגציה, חיים להיות בתוך $GROUP\ BY$.

15.4.2 טבלה וירטואלית - $VIEW$ וטבלה זמנית - $WITH$:

- **יצירת טבלה וירטואלית -** $VIEW$**:** כשנרצה ליצור טבלה זמנית שלא נשמרת בזכרון, נוכל לעשות זאת באמצעות $VIEW$ כך:

$$CREATE\ VIEW\ name\ AS\ QUERY$$

- **עדכון טבלת $VIEW$:** כשנרצה לעדכן (מחיקה, הוספה, שינוי) טבלת $view$ הטבלה שתתעדכן היא הטבלה שה $view$ מוגדרת מעליה.
- **יצירת טבלה זמנית - $WITH$:** הפקודה $WITH$ יוצרת לנו טבלה זמנית שנמחקת בסוף השימוש, נגדיר את הטבלה ואח"כ את השאילתות.

$$WITH\ tableName\ AS\ QUERY\ QUERY;$$

- **שאילתה רקורסיבית:** ניצור בעזרת הפקודה $WITH\ RECURSIVE$, לאחר מכן נשים חלק לא רקורסיבי, איחוד, ושאילתה רקורסיבית.

15.5 אינדקסים:

15.5.1 זכרון ועלויות:

- **עלות חיפוש שורה:** נצטרך לעבור על כל הבלוקים, לכן סה"כ N פעולות I/O.
  עלות הוספת שורה: אם יש אילוצים - העלות היא $N+1$. בלי אילוצים - העלות היא 2, קריאת וכתיבת הבלוק.
  מחיקת שורה: $N+1$ פעולות. קריאת הבלוק, מחיקת השורה וכתיבת ההבלוק.

15.5.2 $B+\ Tree$:

- **תכונות של העץ:** עבור עץ עם דרגת פיצול (מספר הבנים שיש לקודקוד) d מתקיים -
  לכל קודקוד יש לכל היותר d בנים.
  לכל קדקוד שאינו השורש יש לפחות $\lceil \frac{d}{2} \rceil$ בנים.
  לכל קדקוד שאינו עלה ויש לו k בנים, יש $k-1$ ערכי חיפוש.
- **בחירת דרגת הפיצול:** כדי לבחור את דרגת הפיצול של העץ נבצע את החישוב הבא:

$$d \leq \left\lfloor \frac{sizeOfBlock + sizeOfKey}{sizeOfPointer + sizeOfKey} \right\rfloor = d \cdot sizeOfPointer + (d-1) \cdot sizeOfKey \leq sizeOfBlock$$

- **עלות חיפוש בעץ $B+$:** עבור ערך מסויים, שווה לעומק העץ + עלות החיפוש על העלים.
- **נוסחה לחישוב גובה העץ:** אם מאוחבנים בעץ n ערכים אזי גובה העץ הוא $\left\lceil \log_{\lceil \frac{d}{2} \rceil}(n) \right\rceil$.
- $index\ unique\ scan$: מתייחס לסריקה עד עומק העץ ללא טיול על העלים. נשתמש בשיטה זו כאשר אנו צריכים לוודא אם קיים ערך המקיים תנאי כלשהו.

- $index\ Range\ scan$: ירידה בעץ + טיול על העלים בטווח מסויים (נוסחה למטה). נשתמש בשיטה זו כאשר מבקשים ממנו להחזיר תוצאות בטווח מסויים.

- $Table\ access\ by\ rowid$: נשתמש בשיטה זו כשאנו רוצים מידע נוסף שלא שמור באינדקס. לכן נגיע עד העלה הרצוי ולאחר מכן נחפש את המידע בטבלה. העלות היא ירידה בעץ + קריאת הבלוקים המתאימים מהטבלה.

- **נוסחה לחישוב מספר העלים בהם נמצא הפתרון:** אם התשובה נמצאת ב k שורות, אזי מספר הקודקודים המירבי שיכילו את המידע הוא - $\left\lceil \frac{k}{\left\lceil \frac{d}{2} \right\rceil - 1} \right\rceil$, (מספר השורות בהן נמצא המידע חלקי מספר הערכים לקדקוד).

