# BusinessCentral Bootcamp

16TH - 17TH APRIL 2021

AL Language: Coding for performance

Tomáš Kapitán

Dynamics 365 NAV/BC Systems Architect

Microsoft

Powercommunity.com

Register Now

1000 Free Tickets

# AL Language

## Coding for performance

Kepty.CZ

- Basic tips & tricks
  - Flowfields
  - Find Operations
  - Caching Data
  - Loops with data modifications
- Partial Records
  - What  & How
  - Performance evaluation

# Flowfields

Kepty.CZ

FlowFields are virtual fields. The value of these fields are not stored in the database and are calculated once CalcFields or SetAutoCalcFields methods are called.

```
field(60; Amount; Decimal)
{
    AutoFormatExpression = "Currency Code";
    AutoFormatType = 1;
    CalcFormula = Sum("Sales Line".Amount WHERE("Document Type" = FIELD("Document Type"),
                                                "Document No." = FIELD("No.")));
    Caption = 'Amount';
    Editable = false;
    FieldClass = FlowField;
}
field(61; "Amount Including VAT"; Decimal)
{
    AutoFormatExpression = "Currency Code";
    AutoFormatType = 1;
    CalcFormula = Sum("Sales Line"."Amount Including VAT" WHERE("Document Type" = FIELD("Document Type"),
                                                               "Document No." = FIELD("No.")));
    Caption = 'Amount Including VAT';
    Editable = false;
    FieldClass = FlowField;
}
```

# Flowfields

Kepty.CZ

Do not use CalcFields within loops

```
/// <summary>
/// BAD example!
/// For each Sales Header the field Amount has to be calculated one-by-one
/// </summary>
0 references
procedure FlowfieldExample_Bad()
var
    SalesHeader: Record "Sales Header";
    Counter: Integer;
begin
    if SalesHeader.FindSet() then
        repeat
            SalesHeader.CalcFields(Amount);
            if SalesHeader.Amount < 0 then
                Counter += 1;
        until SalesHeader.Next() < 1;
    Message(Format(Counter));
end;
```

```
/// <summary>
/// Good example!
/// Application server can load Amount field together with all standard fields from Sales Header
/// </summary>
0 references
procedure FlowfieldExample_Good()
var
    SalesHeader: Record "Sales Header";
    Counter: Integer;
begin
    SalesHeader.SetAutoCalcFields(Amount);
    if SalesHeader.FindSet() then
        repeat
            if SalesHeader.Amount < 0 then
                Counter += 1;
        until SalesHeader.Next() < 1;
    Message(Format(Counter));
end;
```

Use SetAutoCalcFields instead.

Server can load content of the field at once (or in batch at least).

# Flowfields

Kepty.CZ

And do not forget to clear AutoCalcField once the value of the calculated field is not used anymore.

To clear Automatically Calculated fields, call SetAutoCalcFields again without any parameter.

If you call AutoCalcFields with another field, the previous one is not replaced, but the new field is added to auto-calculated fields.

```
/// <summary>
/// Good example!
/// Application server can load Amount field together with all standard fields from Sales Header
/// </summary>
0 references
procedure FlowfieldExample_Good2()
var
    SalesHeader: Record "Sales Header";
    Counter: Integer;
begin
    SalesHeader.SetAutoCalcFields(Amount);
    if SalesHeader.FindSet() then
        repeat
            if SalesHeader.Amount < 0 then
                Counter += 1;
        until SalesHeader.Next() < 1;
    Message(Format(Counter));

    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);
    SalesHeader.SetAutoCalcFields();
    if SalesHeader.FindSet() then
        repeat
            DoSomethingElseThatDoesNotUseAmountField();
        until SalesHeader.Next() < 1;
end;
```

# Find operations

Kepty.CZ

- IsEmpty()
  - „SELECT TOP 1 NULL FROM xxx“
- Count()
  - „SELECT COUNT(*) FORM xxx“
- FindFirst(), FindLast()
  - „SELECT TOP 1 * FROM xxx“
- Find()
  - „SELECT TOP X * FROM xxx“ (X is server-defined using server statistics and server config.)
- FindSet()
  - „SELECT * FROM xxx“

