March 28, 2025

Wrocław

Sink or soar?
Apex Dynamic Formulas in action

Barbara Laskowska & Fabrice Challier

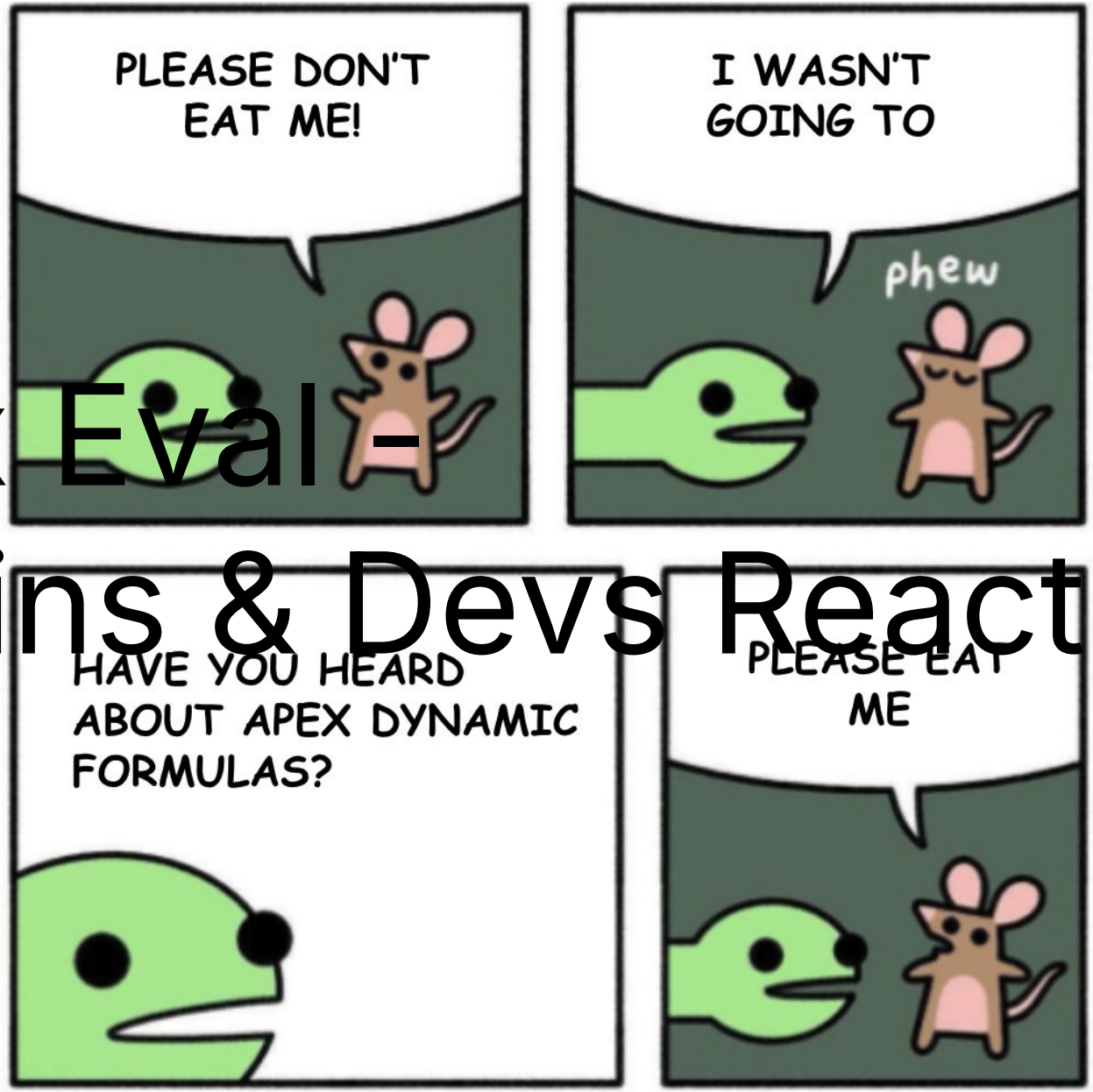
Barbara Laskowska
Solution Consultant

Fabrice Challier
Salesforce Architect

?