15.6 ניתוח פעולת הצירוף:

- **סימונים:** $B(R)$ מסמן את מספר הבלוקים ביחס R. M מסמן את מספר הבלוקים ב RAM. $T(R)$ מסמן את מספר השורות ב R.

- **האלגוריתם** BNL: נבחר את היחס הקטן יותר להיות היחס החיצוני. נטען לתוך $M-2$ בלוקי הזכרון את היחס החיצוני, נטען בכל שלב בלוק אחד מהיחס הפנימי, ונכתוב את התוצאה לבלוק שנותר. אם נותרו עוד בלוקים מהיחס החיצוני נטען אותם בסיבוב הבא.
העלות: אם R הוא החיצוני -

$$B(R) + B(S)\left\lceil \frac{B(R)}{M-2} \right\rceil$$

- **האלגוריתם** $Index\ NL$: האלגוריתם מניח קיום אינדקס על היחס הפנימי, ומשתמש ב 4 בלוקי זכרון - 1 לכל טבלה, 1 לאינדקס ובלוק נוסף לפלט. נבחר את היחס הפנימי (היחס עם האינדקס), לאחר מכן עבור כל שורה ביחס החיצוני נעבור על האינדקס ונייבא את הבלוק של היחס הפנימי עם השורה המתאימה.
העלות: נשים לב כי העלות כוללת את החיפוש בעץ עם האינדקס, אם R הוא החיצוני -

$$B(R) + T(R) \cdot index\ search$$

- **האלגוריתם** $Hash\ join$: הרעיון הוא להפעיל פונקציית האש על השדה שאנו מצרפים לפיו, נגדיר $M-1$ דליים ועבור כל יחס נמיין את הערכים בשדה לדליים. נקרא דליים תואמים במקביל ונצרף בניהם, את התוצאה נכתוב לבלוק הנותר.
העלות: יצירת טבלת האש עבור יחס R לוקחת $2B(R)$ (קריאה, הפעלת הפונקציה וכתיבה לטבלה), ולאחר מכן קריאה של הטבלה עולה לנו עוד $B(R)$, לכן **סה"כ:** $3B(R) + 3B(S)$.
דרישות: נרצה שיהיה מקום בזכרון לכל הדליים של יחס אחד לפחות, לכן נדרוש שיתקיים אחד מהתנאים הבאים:

$$\left\lceil \frac{B(R)}{M-1} \right\rceil \leq M-2 \quad \text{or} \quad \left\lceil \frac{B(S)}{M-1} \right\rceil \leq M-2$$

- **מיון בעזרת אלגוריתם חיצוני:** לעיתים נרצה למיין את מסד הנתונים, נעשה זאת בשני שלבים עם אלגוריתם חיצוני דמוי $merge\ sort$. תחילה נמיין כל בלוק פפני עצמו ולאחר מכן נמזג אותם יחד.
דרישות: עבור מיון היחס R נדרוש שיתקיים - $\left\lceil \frac{B(R)}{M} \right\rceil < M$.
העלות: ראשית נקרא את כל הבלוקים נמיין ונכתוב אותם לכן $2B(R)$, לאחר מכן נעבור עליהם שוב כדי למיין ולכן העלות הכוללת היא $3B(R)$.

- **האלגוריתם** SMJ**:** הרעיון הוא למיין קודם כל כל אחת מהטבלאות, ולאחר מכן לבצע את הצירוף על הטבלאות הממוינות. האלגורית משתמש בשלשה בלוקים - 1 לכל טבלה ובלוק לפלט.
העלות: ראשית נמיין כל טבלה בעלות של $3B(R)$ אח"כ נכתוב אותה לדיסק, ולאחר מכן נקרא אותן שוב ונמיין.
סה"כ: $5B(R) + 5B(S)$
דרישות: נדרוש שיתקיימו שני התנאים הבאים

$$\left\lceil \frac{B(R)}{M} \right\rceil < M \quad and \quad \left\lceil \frac{B(S)}{M} \right\rceil < M$$

ניתן לממש את האלגוריתם ביעילות אם מתקיים התנאי הבא:

$$\left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil < M$$

ואז העלות היא: $3B(R) + 3B(S)$.

