

## מנגנון חישוב דינמי:

### תכנון אופן העבודה-

הפרויקט בוצע בשלבים מוגדרים, תוך הקפדה על מדידה מדויקת של זמני ביצוע והשוואה בין שיטות פיתוח שונות.

#### 1. הקמת סביבת עבודה ובסיס נתונים (SQL Server)

- בסיס נתונים: הוקם בסיס נתונים ב-SQL Server Management Studio (SSMS).
- מבנה הנתונים: הוגדרו ארבע טבלאות מרכזיות:
  - `t_data`: מכילה מיליון רשומות של נתונים מספריים רנדומליים (`a, b, c, d`) המשמשים כמשתנים בחישוב.
  - `t_targil`: מכילה את הנוסחאות הדינמיות השונות (פשוטות, מורכבות, ועם תנאים).
  - `t_result`: טבלת הפלט לשמירת תוצאת החישוב של כל רשומה ונוסחה.
  - `t_log`: טבלת הלוג שבה נשמרו זמני הביצוע של כל נוסחה בכל שיטת חישוב.

#### 2. מימוש שיטות החישוב

כדי להשוות ביצועים, מומשו שתי שיטות עיקריות, המייצגות גישות שונות:

1. Python (Pandas Vectorization):
2. SQL Stored Procedure (SQL דינמי)

#### 3. מנגנון הרצה ומדידת ביצועים

עבור כל אחת מהשיטות, בוצעה לולאה שרצה על כל הנוסחאות בטבלת `t_targil`. עבור כל נוסחה:

1. מדידת זמן: החלה מדידת זמן מדויקת.
2. חישוב: בוצע חישוב של הנוסחה הדינמית על כל מיליון הרשומות ב-`t_data`.
3. שמירת I/O: התוצאות נשמרו בטבלת `t_result`.
4. שמירת לוג: זמן הריצה הכולל (חישוב + I/O) נשמר בטבלת `t_log`.

תהליך זה הבטיח כי המדידה משקפת את הזמן הכולל הנדרש ליישום החישוב ושמירת הנתונים במערכת, ואיפשר השוואה אמינה בין השיטות.

## סיכום מפורט על השיטות:

### שיטת SQL Stored Procedure

הפרוצדורה המאוחסנת `sp_DynamicCalculation` נוצרה במטרה לבצע חישוב דינמי של נוסחאות על מיליון רשומות, תוך שמירה ומדידה של הביצועים.

#### 1. מהות השיטה וארכיטקטורה

- **Server-Side Execution:** כל החישובים, הלולאות וה-I/O מתבצעים ישירות במנוע ה-SQL Server. זהו מקור המהירות העיקרי, כיוון שהוא **מבטל את תקשורת הרשת** ואת הצורך בתרגום נתונים בין שפת תכנות ל-SQL.
- **SQL דינמי (`sp_executesql`):** הפרוצדורה בונה מחרוזת שאילתה מלאה (`@DynamicSQL`) בזמן ריצה, המשלבת את הנוסחה המאוחסנת (`@Formula`) לתוך פקודת `INSERT...SELECT`. לאחר מכן היא מריצה את המחרוזת באמצעות `EXEC sp_executesql`.
- **Batch Processing:** השאילתה הדינמית `INSERT...SELECT` מחשבת את הנוסחה על כל מיליון הרשומות בבת אחת ומכניסה את התוצאות ל-`t_result` בפקודת I/O אחת, וזו הדרך היעילה ביותר של SQL Server לעבוד עם נתונים בנפח גבוה.

#### 2. שלבים קריטיים בתוך הפרוצדורה

תיאור	מנגנון T-SQL	שלב
מנגנון Cursor (מצביע) סורק את טבלת <code>t_targil</code> ושולף את <code>targil_id</code> , <code>targil</code> ו- <code>tnai</code> אחת אחרי השנייה.	DECLARE CURSOR... FETCH NEXT... WHILE	לולאה על הנוסחאות
זמן הריצה נמדד לפני הרכבת ה-SQL הדינמי ומיד לאחריו, באמצעות <code>DATEDIFF(millisecond,...)</code> .	SET @StartTime = (())SYSDATETIME	מידת זמן
נדרש טיפול ממוקד לנוסחאות שאינן נתמכות בסינטקס גנרי:	IF @TargilID = 8 וכו'.	טיפול במקרים מיוחדים
הסינטקס <code>(POWER(base, exp))</code> משמש במקום <code>^</code> (כגון <code>((POWER(c, 2</code> .	נוסחאות חזקה (7, 10):	
נעשה שימוש ב- <code>NULLIF(b, 0)</code> כדי לוודא שאין ניסיון לחשב <code>(log(0\\$(</code> , ובכך נמנעת שגיאת <code>Msg 3623 (An invalid floating point operation occurred</code> .	נוסחה 8 (Log):	

