## חלק 3

### סוגי JOIN:

אני מכיר 5 סוגים שניתן לכתוב בשאילתא + 3 סוגים שחלק מהDB מריצים מאחורי הקלעים ע"מ לייעל את התהליך של הJOIN

1. INNER JOIN - מוצא התאמה בין שתי הטבלאות, משמיט שורות ללא התאמה בין שתי הטבלאות
2. LEFT (OUTER) JOIN - לוקח את כל השורות בטבלה הראשונה ומנסה למצוא התאמה בטבלה השניה. אם אין התאמה - שורה חוזרת עם מידע מטבלה הראשונה וNULL במקום התאמה
3. RIGHT (OUTER) JOIN - כמו LEFT, רק טבלה מובילה היא השניה בשאילת
4. FULL (OUTER) JOIN - איחוד בין פעולות RIGHT ו LEFT, מחזיר את כל השורות והתאמה אם קיימת
5. CROSS JOIN - מכפלה קרטזית בין 2 הטבלאות - לכל שורה בטבלה הראשונה "מוצמדת" כל שורה בטבלה השניה ללא צורך בהתאמה.

בנוסף 3 אלגוריתמים שבעזרתם מנוע DB מייעל את הJOIN:

1. MERGE JOIN - אלגוריתם עיקרי לJOIN, סורק את שתי הטבלאות ועושה התאמה לפי התנאי של JOIN. מתאים לטבלאות גדולות עם אינדקסים.
2. HASH JOIN - בונה HASH TABLE מטבלה הראשונה ודוגם ערכים מהטבלה השניה. מתאים במקרים כשאין אינדקסים.
3. NESTED LOOPS JOIN - עובר על כל הערכים של הטבלה הראשונה ומחפש התאמה מהטבלה השניה. מתאים עבור טבלאות קטנות.

### טבלה זמנית:

השם אומר בעד עצמו - טבלה זמנית הינה טבלה שמוחקת את עצמה כשנגמר הsession מול DB.

קיימת 2 סוגים:

1. טבלה זמנית מקומית - זמינה רק לsession נוכחי ונמחקת אוטומטית בסוף הsession.
2. טבלה זמנית גלובלית - זמינה לכל הsessions ונמחקת אוטומטית כשsession של אחרון המשתמש נגמר.

שימוש עיקרי הוא שמירה נתונים זמניים (caching) בתוך stored procedure או שאילתות מורכבות.

### DELETE vs TRUNCATE

שתי הפעולות משמשות למחיקת נתונים מהטבלה. ההבדלים הם:

Delete:

1. ניתן להגביל את המחיקה בעזרת WHERE
2. כל שורה שנמחקת נרשמת ללוג של DB
3. פעולה יחסית איטית (תלוי בכמות השורות)

Truncate:

1. פעולה שמוחקת את כל הנתונים מהטבלה ללא אפשרות לWHERE
2. שורות שנמחקו לנרשות ללוג, בזכות זה הפעולה מהירה יחסית
3. מאפס את עמודות identity במידה והן auto-increment
4. לא ניתן להריץ במידה וקיים FK בטבלאות אחרות

### View

זה טבלה "וירטואלית" עם שאילתא מוגדרת מראש. לעומת טבלאות קבועות לא שומרת נתונים, אלה מבצעת את השאילתא כל פעם שעושים SELECT מתוך view.

שימוש עיקרי הוא להסתיר שאילתות מורכבות ולשפר את הreadability של השאילתות מורכבות.

### Commit

פעולה לשמירה של כל הנתונים לתוך DB באופן קבוע.

חייב לבוא בצמד עם BEGIN TRANSACTION.

לפני COMMIT כל השינויים נשמרים בTempDB וניתן לבטל אותם ע"י פקודת ROLLBACK. אחרי ביצוע COMMIT לא ניתן לשחזר את השינויים.

\*\* חלק 4 בעמוד הבא

## חלק 4

### Interface vs Abstract class

אלו הכלים להגדיר פולימורפיזם. שניהם מגדירים התנהגות לclass היורש (abstract class) או class המממש (interface).

Interface מגדיר רק signature של הmethods ללא מימוש (החל מגרסה 8 ניתן להוסיף default implementation). **ניתן לממש מספר interfaces בclass אחד.**

Abstract class יכול להכיל גם propeties, גם asbtract methods וגם מימוש של methods.

**Class יכול לרשת רק מclass אחד.**

### Virtual methods

זו הדרך לclass האב לספק מימוש ברירת מחדש לclass היורש. ניתן לדרוס את המימוש בclass היורש בעזרת override.

לא ניתן להגדיר private virtual.

יש הבדל קטן בביצועים לעומת non-virtual method בגלל runtime polymorphism (שימוש של compiler בvtable במקום קריאה ישירה לnon-virtual method)

### Design Pattern

1. **Factory method** - אני משתמש בDP הזה בעיקר בטסטים כשצריך להתחל אובייקט לפי context. שימוש עיקרי - encapsulate של יצרת אובייקט. תורם לגמישות בקוד ומניעת tight coupling בין האובייקטים.
2. **Repository** - מוסיף שכבת אבסטרקציה בין לוגיקה עסקית לבין DB. מפשט את הפעולות מול DB, משפר testability. אבן יסוד בגישת Clean Architecture.
3. **Unit Of Work** - שכבת אבסטרקציה נוספת בין לוגיקה עסקית לבין DB. מאפשר לנהל טרנזקציות כשיש מספר repositories. מבטיח שכל השינויים בכל repositories ישמרו בDB בבת אחת, או אף שינוי לא ישמר. Entity Framework ממש את הpattern ע"י DbContext.

### Razor

Razor זה סך הכל מנוע של ASP.NET ליצירת HTML. זה משמש "לשתול" קוד dotnet לתוך html.

תהליך זה רץ **בשרת**. בקליינט רץ javascript.

דומה ל SSR בangular.

### IQueryable vs IEnumerable

**IEnumerable** - עיבוד נתונים מתבצע in-memory. לפני העיבוד כל הנתונים נטענים לתוך הזכרון.

**IQueryable** - עיבוד נתונים מתבצע בremote (למשל בDB). לפני העיבוד LINQ Provider מתרגם את ה-expression tree לשפה אחרת (למשל SQL).

## חלק 5

### Change Detection

השם מדבר בעד עצמו - זהו תהליך של זיהוי שינויים בstate וביצוע שינויים בDOM בהתאם.

Angular משתמש בzone.js למימוש של התהליך זה.

תהליך זה אמור להתבצע אוטומטי כל פעם כשיש שינוי בstate - למשל כשמשתמש לחץ על כפתור, הגיע תשובה בHTTP או יש timer. יחד עם זאת ניתן להפעיל את התהליך "ידנית" בעזרת ChangeDetectorRef.

### Pipes

הינם משמשים לשינוי של מידע לפני התצוגה בדף. למשל פורמט של תאריכים, מטבע וכו.

השינוי לא חל על מידע מקור אלה רק ההתצוגה.

קיימים 2 סוגים של pipes:

Pure pipe: עבור אותו input תמיד מתקבל אותו output. מתבצע רק כש input משתנה. Angular שומר את התוצאות ולא מחשב אותן מחדש. יעלות מאוד.

Impure pipe: מתבצע כל detection cycle ולא יעילות במיוחד.

דוגמה לשימוש:

<p>{{ today | date: ‘dd/MM/yyyy’ }}</p