15.7 הערכת גודל הפלט:

- **סימונים:** $V(R, A)$ - מסמן את מספר הערכים השונים שיש לאטרביוט A ביחס R.

- **כלל האצבע:** כדי לדעת כמה שורות יהיו לנו בפלט נכפול את מספר השורות בטבלה $T(R)$, בהסתברות לקבל את הערך.

$$P(A = a) = \frac{1}{V(R, A)}, \quad P(A \leq x) = \begin{cases} \frac{x-y+1}{z-y+1} & if\ A \in [y, z] \\ \frac{1}{3} & else \end{cases}$$

- **עבור שאילתה** $A = "a"$**:** אם נניח התפלגות אחידה נחשב את מספר השורות בטבלה חלקי מספר הערכים השונים בשדה A. - $\frac{T(R)}{V(R,A)}$.

- **עבור שאילתה** $A \leq x$**:** אם יש לנו את טווח של ערכי $A \in [x, z]$, נחשב את האחוש של x מהטווח ע"י $\frac{x-y+1}{z-y+1}$ ונכפול במספר השורות $T(R)$ - סהכ $\frac{x-y+1}{z-y+1} \cdot T(R)$.

- אם אין לנו מידע על התפלגות הערכים - נניח כי ההתפלגות היא $\frac{1}{3}$.

- **עבור תנאי של** and**:** נחשב את $T(R) \cdot P(a) \cdot P(b)$. נחלק את מספר השורות בהסתברות. כאשר $P(B)$ יכול להיות $\frac{1}{V(R,B)}$ עבור התנאי $B = b$, או $\frac{1}{3}$ אם אין לנו מידע על טווח הערכים.

- **עבור תנאי** or**:** נכפול את מספר השורות בטבלה, ב
(שני התנאים לא מתקיימים)$1 - P$

$$\left(1 - \left(\underbrace{1 - \frac{1}{V(R,A)}}_{P(A \neq a)}\right) \cdot \left(\underbrace{1 - \frac{1}{V(R,B)}}_{P(B \neq b)}\right)\right)$$

- **עבור צירוף בתנאי** $R.A = S.A$**:** נכפול את מספר השורות בכל טבלה, בהסתברות למספר הערכים השווים (שיכולים להצטרף) בשתי הטבלאות

$$T(R) \cdot T(S) \cdot \frac{1}{\max\{V(R,A), V(S,A)\}}$$

- **עבור צירוף בתנאי ותנאי בחירה:** נעשה את אותו החישוב כמו מקודם, אך נכפול גם בהסתברות לתנאי הבחירה $P(B = b) = \frac{1}{V(R,B)}$.

- **עבור צירוף בשני תנאים** $R.A = S.A \ and \ R.B = S.B$**:** נכפול את מספר השורות בכל טבלה, בהסתברות למספר הערכים השווים (שיכולים להצטרף) בשתי הטבלאות עבור כל תנאי צירוף

$$T(R) \cdot T(S) \cdot \frac{1}{\max\{V(R,A), V(S,A)\}} \cdot \frac{1}{\max\{V(R,B), V(S,B)\}}$$

- **עבור הטלה:** ללא מחיקת כפילויות - $T(R)$. עם מחיקת כפילויות - $V(R,A)$.

15.8 נרמול מסד נתונים:

- **הגדרה - תלות פונקציונלית:** לדוגמה אם מתקיימת התלות $ID \Rightarrow name$, אזי כל השורות עם אותו ערך ID ייצטרכו להכיל בשדה $name$ את אותו השם (אך אם ה ID שונה אין בעיה שיהיה אותו השם - עדיין יש תלות).
אם כל השורות של האטרביוט השמאלי שונות אחת מהשניה - התלות מתקיימת באופן ריק (גם אם השרות של האטרביוט הימני זהות).
מתכנן הסכמה צריך לספק קבוצת תלויות F שצריכות להתקיים בסכמה.