POLISH DREAMIN'

Apex Eval - Admins & Devs Reaction



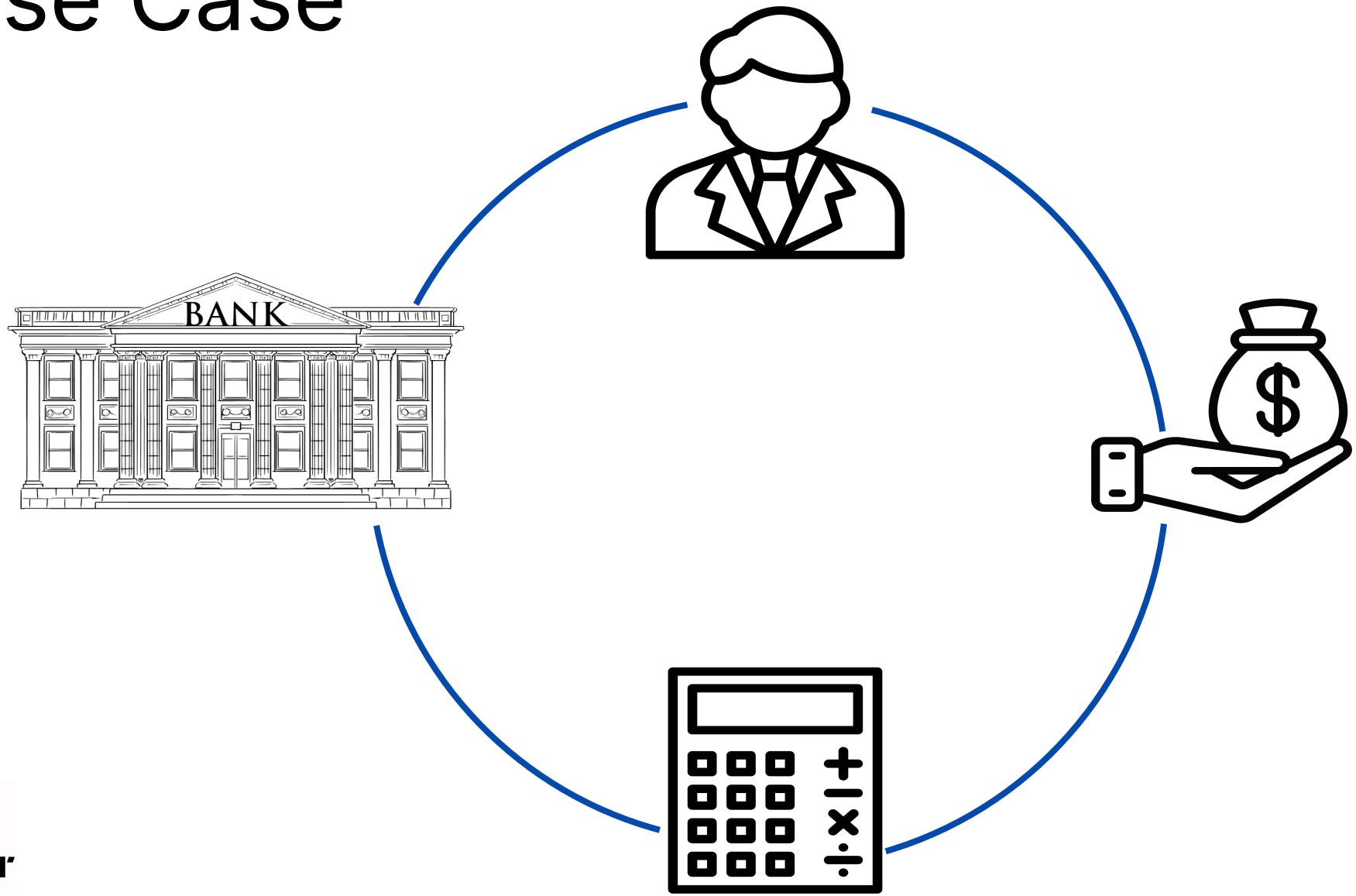
POLISH DREAMIN'

Use Case

Use Case

- Bank uses Salesforce to store information about the clients and loan applications.
- Financial Advisors need to simulate loan scenarios in Salesforce, showing clients their monthly payments for different loan options.

