

מנגנון חישוב דינמי:

תכנון אופן העבודה-

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

1. הקמת סביבת עבודה ובסיס נתונים (SQL Server)

- בסיס נתונים: הוקם בסיס נתונים ב-SSMS (SQL Server Management Studio).
- מבנה הנתונים: הוגדרו ארבע טבלאות מרכזיות:
 - מכילה מיליון רשומות של נתונים מספריים רנדומליים (a, b, c, d) המשמשים כ משתנים בחישוב.
 - מכילה את הנוסחאות הדינמיות השונות (פשוטות, מורכבות, עם תנאים).
 - טבלת הפלט לשימרת תוצאות החישוב של כל רשומה ונוסחה.
 - טבלת הלוג שבה נשמרו זמני הביצוע של כל נסחה בכל שיטת חישוב.

2. מימוש שיטות החישוב

כדי להשווות ביצעים, מומשו שתי שיטות עיקריות, המיצגות גישות שונות:

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

3. מנגנון הרצה ומדידת ביצעים

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

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

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

סיכום מפורט על השיטות:

שיטה SQL Stored Procedure

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

1. מהות השיטה וארQUITטורה

- **SQL Server-Side Execution**: כל החישובים, הלולאות וה-O/I מתבצעים ישירות במנוע ה-SQL. זהו מקור המהירות העיקרי, כיוון שהוא **לבטל את תקשורת הרשות** ואת הצורך בתרגום נתונים בין שפת תכנות ל-SQL.
- **SQL דינמי (sp_executesql)**: ה프로그זידורה בונה מחרוזת שאילתת מלאה (DynamicSQL@) `INSERT...SELECT` (Formula@) לטור פקודות `EXEC sp_executesql`. בזמן ריצה, המשלבת את הנוסחה המאוחסנת `INSERT...SELECT` לאחר מכן היא מರיצה את המחרוזת באמצעות `t_result`.
- **Batch Processing**: השאילתת הדינמית `INSERT...SELECT` מחשבת את הנוסחה על כל מיליון הרשומות בבבאת אחת ומכניסה את התוצאות ל-`t_result` בפקודת O/I אחת, וזה הדרך העילית ביותר של SQL Server לעבוד עם נתונים בנפח גבוה.

2. שלבים קריטיים בתחום ה프로그זידורה

שלב	מנגנון T-SQL	תיאור
לולאה על הנוסחאות	<code>DECLARE CURSOR... FETCH NEXT... WHILE</code>	מנגנון Cursor (מצבייע) סורק את טבלת <code>t_targil</code> ושולף את <code>t_targil_id</code> , <code>t_targil</code> .
מדידת זמן	<code>SET @StartTime = ()SYSDATETIME</code>	זמן הריצה נמדד לפני הרכבת ה-SQL הדינמי ומיד לאחריו, באמצעות <code>DATEDIFF(millisecond,...)</code> .
טיפול במקרים מיוחדים	<code>IF @TargillID = 8</code>	נדרש טיפול ממוקד לנוסחאות שאין נתמכות בסינטקס גנרי:
	נוסחאות חזקה (10,7):	הסינטקס <code>POWER(base, exp)</code> משמש במקום <code>(c)^2</code> .
	נוסחה 8 (Log):	נעשה שימוש ב- <code>0(b, log)</code> כדי לוודא שאין ניסיון לחשב <code>log(0)</code> , ובכך מנענת שגיאת <code>Msg 3623 (An invalid floating point operation occurred</code> .

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

הסבר מפורט: שיטת Python (וקטוריזציה)

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

1. הכנה וטעינת נתונים

- יצרת חיבור: הפונקציה `connect_db()` יוצרת חיבור ישיר ל-SQL Server באמצעות ספריית `pyodbc` והגדירות המחרוזות ב-`py.config`.
- טעינת נתונים (Batch Read): הפונקציה `execute_fast_calculation()` טעונה את כל הנוסחאות `t targil_t` ואת כל מיליון רשומות הנתונים מ-`t_data` אל תוך זיכרון ה-RAM של ה-`Python`, ובמבנה `Pandas DataFrame` - . טעינה זו מתבצעת פעם אחת בלבד.

2. ליבת החישוב: וקטורייזציה (`calculate_vectorized_result`)

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

- המרה לסינטקס: הפונקציה מבצעת המרות תחביריות בסיסיות (כגון `^` ל-`**` של Python) כדי להתחאמים את הנוסחה למנוע החישוב.
- חישוב גנרי (`expr`): עבר רוב הנוסחאות, נעשה שימוש ב-`..., data_df.eval()`. מנוע `Numexpr` הוא תוסף מומלץ של NumPy שמאפשר ל-Pandas לפרש את מחרוזת הנוסחה **במהירות C/Fortran/C**, ובכך מנצל את ה-CPU באופן מיטבי ללא צורך בלילאות Python איטיות.
- טיפול מתמטי חריג (Num): נעשה טיפול מיוחד בנוסחאות מורכבות:
 - נוסחה 8 $(\$c + \$b)\log(\$a)$: מקום להסתמך על `eval` (שיזכר שגיאות $\log(0)$), הקוד משתמש שירות בפונקציות `log` ו-`np.where` של NumPy כדי למנוע ערכי אפס נקלוט, ובכך מבטיח יציבות מתמטית.
 - נוסחה 7 $(\$a^2 + \$c^2)^{\sqrt{\$b}}$: נעשה שימוש ישיר בפונקציה `sqrt.sqrt` כדי להבטיח את הביצועים הטובים ביותר לשורש ריבועי.
- חישוב מותנה: עבר נוסחאות עם תנאים, נעשה שימוש ב-`np.where`, שהוא דרך וקטוריית מהירה לבצע `if-else` על מערכיהם שלמים בבת אחת.

3. מדידה ושמירת נתונים (O/I)

- **מדידת זמן:** הקוד מודד את **זמן הכלל** מרגע תחילת החישוב ועד לסיום השמירה ל-DB. **start_time** (עד **end_time**), ובכך מספק מدد ביצועים מדויק.
- **הכנות התוצאות:** התוצאות מהחישוב (סדרת DataFrame Pandas) הופכות ל-**DataFrame** בשם **results_batch**.
- **שמירת הנתונים (Batch Insert):** נעשה שימוש ב-**sqlalchemy Engine** (Batch Insert) של Pandas המשמש ב-**SQLAlchemy Engine** כדי לבצע הכנסת נתונים בבלוקים (Chunksize=50000). Pandas מוכן לשמירה.
- **שמירת לוג:** זמן הריצה שנמדד נשמר בטבלה **log_t** עבור הנוסחה הנוכחית, והחיבור נסגר לבסוף.