# Find operations – Record Exists

Kepty.CZ

## Record.IsEmpty() Method

Determines whether a table or a filtered set of records is empty.

```
/// <summary>
/// Always check whether the record exists using IsEmpty()
/// </summary>
0 references
procedure FindOperation_Example1()
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if not SalesHeader.FindFirst() then
        DoSomething();
    // Bad
    if SalesHeader.Count() = 0 then
        DoSomething();
    // Good
    if SalesHeader.IsEmpty() then
        DoSomething();
end;
```

# Find operations – Record Exists

Kepty.CZ

```
/// <summary>
/// If you need a value of any field within the record, you can check existence using the Find operation
/// </summary>
0 references
procedure FindOperation_Example2()
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if not SalesHeader.IsEmpty() then
        if SalesHeader.FindFirst() then
            DoSomethingWithValue(SalesHeader."No.");
    // Bad
    if not SalesHeader.IsEmpty() then begin
        SalesHeader.FindFirst();
        DoSomethingWithValue(SalesHeader."No.");
    end;
    // Good
    if SalesHeader.FindFirst() then
        DoSomethingWithValue(SalesHeader."No.");
end;
```

# Find operations – Record Exists

Kepty.CZ

```
/// <summary>
/// If there is some static condition (or less demanding calculation), evaluate this condition first.
/// This approach will generate DB query only when MyCondition is true = fewer queries = better performance.
/// </summary>
/// <param name="MyCondition">Boolean parameter, condition</param>
0 references
procedure FindOperation_Example3(MyCondition: Boolean)
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if SalesHeader.FindFirst() and MyCondition then
        DoSomethingWithValue(SalesHeader."No.");
    // Good
    if MyCondition then
        if SalesHeader.FindFirst() then
            DoSomethingWithValue(SalesHeader."No.");
end;
```

# Find operations – First/Last record

Kepty.CZ

```
/// <summary>
/// To get the first/last record, always use FindFirst or FindLast
/// </summary>
0 references
procedure FindOperation_FirstLast_Example()
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if SalesHeader.FindSet() then
        DoSomethingWithValue(SalesHeader."No.");
    // Bad
    if SalesHeader.Find('-') then
        DoSomethingWithValue(SalesHeader."No.");
    // Good
    if SalesHeader.FindFirst() then
        DoSomethingWithValue(SalesHeader."No.");

    // Bad
    if SalesHeader.Find('+') then
        DoSomethingWithValue(SalesHeader."No.");
    // Good
    if SalesHeader.FindLast() then
        DoSomethingWithValue(SalesHeader."No.");
end;
```

To retrieve only one record, use FindFirst() or FindLast() methods.

## Record.FindFirst(), Record.FindLast() Methods

Finds the first/last record in a table based on the current key and filter.

# Find operations – Loops

Kepty.CZ

To retrieve a set of records, use FindSet() method.

## Record.FindSet() Method

Finds a set of records in a table based on the current key and filter.

```
/// <summary>
/// For looping through records, use FindSet, never FindFirst or FindLast.
/// </summary>
0 references
procedure FindOperation_FindNext_Example()
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if SalesHeader.FindFirst() then
        repeat
        until SalesHeader.Next() < 1;
    // Bad
    if SalesHeader.FindLast() then
        repeat
        until SalesHeader.Next(-1) > -1;

    // Good
    if SalesHeader.FindSet() then
        repeat
        until SalesHeader.Next() < 1;
end;
```

# Find operations – Loops

Kepty.CZ

```
/// <summary>
/// For looping through records, use FindSet. In some cases Find('-') can be also used.
/// </summary>
/// <param name="MyCondition">Boolean parameter, condition</param>
procedure FindOperation_FindNext2_Example(MyCondition: Boolean)
var
    SalesHeader: Record "Sales Header";
begin
    SalesHeader.SetRange("Document Type", SalesHeader."Document Type"::Order);

    // Bad
    if SalesHeader.Find('-') then
        repeat
        // No breakpoint
        until SalesHeader.Next() < 1;
    // Good
    if SalesHeader.FindSet() then
        repeat
        until SalesHeader.Next() < 1;


    // Acceptable (if we have some knowledge about data)
    if SalesHeader.Find('-') then
        repeat
            if MyCondition then
                break;
        until SalesHeader.Next(-1) < 1;
    // Good (if we have no predictions about data)
    if SalesHeader.FindSet() then
        repeat
            if MyCondition then
                break;
        until SalesHeader.Next() < 1;
end;
```

FindSet() and Find(-) methods have the same behaviour (and, usually, same performance in the Business Central).