Use Case



$$Monthly\ Payment = \frac{Loan\ Amount * r}{n * (1-(\frac{n}{n+r})^{tn}}$$

Interest Rate

$$Monthly\ Payment = \frac{\boxed{Loan\ Amount} * \boxed{r}}{\boxed{n} * \left(1-\left(\frac{n}{n+r}\right)^{\boxed{tn}}\right.}$$

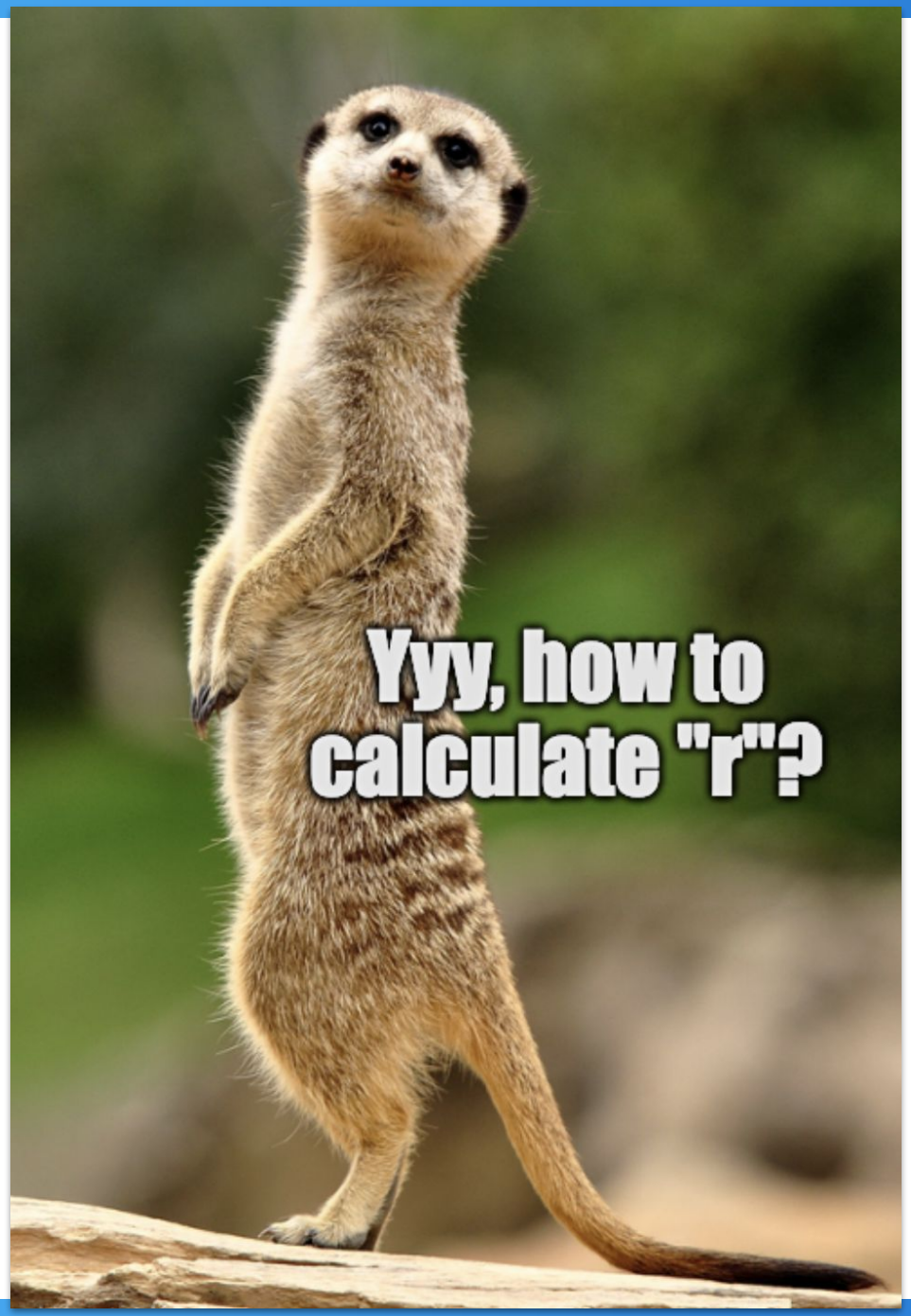
Yearly number of Installments

Total Number of Installments during the loan term

Modified by Financial Advisor

Calculated based on formulas

POLISH DREAMIN'


Yyy, how to
calculate "r"?

