

דוח מסכם: מנוע חישוב תשלומים דינמי

מגישה: בתשבע ברזיליא

תאריך: 03/12/2025

1. תקציר מנהליים (Executive Summary)

מטרת הפרויקט הייתה לפתח ארכיטקטורה לחישוב שכר ותשלים המבוססת על נוסחאות דינמיות המשתנות תדירות, ולבחון את הביצועים על סט נתונים של 1,000,000 רשומות.

בפרויקט מומשו והושוו שתי גישות: עיבוד בתוך בסיס הנתונים (SQL) ועיבוד בקוד אפליקטיבי (C#).

המסקנה: מנוע NET-C#-based ארכיטקטורת In-Memory הציג ביצועים עדיפים משמעותית בחישוב נטו (פי 10-6 מהר יותר מ SQL)-וגמישות רבה יותר בתחום הקוד ובארכיטקטורת המערכת.

2. פירוט הפתרונות הטכנולוגיים

שיטת A-Stored Procedure:

- עיקרון:** ביצוע החישוב מצד השרת (Server Side) באמצעות פרוצדורה מאוחסנת.
- הימוש:** שימוש ב Dynamic SQL ו Cursor-CDI לעבור על הנוסחאות, תרגום הפקודות לשפת T-SQL והרצת שאלות SELECT... INSERT INTO... מסיביות.
- יתרונות:** אין תעבורת רשת (הנתונים נשאים בשרת).
- חסרונות:** עומס כבד על ה Log-Transaction-Shell השרת, קשיי ביצוע לוגיקה מורכבת שאינה נתמכת ב SQL-טבע.

שיטת B-Optimized Engine C# .NET:

- עיקרון:** ביצוע חישוב בזיכרון (Memory Processing) בזיכרון האפליקציה.
- השימוש:**

1. **שליפה:** (Extract) טיענת כל המידע לזכרון באמצעות DataTable

2. **חישוב:** (Transform) שימוש במנוע החישוב הווקטורי של DOT-NET לחישוב מיליון שורות בשניות בודדות.

3. **שמירה:** (Load) שימוש בעקבCopy-SqlBulkWrite-לכתיבה מהירה במיוחד לDB-

- אופטימיזציה מיוחדת:** פיתוח מנגנון תרגום (Transpiler) שיודיע להמיר סינטקס עסקי (כגון ABS, if) לפקודות שהמנוע מבין, כדי לעקוף מגבלות טכניות ולשמור על ביצועים.

הארכיטקטורה והפתרונות שנבחנו:

בפרויקט יושמה ארכיטקטורת תוכנה מודרנית תוך הקפה על עקרונות הנדסיים (Separation of Concerns) הקוד ב-C# חולק ל-3 שכבות לוגיות:

- DAL: שכבה ייועדת לניהול התקשרות מול ה-DB (שליפה ו-Bulk Insert).
- BL: מנוע לוגי נקי שמבצע את הטרנספורמציה והחישובים בזיכרון.
- Program (Orchestrator Pipeline): ניהול תהליך העבודה.

3. ניתוח ביצועים והשוואה

המערכת נבדקה על סט של 8 נוסחות בדרגות קושי שונות, החל מחישובים ארכיטמטיים פשוטים ועד לוגיקה מותנית (if/else) ושימוש בפונקציות -

טבלת השוואת זמנים (זמן ריצה למיליאון רשותות):

המנצח	שיטה ב .NET #:	שיטה SQL Stored Procedure:	מדד להשוואה
#C- בפער משמעותי	1.5 ~שניות בלבד	לא ניתן להפרדה (משולב עם כתיבה)	זמן חישוב גטו(CPU)
יתרונות SQL- במצב מלא Cache	9.5 ~שניות-קבוע ויציב	15 שניות (ריצה ראשונה) / 8 שניות (ריצות עוקבות)	זמן ריצה כלול-End-to-End (End)

```

C:\Users\lsource\repos\Dynamic> .\Dynamax
Finished!
    > Calculation Time (CPU): 1.711 sec
    > Total Time (with DB Save): 9.761 sec

Processing Formula ID: 2 (c * 2)...
Finished!
    > Calculation Time (CPU): 1.437 sec
    > Total Time (with DB Save): 9.217 sec

Processing Formula ID: 3 ((a + b) * 8)...
Finished!
    > Calculation Time (CPU): 1.589 sec
    > Total Time (with DB Save): 9.352 sec

Processing Formula ID: 4 (ABS(d - b))...
Finished!
    > Calculation Time (CPU): 1.610 sec
    > Total Time (with DB Save): 9.428 sec

Processing Formula ID: 5 ((c * c) + (d * d))...
Finished!
    > Calculation Time (CPU): 1.727 sec
    > Total Time (with DB Save): 9.492 sec

Processing Formula ID: 6 (if(a > 500, b * 1.20, b * 1.05))...
Finished!
    > Calculation Time (CPU): 1.444 sec
    > Total Time (with DB Save): 9.384 sec

Processing Formula ID: 7 (if(c < 5, a + 200, a))...
Finished!
    > Calculation Time (CPU): 1.405 sec
    > Total Time (with DB Save): 9.771 sec

Processing Formula ID: 8 (if(a == c, 1, 0))...
Finished!
    > Calculation Time (CPU): 1.288 sec
    > Total Time (with DB Save): 9.342 sec

All Done!