- **הגדרה - הסקת תלות:** נוכל להסיק תלות חדשה מהקבוצה F, אם היא נגררת מהתלויות שב F.

- **הגדרה - סגור:** עבור קבוצת אטרביוטים בקבוצת התלויות F. הסגור של X מסומן כך X_F^+, ומוגדר להיות כל האטרביוטים שמתקבלים ע"י גרירות מהקבוצה X.
אלגוריתם לחישוב סגור: אם נרצה לחשב סגור של אטרביוט A- נתחיל מהוספת A לקבוצת הסגור, לאחר מכן נוסיף את כל האטרביוטים שנגררים ע"י A. לאחר מכן נעבור על כל האטרביוטים שבקבוצת הסגור, ונוסיף את האטרביוטים שנגררים על ידיהם.

- **למה לתכונת סגור:** הגרירה $X \Rightarrow Y$ נובעת מהקבוצה F אמ"מ Y מוכל בסגור של X.

- **הגדרה - מפתח על:** קבוצת אטרביוטים X, כך שכל האטרביוטים ב R נמצאים בסגור שלה.
אלגוריתם למציאת מפתח על: עבור כל תת קבוצה של אטרביוטים ב R, נמצא את הסגור של תת הקבוצה. אם הגענו לכל R אזי קבוצה זו היא מפתח על.

- **הגדרה - מפתח:** קבוצת אטרביוטים X שהיא מפתח על, בנוסף היא מינימלית - אם יוריד ממנה אטרביוט היא לא תהיה מפתח על.
הערה: יכולים להיות כמה מפתחות מינימלים בגדלים שונים.
אלגוריתם למציאת מפתח: נקח קבוצה שהיא מפתח על, ועבור כל אטרביוט בה נבדוק אם נוכל לקבל אותו גם אם הוא לא יהיה בקבוצה. אם כן - נוריד אותו.
אלגוריתם למציאת כל המפתחות: תחילה נמצא מפתח k כלשהו, לאחר מכן נסתכל על כל אטרביוט ב F שגורר אחד או יותר מהאטרביוטים של k, וננסה להחליף את האטרביוט ב k באטרביוט שגורר אותו. נוכל לעשות זאת במבנה של עץ.

15.8.1 הצורות הנורמליות:

- **צורה ראשונה -** $FNF - first\ normal\ form$**:** נאמר כי טבלה בצורה נורמלית ראשונה אם היא מכילה רק אטומים (לא רשימות) ואין שדות שחוזרים על עצמם ($client1, client2$).

- **הצורה** $BCNF$**:** נאמר כי R הוא בצורה הנורמלית $BCNF$ אם כל תלות ב F היא **טרוויאלית** או **מפתח על**.
הערה: אם F קבוצה ריקה, אזי היחס ב $BCNF$.
טענה: כל יחס בגודל 2 הוא ב $BCNF$.

- **הצורה** $3NF$**:** נאמר כי R הוא בצורה הנורמלית $3NF$ אם כל תלות ב F היא **1** - מפתח על, או **2** - שכל אטרביוט שמופיע במד ימין של תלות מופיע גם בצד שמאל שלה, או חלק **ממפתח**.

15.8.2 פירוקים:

- **הגדרה - פרוק ללא אובדן:** פירוק היחס Rלתתי יחסים $R_1..R_n$, כך שצירוף תתי היחסים ישחזר את היחס המקורי.
פורמלית: פירוק הוא ללא אובדן אם הסגור של החיתוך של תתי הטבלאות, מכיל את אחת מהטבלאות.
אלגוריתם לבדיקת אובדן:
1: ניצור טבלה שהעמודות מייצגות את האטרביוטים ב R, והשורות מייצגות את היחסים.
2: אם האטרביוט מופיע בתת היחס - נסמן את התא ב a_{col}. אם האטרביוט לא מופיע ביחס - נסמן את התא ב

$b_{row,col}$.