Interest Rate rules

Loan Amount	Interest Rate (Loan Term)		
	<5 years (Short Term)	5-10 years (Medium Term)	>10 years (Long Term)
>$20,000 - $40,00	8%	10%	11%
$40,00 - $80,000	9%	12%	13%
>$80,000	11%	13%	15%

Drama Starts Here

Technical Requirements

1. Implement formulas for *Interest Rate* rules.

2. Store interest rate rules in a configurable and scalable way, allowing non-developers to update them.

Developer

Admin

Credit Officer

POLISH DREAMIN'

Act 1 - Admin Solution

```
IF(
    AND(Loan_Amount__c > 20000, Loan_Amount__c <= 40000, Loan_Term__c < 5),
    0.08,
    IF(
        AND(Loan_Amount__c >= 40000, Loan_Amount__c <= 80000, Loan_Term__c > 5),
        0.10,
        IF(
            AND(Loan_Amount__c > 40000, Loan_Amount__c <= 80000, Loan_Term__c > 5),
            0.12,
            IF(
                AND(Loan_Amount__c > 40000, Loan_Amount__c <= 80000, Loan_Term__c < 5),
                0.09,
                IF(
                    AND(Loan_Amount__c > 80000, Loan_Term__c < 5),
                    0.11,
                    IF(
                        AND(Loan_Amount__c > 80000, Loan_Term__c > 5),
                        0.13,
                        NULL
                    )
                )
            )
        )
    )
)
```

✓ 1. Implement formulas for *Interest Rate* rules.

✗ 2. Store interest rate policies in a configurable and scalable way, allowing non-developers to update them.

Act 2 - Dev Solution

```
trigger LoanInterestRateTrigger on Loan__c (before insert, before update) {
    for (Loan__c loan : Trigger.new) {
        loan.Interest_Rate__c = LoanInterestRateHelper.calculateRate(loan.Loan_Amount__c, loan.Loan_Term__c);
    }
}
```

```
public class LoanInterestRateHelper {
    public static Decimal calculateRate(Decimal loanAmount, Decimal loanTerm) {
        if (loanAmount >= 20000 && loanAmount <= 40000 && loanTerm < 5) {
            return 0.08;
        } else if (loanAmount >= 20000 && loanAmount <= 40000 && loanTerm > 5) {
            return 0.10;
        } else if (loanAmount > 40000 && loanAmount <= 80000 && loanTerm > 5) {
            return 0.12;
        } else if (loanAmount > 40000 && loanAmount <= 80000 && loanTerm < 5) {
            return 0.09;
        } else if (loanAmount > 80000 && loanTerm < 5)
            return 0.11;
        } else if (loanAmount > 80000 && loanTerm > 5)
            return 0.13;
        }
        return null; // Default case if no condition is
    }
}
```

✓ 1. Implement formulas for *Interest Rate* rules.

✗ 2. Store interest rate policies in a configurable and scalable way, allowing non-developers to update them.

What if?

Loan Amount	Interest Rate (Loan Term)		
	<5 years (Short Term)	5-10 years (Medium Term)	>10 years (Long Term)
>$0,00 - $40,00	8%	10%	11%
$40,00 - $80,000	9%	12%	13%
$80,000 - $100,000	11%	13%	15%
$100,000 - $200,000	11%	13%	15%
$200,000 - $500,000	11%	13%	15%
>$500,000	11%	13%	15%

POLISH DREAMIN'

Dev & Admin Strategy

Developer

Admin

POLISH DREAMIN'

What if...
we use Dynamic Formulas ?

	dynamic formula		dynamic formula
if	record.Loan_Amount__c>=bound1 && record.Loan_Amount__c<=bound2	**rate__c** =	rate1
else if	record.Loan_Amount__c>=bound2 && record.Loan_Amount__c<=bound3	**rate__c** =	rate2
else if	record.Loan_Amount__c>=bound3 && record.Loan_Amount__c<=bound4	**rate__c** =	rate3

etc ...

---

record.Loan_amount__c : field value from LoanApplication__c

boundX,boundX,rateX : field value from InterestmentRatePolicy__c exposed as property

LoanApplication__c and **InterestmentRatePolicy__c** : unrelated custom objects

Apex Dynamic Formulas

Step 1 - Define Interest Rate rules in custom object

Interest Rate Policies

All

New | Import | Change Owner | Printable View | Assign Label

3 items • Sorted by Loan Term • Filtered by All interest rate policies • Updated a few seconds ago

	Interest Rate Definition	Loan Term	Boundary1	Boundary2	Boundary3	Boundary4	rate1	rate2	rate3
1	Short Term	5	10,000	40,000	60,000	500,000	5	6	7
2	Medium Term	10	10,000	40,000	60,000	500,000	6	7	8
3	Long Term	15	10,000	40,000	60,000	500,000	7	8	9

POLISH DREAMIN'