טיפול בתנאים	CASE WHEN ...@TnaiWorking THEN	נוסחאות מותנות (11-13) מומרות לביטוי CASE תקין ב-SQL, והתנאי == מוחלף ב-== באמצעות REPLACE(@Tnai, '=', '==').
הפעלת SQL דינמי	EXEC sp_executesql @DynamicSQL	פקודה המריצה את השאילתה שנבנתה (הכוללת את נוסחת החישוב ואת פקודת ה-INSERT).
שמירת ביצועים	INSERT INTO t_log	זמן הריצה המדויק (כולל החישוב והשמירה ל-t_result) נשמר עבור כל נוסחה.

## הסבר מפורט: שיטת Python (וקטורизציה)

שיטה זו מבוססת על שימוש בספריות Python יעילות (NumPy ו-Pandas) כדי לבצע חישובים מחוץ למסד הנתונים (Client-Side) בצורה מהירה במיוחד, תוך שימוש במבני נתונים וקטוריים.

### 1. הכנה וטעינת נתונים

- **יצירת חיבור:** הפונקציה `connect_db()` יוצרת חיבור ישיר ל-SQL Server באמצעות ספריית `pyodbc` והגדרות המחרוזת ב-`config.py`.
- **טעינת נתונים (Batch Read):** הפונקציה `execute_fast_calculation()` טוענת את כל הנוסחאות מ-`t_targil` ואת כל מיליון רשומות הנתונים מ-`t_data` אל תוך זיכרון ה-RAM של ה-Python, במבנה `Pandas DataFrame` - טעינה זו מתבצעת פעם אחת בלבד.

### 2. ליבת החישוב: וקטורизציה (`calculate_vectorized_result`)

פונקציה זו היא המנוע של השיטה והיא אחראית על פירוש הנוסחה הדינמית (כמחרוזת) וחישובה על כלל הנתונים בבת אחת:

- **המרה לסינטקס:** הפונקציה מבצעת המרות תחביריות בסיסיות (כגון `^` ל-`**` של Python) כדי להתאים את הנוסחה למנוע החישוב.
- **חישוב גנרי (Numexpr):** עבור רוב הנוסחאות, נעשה שימוש ב-`data_df.eval(..., engine='numexpr')`. מנוע `Numexpr` הוא תוסף ממוטב של NumPy שמאפשר ל-Pandas לפרש את מחרוזת הנוסחה במהירות `C/Fortran`, ובכך מנצל את ה-CPU באופן מיטבי ללא צורך בלולאות Python איטיות.
- **טיפול מתמטי חריג (NumPy):** נעשה טיפול מיוחד בנוסחאות מורכבות:
  - **נוסחה 8**  $\log(|b|) + c$ : במקום להסתמך על `eval` (שיוצר שגיאות `log(0)`), הקוד משתמש ישירות בפונקציות `np.log` ו-`np.where` של NumPy כדי למנוע ערכי אפס כקלט, ובכך מבטיח יציבות מתמטית.
  - **נוסחה 7**  $\sqrt{c^2 + d^2}$ : נעשה שימוש ישיר בפונקציה `np.sqrt` כדי להבטיח את הביצועים הטובים ביותר לשורש ריבועי.
- **חישוב מותנה:** עבור נוסחאות עם תנאים, נעשה שימוש ב-`np.where`, שהיא דרך וקטורית מהירה לבצע `if-else` על מערכים שלמים בבת אחת.

### 3. מדידה ושמירת נתונים (I/O)

- **מידת זמן:** הקוד מודד את הזמן הכולל מרגע תחילת החישוב ועד לסיום השמירה ל-DB (`start_time` עד `end_time`), ובכך מספק מדד ביצועים מדויק.
- **הכנת התוצאה:** התוצאות מהחישוב (סדרת Pandas) הופכות ל-DataFrame בשם `results_batch` מוכן לשמירה.
- **שמירת הנתונים (Batch Insert):** נעשה שימוש ב-`results_batch.to_sql()` של Pandas, המשתמש ב-**SQLAlchemy Engine** כדי לבצע הכנסת נתונים בבלוקים (`Chunksize=50000`) לטבלת `t_result`. זו שיטת השמירה היעילה ביותר של Pandas.
- **שמירת לוג:** זמן הריצה שנמדד נשמר בטבלת `t_log` עבור הנוסחה הנוכחית, והחיבור נסגר לבסוף.