3: לאחר מכן נעדכן את הטבלה לפי התלויות - כל מקום שניתן להשלים אותו עם ערכי a נשלים אותו.

4: אם יש שורה שמכילה רק a-ים אזי הפירוק ללא אובדן.

- **הגדרה - הטלה של F על R_i:** קבוצת כל התלויות שנובעות מ F, אך מכילות רק אטרביוטים שנמצאים ב R_i.

כיצד נמצא את ההטלה: עבור כל קבוצת אטרביוטים ב R_i, נחשב את הסגור שלה ונקח רק את האטרביוטים שנמצאים ביחס - נוסיף את התלות הזאת לקבוצה.

- **הגדרה - פירוק משמר תלויות:** אם כל אחת מהתלויות שהיו על R יישמרו גם על תתי היחסים.

פורמלית: פירוק הוא משמר תלויות, אם כל תלות שנובעת מ F, נובעת גם מאיחוד ההטלות F_{R_i}.

אלגוריתם לבדיקת שימור תלויות:

1: עבור כל תלות $X \Rightarrow Y$ ב F, נגדיר $Z = X$.

2: נחשב עבור כל תת יחס - $Z := Z \cup \left((Z \cap R_i)^+ \cap R_i\right)$, כלומר - נעדכן את הקבוצה Z בכל שלב.

3: נבדוק אם $Y \in Z$, אם כן - הפירוק משמר את התלות, ונעבור לתלות הבאה. אחרת - הפירוק אינו משמר את התלות ולכן אינו משמר תלויות.

- **חישוב הצורה הנורמלית של תת יחס:** נצטרך לחשב את ההטלה - F_{R_i} ולבדוק איזו צורה נורמלית היא מקיימת, ע"י מעבר על כל התלויות בהטלה.

- **הגדרה - כיסוי מינימלי:** קבוצה תוגדר מינימלית אם כל תלות בה לא יכולה לנבוע מהאחרות.

פורמלית: כיסוי מינימלי של קבוצה F, מוגדר ע"י קבוצה G המקיימת: לכל תלות ב G יש רק אטרביוט 1 בצד ימין. הקבוצות שקולות. G מינימלית - אם נוריד ממנה תלות היא לא תהיה שקולה ל F.

אלגוריתם למציאת כיסוי מינימלי:

1: נפרק כל תלות שיש לה כמה אטרביוטים בצד ימין, לכמה תלויות.

2: נסתכל על צד שמאל של כל תלות, ונוודא שכל אטרביוט שם אכן נחוץ. אם ניתן להגיע לצד ימין עם חלק מאטרביוטים שבצד שמאל - נמחק את המיותרים.

3: נמחק תלויות מיותרות.

- **אלגוריתם לפירוק $3NF$:** משמר תלויות וללא אובדן.

נחשב את הכיסוי המינימלי G של F.

עבור כל תלות $X \Rightarrow Y$ ב G נוסיף את הסכמה XY.

אם אין סכמה שמכילה מפתח על, נוסיף בכמה של מפתח על.

נוריד סכמות מיותרות.

- **אלגוריתם רקורסיבי לפירוק $BCNF$:** ללא אובדן, אך לא נוכל להבטיח שימור תלויות.

אם הסכמה שקיבלנו היא ב $BCNF$ נחזיר אותה.

אחרת - נמצא תלות $X \Rightarrow Y$ שמפרה את $BCNF$, ונגדיר כך: $R_1 = X^+,\ R_2 = X \cup (R - X^+)$. כלומר שתי הטבלאות יכילו את X, בנוסף - טבלה אחת תכיל את הסגור, וטבלה שניה תכיל את המשלים של הסגור.

נקרא לאלגוריתם על R_1 ו R_2.