Apex Dynamic Formulas

Step 2 - Use Custom Metadata Types for calculation logic

Action	Label	LoanCalculationRules Name	Condition	FieldValue	Order	targetField
Edit | Del	range1	range1	record.Loan_Amount__c>=bound1 && record.Loan_Amount__c<=bound2	Rate1	1,000	Loan_Rate__c
Edit | Del	range2	range2	record.Loan_Amount__c>=bound2 && record.Loan_Amount__c<=bound3	Rate2	2,000	Loan_Rate__c
Edit | Del	range3	range3	record.Loan_Amount__c>=bound3 && record.Loan_Amount__c<=bound4	Rate3	3,000	Loan_Rate__c
Edit | Del	Monthly Payment	monthlyPayment	true	record.Loan_Amount__c*record.Loan_Rate__c/1200/(1-(1/(1+record.Loan_Rate__c/1200)^(12*paymentPlan.number__c)))	4,000	monthly_cost__c
Edit | Del	Loan Cost	Loan_Cost	true	record.monthly_cost__c*12*paymentPlan.number__c-record.Loan_Amount__c	5,000	total_interests__c

POLISH DREAMIN'

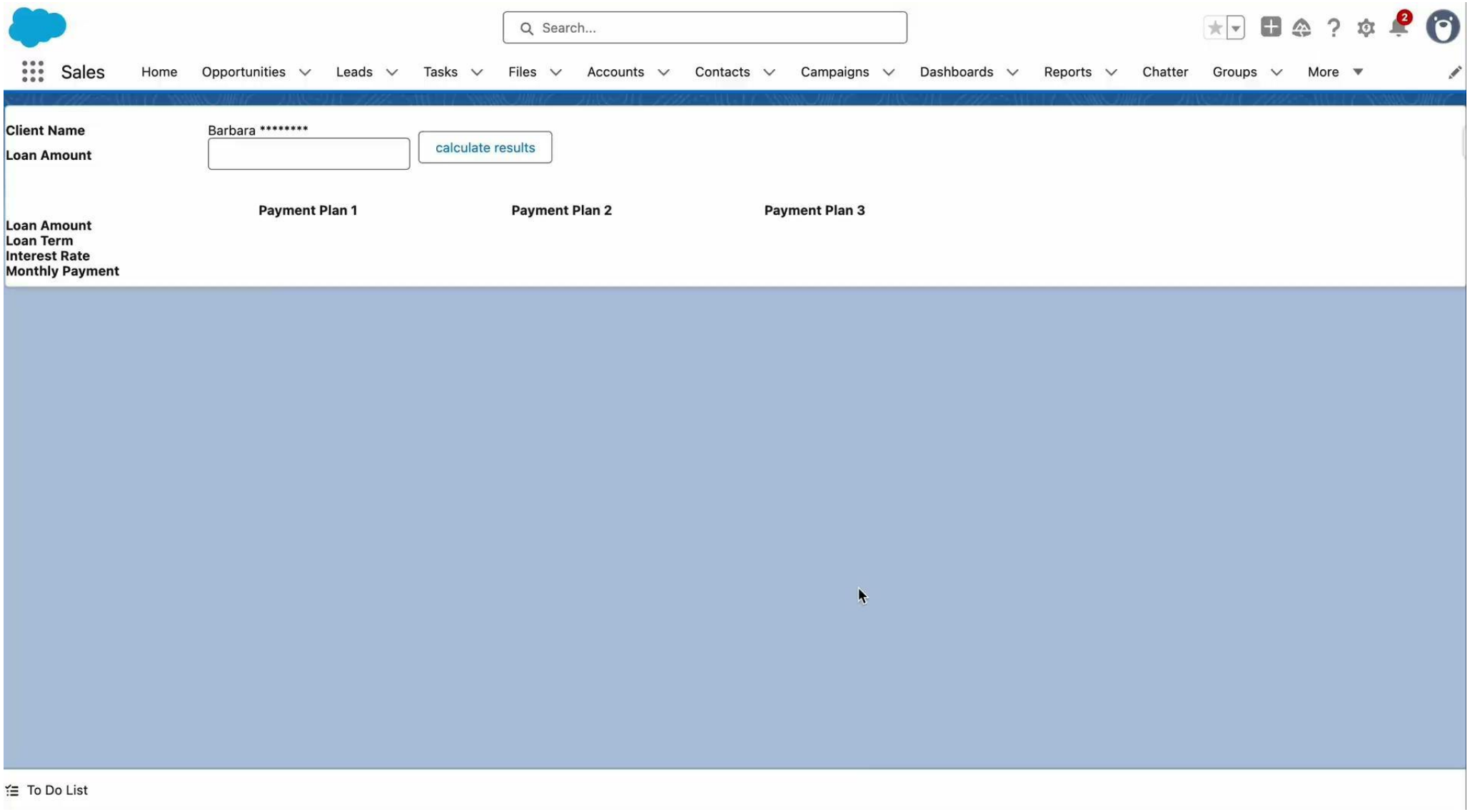
Apex Dynamic Formulas

Step 3 - Use Dynamic Formula and Visualize the calculation results

Demo Time!