However, it is customary to use FindSet when all records are read and Find(-) when the loop can be break dynamically.

# Caching Data

Kepty.CZ

To avoid unnecessary recalculation of expensive results, consider caching the data and refresh the cache regularly.

Example

- Stock level for e-Shop
  - API page based on Inventory field from Stockkeeping Unit table?
  - NO!
    - Every call to this API has to recalculate the current stock level!
  - Solution
    - Cache Stockkeeping units (interval depends on business requirements)

```
page 50005 "TKA Stockkeeping Units API"
{
    PageType = API;
    Caption = 'stockkeepingUnits';
    APIPublisher = 'keptyCZ';
    APIGroup = 'businessCentralBootcamp';
    APIVersion = 'v1.0';
    EntityName = 'stockkeepingUnit';
    EntitySetName = 'stockkeepingUnits';

    SourceTable = "Stockkeeping Unit";

    InsertAllowed = false;
    ModifyAllowed = false;
    DeleteAllowed = false;
    DelayedInsert = true;

    layout
    {
        0 references
        area(Content)
        {
            0 references
            repeater(GroupName)
            {
                0 references
                field(locationCode; Rec."Location Code") { }
                0 references
                field(itemNo; Rec."Item No.") { }
                0 references
                field(inventory; Rec.Inventory) { }
            }
        }
    }
}
```

# Caching Data

Kepty.CZ

## Solution

- Create a new table (Cached Stockkeeping Unit) with four standard fields - Location Code, Item No., Variant Code and Inventory.
- Copy values from Stockkeeping Unit to the Cached table.

```
procedure CachingData_Example1()
var
    StockkeepingUnit: Record "Stockkeeping Unit";
    TKACachedStockkeepingUnit: Record "TKA Cached Stockkeeping Unit";
begin
    StockkeepingUnit.SetAutoCalcFields(Inventory);
    if StockkeepingUnit.FindSet() then
        repeat
            if TKACachedStockkeepingUnit.Get(StockkeepingUnit."Location Code", StockkeepingUnit."Item No.", Stockkeep
                TKACachedStockkeepingUnit.Inventory := StockkeepingUnit.Inventory;
                TKACachedStockkeepingUnit.Modify();
            end else begin
                TKACachedStockkeepingUnit.Init();
                TKACachedStockkeepingUnit."Location Code" := StockkeepingUnit."Location Code";
                TKACachedStockkeepingUnit."Item No." := StockkeepingUnit."Item No.";
                TKACachedStockkeepingUnit."Variant Code" := StockkeepingUnit."Variant Code";
                TKACachedStockkeepingUnit.Inventory := StockkeepingUnit.Inventory;
                TKACachedStockkeepingUnit.Insert();
            end;
        until StockkeepingUnit.Next() < 1;
end;
```