```

ניתוח הממצאים:

בבדיקה עמוקה של התוצאות, עולה תמונה ארכיטקטונית ברורה:

1. **צואר הבקבוק (The Bottleneck):** הגורם המרכיב העיקרי העיקרי אינו הלוגיקה של הקוד, אלא מהירות הכתיבה לדיסק (O Bound). בעוד שזמן החישוב ב-C# הוא אףוי,

כמעט (כ-1.5 שניות למיליאון שורות!), פועלות השמירה הפיזית למסד הנתונים אורך כ-8 שניות בשתי השיטות באופן זהה.

2. **אפקט ה SQL-Caching-ב:** ניכר פער בביצועי ה SQL-בין הריצה הראשונה ("קרה" - 15 שניות) לבין הריצות הבאות ("חמות" - 8 שניות), הנובע מכך שה SQL-טוען את הנתונים לזכרון. לעומת זאת, ביצועי ה #C-נסחרו יציבים ועקביים בכל הריצות.
3. **מסקנה לעתיד:** בחישובים מורכבים יותר שיתווסף בעתיד, היתרון של ה #C-ילך יגדל משמעותית, מכיוון שימושו של מעבד (CPU) באפליקציה פנויים וחזקים יותר מאשר מנוע ה SQL-שעומס בניהול הطالאות.

4. אתגרים ופתרונות ייצורתיים

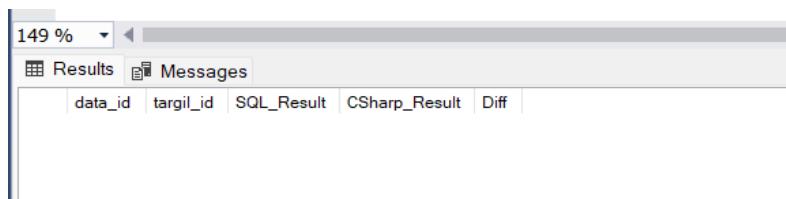
במהלך הפיתוח התמודדתי עם דרישות לשינטקס ייחודי ($==$, f) ולפונקציות (ABS) שלא מתמכות ישירות בכל המנועים.

- **אתגר:** מנוע ה SQL-וה DataTable לא מכירם את הפקודה if או ABS בצורתן הכלכלנית.
- **פתרון הייצורתי:** מימוש שכבת Transpilation (תרגום קוד). לפני החישוב, המערכת סורקת את הנוסחה ומתרגמת אותה:
 - if הופך ל IF-.
 - (x)ABS הופך לוגיקה של (x < 0, -x).
- **התוצאה:** המערכת תומכת בכל נוסחה עסקית, מבלי לפגוע בביצועים וambil' להזדקן למספריות חייזניות כבדות.

5. אמינות הנתונים(Data Integrity)

בוצעה הצלבה מלאה בין תוצאות ה SQL-لتוצאות ה #C-הורץ סקריפט השוואת כל 8 מיליון רשומות.

תוצאה: 0 פערים. שתי השיטות הניבו תוצאות זהות לחלווטין.



data_id	targil_id	SQL_Result	CSharp_Result	Diff

6. דיון, מגבלות והמלצות להמשך

במהלך העבודה בוצע ניתוח מעמיק של היתונות והחסכנות (Trade-offs) בבחירה המנוע #C-ב.

הדילמה הארכיטקטונית: נבחר להשתמש במנוע DataTable וביחסוב וקטורי.

- **היתרון:** ביצועים פנומנליים (כ-1.5 שניות למיליאון רשומות).

- **המגבלה:** המנווע אינו תומך באופן טבעי בפונקציות מדעיות מורכבות (כגון SQRT, LOG, POWER) אלא בפעולות ארכיטמטיות ולוגיות בלבד.

פתרונות שנסקלו: נבחנה אפשרות להשתמש בספריות חיצונית (כגון NCalc או Roslyn) הותמכות בכל הפונקציות המתמטיות. עם זאת, בדיקות הראו שפתרונות אלו עובדים בשיטת "שורה-אחר-שורה" (Row-by-Row) ומאטים את זמן הריצה פי 10 ועודתא.

המסקנה והמלצת הסופית: עבור מערכת שכר (המתאפיינת לרוח בחיבור, חיסור, אחוזים ותנאים לוגיים), **המלצת היא חד-משמעות להישאר עם הפתרון הנוכחי (C# Optimized)** בשל המהירות הקритית. במידה ובעתיד ידרשו חישובים מדעיים מורכבים (שורשים וחזקות), ניתן יהיה לשלב פתרון היברידי: חישובים רגילים ב-C# וחישובים מדעיים באמצעות -SQL Servern