Apex Dynamic Formulas

Step 3 - Visualize the calculation results



Deeper Look at Apex Dynamic Formula

Apex Dynamic Formulas (How it work)

Step 1- Building a formula (validate and complie the formula)

```
FormulaEval.FormulaInstance formulaInstance = Formula.builder()
        .withReturnType ('THE FORMULA RESULT TYPE')
        .withType ('TYPE OF THE RECORD/CLASS')
        .withFormula ('FORMULA TO EVALUATE')
        .build();
```

Apex Dynamic Formulas (How it work)

Step 2 (optional) - Retrieving the record information

Formula example for an Account :

'For contact named '+Name +' , account name is ' + account.name+ ' with owner ' + owner.name

extract needed fields :

formulaInstance.getReferencedFields() will give you

```
List<String> neededFields = {'Name', 'account.name','owner.name'};
```

Retrieving the record with needed fields for the formula :

```
Account formulaAcct = Database.query('Select '+String.join(',',neededFields )+' FROM ACCOUNT WHERE
Id=:XXXX);
```

POLISH DREAMIN'

Apex Dynamic Formulas (How it work)

Step 3 - Retrieving the formula result

To get the result , we just apply this command :

```
Object result = formulaInstance.evaluate(formulaAcct );
```

you can cast and use the value now

Working with Apex wrapper class

public records : simple/complex

public property

formula example :

'For contact named '+**cont**.Name +' , account name is ' +**cont**.account.name+ ' with owner ' + **cont**.owner.name '. The account has '+ **nbContacts**+ ' related contacts'

'The opportunity name is '+**opp**.Name +' with amount ' + **opp**.Amount

getReferencedFields example :

{'**cont**.Name','**cont**.account.name','**cont**.owner.name','nbContacts','**opp**.Name','**opp**.Amount '}

Managing a “complex” record

getReferenceFields will return this :
{‘**cont**.Name’,’**cont**.account.name’,’**cont**.owner.name’,’nbContacts’,’**opp**.Name’,’**opp**.Amount ’}

problem : we need to get all the needed fields (including relational ones)

Step 1
clean the result to keep only real fields related to the “complex” record.
result :
List<String> cleanFields = {‘name’,’account.name’,’owner.name’};

Step 2
Build the complex record with all the fields (including relational ones) that the formula expect.
result :
Contact complexRecord = Database.query('SELECT ' + String.join(cleanFields,',') + ' FROM Contact WHERE Id=:recordId');

POLISH DREAMIN’

Managing a “simple” record

getReferenceFields will return this :
{‘**cont**.Name’,’**cont**.account.name’,’**cont**.owner.name’,’nbContacts’,’**opp**.Name’,’**opp**.Amount ’}

problem : we need to get all the needed fields (but no relational ones)

Step 1
get all fields for the object (wether it is used or not, we don’t care)
result :

```
 List<String> allfields = new
List<String>(Opportunity.SObjectType.getDescribe().fields.getMap().keySet());
```

Step 2
Build the complex record with all the fields (including relational ones) that the formula expect.
result :

```
Opportunity simpleRecord = Database.query('SELECT ' + String.join(allfields,',') + ' FROM Contact
WHERE Id=:recordId');
```

POLISH DREAMIN’

Managing a wrapper property

getReferenceFields will return this :
{'**cont**.Name','**cont**.account.name','**cont**.owner.name','nbContacts','**opp**.Name','**opp**.Amount '}

problem : we need to define what the property nbContacts will return

result could be for example :

```
public String nbContacts{
    get {
        return ''+Database.countQuery('SELECT COUNT() FROM Contact WHERE AccountId =\''+cont.AccountId+'\'');
    }
}
```