15.9 ניהול טרנזקציות:

- **הגדרה - טרנזקציה:** כשנרצה שפעולות מסויימות יתרחשו באופן אטומי נגדיר אותן בתוך טרנזקציה.
נתחיל עם $BEGIN\ TRANSACTION$ נכתוב את הפעולות שברצוננו לבצע, לבסוף נסיים עם אחת מהפקודות הבאות $COMMIT, END\ TRANSACTION, ROLLBACK$

- **הגדרה - כתיבה מלוכלכת:** טרנזקציה A ביצעה שינוי ולא ביצעה קומיט, טרנזקציה B שינתה את מה שטרנזקציה A כתבה.

- **הגדרה - קריאה מלוכלת:** טרנזקציה A ביצעה שינוי, טרנזקציה B קראה את השינוי לפני שטרנזקציה A ביצעה קומיט.

- **הגדרה - קריאה שאינה ניתנת לשחזור:** טרנזקציה A קוראת ערך מסויים במהלך התכנית, כשהיא ניגשת לקרוא אותו בפעם השניה היא מקבלת ערך ששונה ע"י טרנזקציה B.

- **הגדרה - קריאת פאנטום:** טרנזקציה A שולחת אילתה והכל פעם מקבלת עבורה סט ערכים אחר. לדוגמה - הטבלה התעדכנה בין הקריאות ונוספה שורה.

- **הגדרה - אנומליה סדרתית:** מתרחשת כאשר קיים שוני בין הרצת טרנזקציות במקביל, לבין הרצתן אחת אחרי השניה.

- **הגדרה - תזמון מלא:** תזמון של טרנזקציות כך שכל אחת מבצעת בסוך ריצתה $commit$ או $abort$.

- **הגדרה - תזמון סדרתי:** תזמון בו טרנזקציה מתחילה רק לאחר שהקודמת סיימה.

- **הגדרה - תזמון בר סידור:** אם הסידור שקול לתזמון סדרתי עבור הטרנזקציות שמבצעות $commit$, (נתעלם מטרנזקציות שעושות $abort$) .

- **קונפליקטים:** קונפליקט יכול לגרום לטרנזקציות שלנו לא להיות ברות סידור, במקרים הבאים: אם יש טרנזקציה אחת שכותבת אובייקט מסויים וטרנזקציה אחרת קוראת או כותבת אותו.

- **הגדרה - תזמון בר התאוששות:** נאמר שתזמון הוא בר התאוששות אם טרנזקציה מבצעת $commit$, רק לאחר שטרנזקציה שרצה לפניה וביצעה שינויים עשתה $commit$. (אין בעיה שתהיה קריאה או כתיבה מלוכלת, אלא רק שסדר ה$commit$ יתבצע לפי סדר השינויים).

- **הגדרה - $cascading\ aborts$:** פעולות של $abort$ שמתבצעות כתוצאה מכך שטרנזקציה אחרת ביצעה $abort$. תתרחש כאשר יש קריאה מלוכלכת.

- **הגדרה - תזמונים שקולי קונפליקטים:** נאמר כי שני תזמונים הם שקולי קונפליקטים, אם כל זוג פעולות שנמצאות בקונפליקט (של טרנזקציות שביצעו $commit$) מופיעות באותו הסדר (של הטרנזקציות) בשני התזמונים.

- **הגדרה - תזמון בר סידור קונפליקטים:** אם הוא שקול קונפליקטים לתזמון סדרתי.
בדיקה: ניצור גרף מכוון שבו כל קדקוד הוא טרנזקציה, נמתח צלע בין טרנזקציות אם יש קונפליקט שמערב אותן. התזמון יהיה בר סידור קונפליקטים אם אין בגרף מעגלים. כדי למצוא את התזמון הסדרתי אליו הוא שקול - נפעיל מיון טופולוגי.

15.9.1 פרוטוקולים מבוססי נעילות:

- **פרוטוקולים מבוססי נעילות:** נגדיר שני סוגי מנעולים - $Shared$ - מנעול לקריאה. $Exlusive$ - מנעול לכתיבה. מנעול משותף כולם יכולים להחזיק. בעוד שמנעול אקסקלוסיבי יכול להיות מוחזק ע"י טרנזקציה אחת בכל פעם בתנאי שאף טרנזקציה אחרת לא מחזיקה אף מנעול על האובייקט. כדי לגשת למשאב - הטרנזקציה צריכה להחזיק במנעול המתאים.