# Caching Data - loops with data modification

Kepty.CZ

## Solution

- Better approach: Do all reads without any insert/modify (do not open write transaction on SQL server) first.

```
procedure CachingData_Example2()
var
    StockkeepingUnit: Record "Stockkeeping Unit";
    TKACachedStockkeepingUnit: Record "TKA Cached Stockkeeping Unit";
    TempTKACachedStockkeepingUnit: Record "TKA Cached Stockkeeping Unit" temporary;
begin
    TempTKACachedStockkeepingUnit.DeleteAll();
    StockkeepingUnit.SetAutoCalcFields(Inventory);
    if StockkeepingUnit.FindSet() then
        repeat
            TempTKACachedStockkeepingUnit.Init();
            TempTKACachedStockkeepingUnit."Location Code" := StockkeepingUnit."Location Code";
            TempTKACachedStockkeepingUnit."Item No." := StockkeepingUnit."Item No.";
            TempTKACachedStockkeepingUnit."Variant Code" := StockkeepingUnit."Variant Code";
            TempTKACachedStockkeepingUnit.Inventory := StockkeepingUnit.Inventory;
            TempTKACachedStockkeepingUnit.Insert();
        until StockkeepingUnit.Next() < 1;

    if TempTKACachedStockkeepingUnit.FindSet() then
        repeat
            if TKACachedStockkeepingUnit.Get(TempTKACachedStockkeepingUnit."Location Code", TempTKACachedStockkeepingUnit."Item No.", TempTK
                TKACachedStockkeepingUnit.Inventory := TempTKACachedStockkeepingUnit.Inventory;
                TKACachedStockkeepingUnit.Modify();
            end else begin
                TKACachedStockkeepingUnit.Init();
                TKACachedStockkeepingUnit."Location Code" := TempTKACachedStockkeepingUnit."Location Code";
                TKACachedStockkeepingUnit."Item No." := TempTKACachedStockkeepingUnit."Item No.";
                TKACachedStockkeepingUnit."Variant Code" := TempTKACachedStockkeepingUnit."Variant Code";
                TKACachedStockkeepingUnit.Inventory := TempTKACachedStockkeepingUnit.Inventory;
                TKACachedStockkeepingUnit.Insert();
            end;
        until TempTKACachedStockkeepingUnit.Next() < 1;
end;
```

# Partial Records

# Partial Records

Kepty.CZ

Allows specifying fields that should be loaded when accessing SQL based data

SQL Query

- Without Partial Records
  - All fields from all extensions are loaded

- With Partial Records
  - Only system fields & defined fields are loaded

Non-loaded fields are loaded automatically using the Just-In-Time (JIT) mechanism (big impact on performance!)

```
procedure ComputeArithmeticMean(): Decimal;
var
    Item: Record Item;
    SumTotal: Decimal;
    Counter: Integer;
begin
    Item.SetLoadFields(Item."Standard Cost");
    if Item.FindSet() then begin
        repeat
            SumTotal += Item."Standard Cost";
            Counter += 1;
        until Item.Next() = 0;
        exit(SumTotal / Counter);
    end;
end
```

# Partial Records

Kepty.CZ

[Ok := ] Record.SetLoadFields( [Fields: Any, ...] )

- Specifies a set of fields that the server should load from the database.
- The last specified fields are valid.

[Ok := ] Record.AddLoadFields( [Fields: Any, ...] )

- Add additional field to already defined fields. In comparison to SetLoadFields, it does not rewrite already defined fields.

Ok := Record.AreFieldsLoaded( Fields: Any, ... )

- Checks whether the fields are already retrieved from the database.

[Ok := ] Record.LoadFields(Fields: Any,...)

- Loads fields on-fly.

# Partial Records – Table without extensions

Kepty.CZ

```
procedure ForEachGLEntries(var GLEntry: Record "G/L Entry"): Decimal;
var
    SumTotal: Decimal;
    Counter: Integer;
begin
    if GLEntry.FindSet() then begin
        repeat
            SumTotal += GLEntry.Amount;
            Counter += 1;
        until GLEntry.Next() = 0;
        exit(SumTotal / Counter);
    end;
end;
```

```
procedure PartialRecords()
var
    GLEntry: Record "G/L Entry";
begin
    // Load all fields
    ForEachGLEntries(GLEntry);
    Clear(GLEntry);

    // Load only Amount field
    GLEntry.SetLoadFields(Amount);
    ForEachGLEntries(GLEntry);
    Clear(GLEntry);
end;
```