Working with Apex wrapper class

1. Load and compile all formulas, store all referenceFields

Working with Apex wrapper class

1. Load and compile all formulas, store all referenceFields

2. Load “complex” records using referenceFields

Working with Apex wrapper class

1. Load and compile all formulas, store all referenceFields

2. Load “complex” records using referenceFields

3. Initialize the wrapper with records (simple can be calculated within the wrapper if needed)

Working with Apex wrapper class

1. Load and compile all formulas, store all referenceFields

2. Load “complex” records using referenceFields

3. Initialize the wrapper with records (simple can be calculated within the wrapper if needed)

4. For each ordered rules , if condition is true
   a. store the value in the record if needed for other formula computation
   b. store the information to the list of info to display to the user

Working with Apex wrapper class

1. Load and compile all formulas, store all referenceFields

2. Load “complex” records using referenceFields

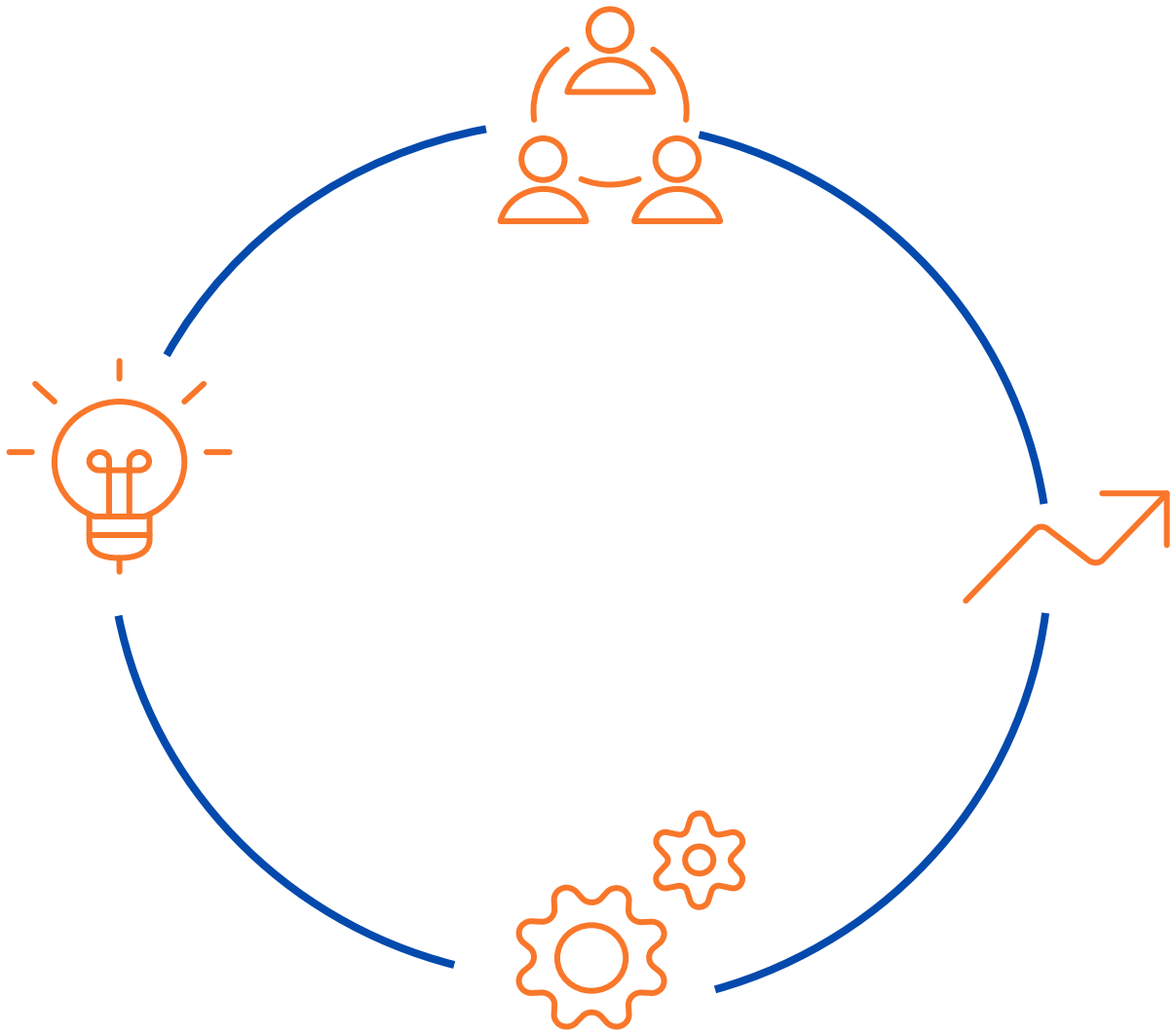
3. Initialize the wrapper with records (simple can be calculated within the wrapper if needed)