- **פרוטוקול** $2PL$**:** מבוסס על שני שלבים, שלב בקשת המנעולים ושלב השחרור. ברגע שטרנזקציה משחררת מנעול היא עוברת שלב, ולכן היא לא יכולה לחזור ולבקש מנעול.
  טענה: סידור שניתן להשגה ע"י $2PL$ הוא בר סידור קונפליקטים.

- **פרוטוקול** $Strict\ 2PL$**:** פרוטוקול מחמיר יותר, שמבטיח שהסידור הינו בר התאוששות. נאמר שתזמון הוא $Strict\ 2PL$ אם אין קריאה או כתיבה מלוכלכת. הדרישה היא שהשחרור יבוצע בסיום, לאחר פעולת $commit$ או $abort$.
  טענה: סידור שניתן להשגה ע"י $Strict\ 2PL$ הוא בר סידור קונפליקטים, בר התאוששות, ונמנע מ $cascading\ aborts$.

15.9.2 דדלוק:

- **טיפול:** מכיוון שאנו משתמשים במנעולים אנו יכולים להגיע למצב של קיפאון. לכן נצטרך לטפל בבעיה זו. תמיד נעדיף את הטרנזקציה שהתחילה ראשונה. וכשטרנזקציה מבטלת היא חוזרת עם הזמן ההתחלתי הראשון. יש שתי דרכים:
  1 $wait-die$**:** לא נפקיע מנעול מטרנזקציה, אלא: אם המבקשת התחילה אחרי המשתמשת - המבקשת תבטל את עצמה. אחרת - המבקשת תמתין.
  2 $wound-wait$**:** כן נפקיע מנעול, אם המבקשת התחילה לפני המשתמשת - המשתמשת תתבטל. אחרת - המבקשת תחכה.

- **שיטת** $waitsforgraph$**:** המערכת תחזיק גרף של בקשות, כל קדקוד יהיה טרנזקציה, ונמתח צלע בין טרנזקציות שמחכות אחת למנעול של השניה. אם יש מעגל בגרף יש דדלוק. במצב זה המערכת תקריס את אחת מהטרנזקציות.

15.9.3 פרוטוקול חותמות זמן:

- **הרעיון:** נרצה פרוטוקול שקול קונפליקטים לתזמון סדרתי, שבו הטרנזקציות רצות אחת אחרי השניה לפי זמן התחלתן. כל טרנזקציה תקבל זמן התחלה, וכל אובייקט יקבל חותמת זמן עבור הפעם האחרונה שקראו אותו - $RTS(A)$ ועבור הפעם האחררונה ששינו אותו $WTS(A)$.
  אין לי כח להמשיך

15.10 התאוששות:

- **מדיניות גניבה -** $Steal$**:** המערכת תכתוב לדיסק גם ערכים שלא ביצענו עליה $commit$.
  מדיניות אי הגניבה - $Force$**:** נכתוב לדיסק רק ערכים שבוצע עליהם $commit$.

- **מדיניות כפיה:** המערכת תכריח טרנזקציות לכתוב את הערכים לדיסק לאחר ביצוע $commit$.
  מדיניות אי כפיה: ערכים שבוצע עליהם $commit$ לא בהכרח ייכתבו לדיסק.

- **שיטת** WAL: ניהול הזכרון נעשה במדיניות גניבה + אי כפיה. המערכת תתחזק קובץ לוג שאליו היא כותבת את השינויים שנעשו - כל פעולה שטרנזקציה מבצעת נכתבת ללוג. מידיי פעם היא תכתוב את הלוג לדיסק - כאשר טרנזקציה מבצעת $commit$, לפני שכותבים בלוק לדסק, כשקובץ הלוג מתמלא בזכרון הראשי.

15.10.1 אלגוריתם התאוששות $ARIES$:

- ת

15.11 $NoSQL$:

- ך