4. For each ordered rules , if condition is true
   a. store the value in the record if needed for other formula computation
   b. store the information to the list of info to display to the user

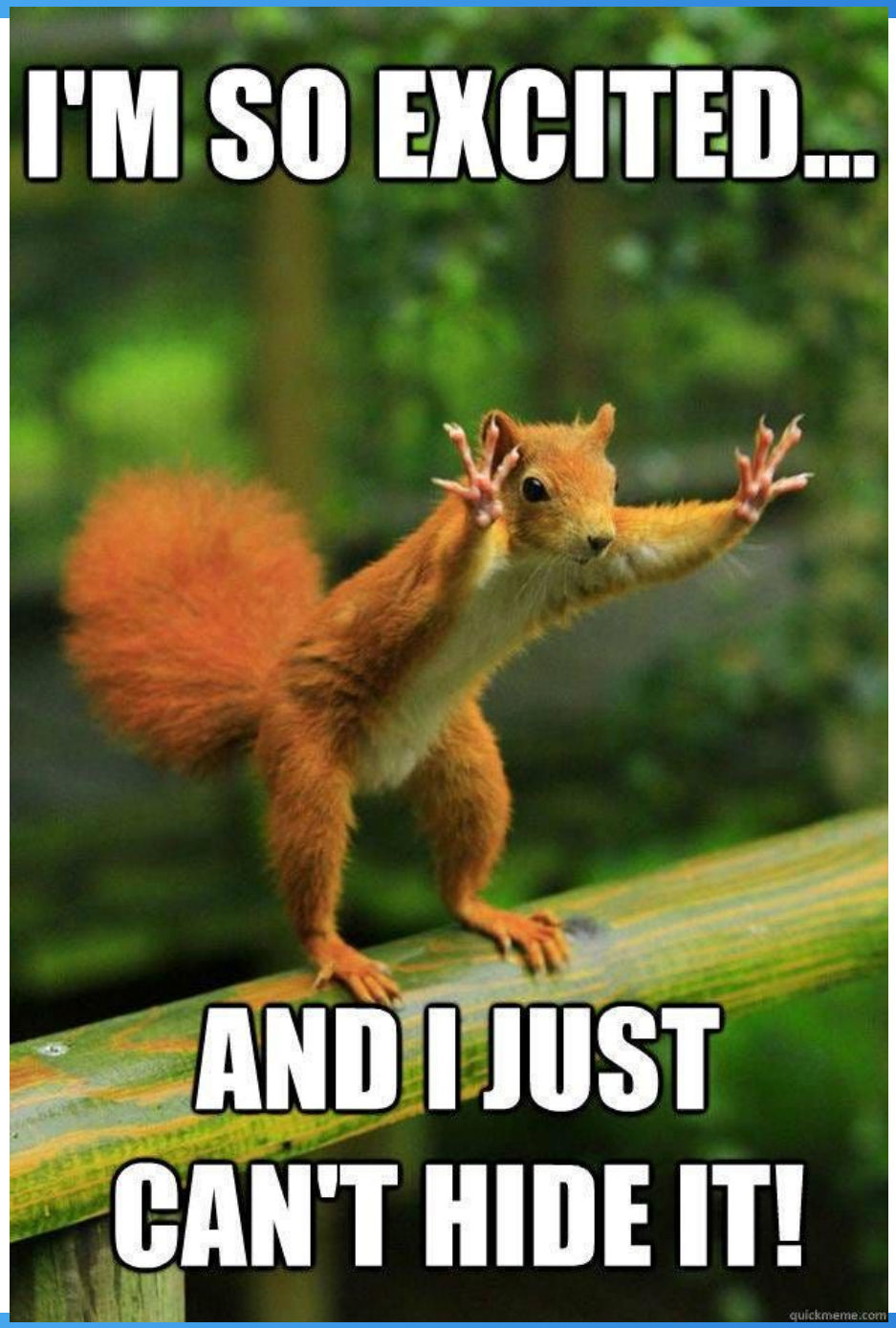
5. Send information back to the LWC.

Summary

Power of Apex Dynamic Formulas



POLISH DREAMIN'



POLISH DREAMIN'

Thank You

In progress :)

POLISH DREAMIN'

Give feedback in the app!
Your input matters!