# Partial Records – Table without extensions

Kepty.CZ

First
- Without Partial Records

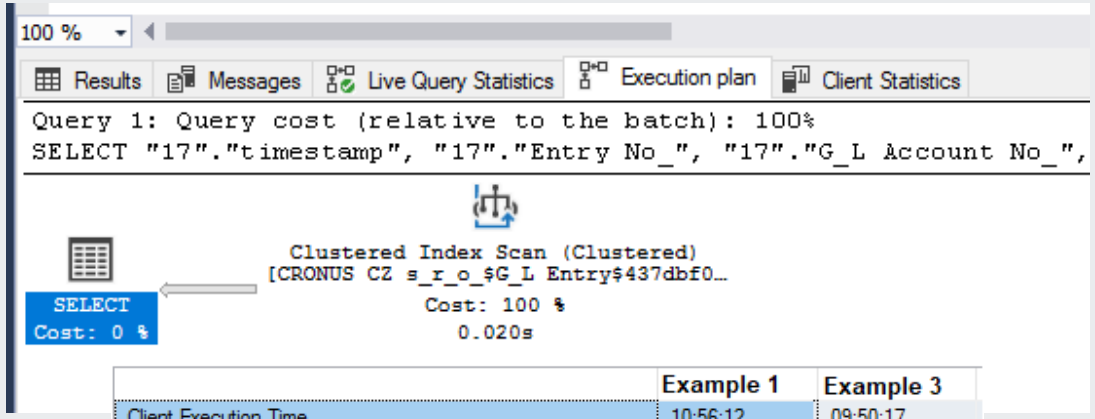
```
SELECT
    "17"."timestamp",
    "17"."Entry No_",
    ... All Table Fields (56 fields hidden) ...
    "17"."Closed",
    "17"."$systemId",
    "17"."$systemCreatedAt",
    "17"."$systemCreatedBy",
    "17"."$systemModifiedAt",
    "17"."$systemModifiedBy"
FROM "CRONUS".dbo."CRONUS CZ s_r_o_$G_L Entry$437dbf0e-84ff-417
ORDER BY "Entry No_" ASC OPTION(OPTIMIZE FOR UNKNOWN, FAST 50)
```

Second
- With Partial Records
- SetLoadFields(Amount)

```
SELECT
    "17"."timestamp",
    "17"."Entry No_",
    "17"."G_L Account No_",
    "17"."Amount",
    "17"."Bal_ Account Type",
    "17"."Source Type",
    "17"."FA Entry Type",
    "17"."$systemId",
    "17"."$systemCreatedAt",
    "17"."$systemCreatedBy",
    "17"."$systemModifiedAt",
    "17"."$systemModifiedBy"
FROM "CRONUS".dbo."CRONUS CZ s_r_o_$G_L Entry$437dbf0e-84ff-417a
ORDER BY "Entry No_" ASC OPTION(OPTIMIZE FOR UNKNOWN, FAST 50)
```

# Partial Records – Table without extensions

Kepty.CZ



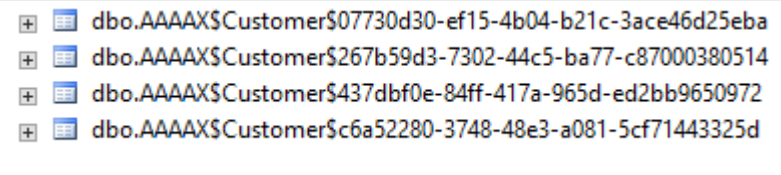

| | Example 1 | Example 3 |
|---|---|---|
| Client Execution Time | 10:56:12 | 09:50:17 |
| Query Profile Statistics | | |
| Number of INSERT, DELETE and UPDATE statements | 0 → | 0 → |
| Rows affected by INSERT, DELETE, or UPDATE stateme... | 0 → | 0 → |
| Number of SELECT statements | 5 → | 4 → |
| Rows returned by SELECT statements | 2739 → | 2739 → |
| Number of transactions | 0 → | 0 → |
| Network Statistics | | |
| Number of server roundtrips | 5 → | 4 → |
| TDS packets sent from client | 5 → | 4 → |
| TDS packets received from server | 332 → | 90 → |
| Bytes sent from client | 3768 → | 1204 → |
| Bytes received from server | 1342607 → | 354743 → |
| Time Statistics | | |
| Client processing time | 92 → | 31 → |
| Total execution time | 109 → | 46 → |
| Wait time on server replies | 17 → | 15 → |

# Partial Records – Table extensions

Kepty.CZ

Every TableExtension object add new SQL table.

- Number of tables = BaseApp + Number of tableextensions




All tables have the same Primary Key (derived from the table object).
Every query must merge results from all tables using Inner Join.

```
SELECT
    *
FROM [AAAAX$Customer$437dbf0e-84ff-417a-965d-ed2bb9650972] A
INNER JOIN [AAAAX$Customer$07730d30-ef15-4b04-b21c-3ace46d25eba] B ON (A.[No_] = B.[No_])
INNER JOIN [AAAAX$Customer$267b59d3-7302-44c5-ba77-c87000380514] C ON (A.[No_] = C.[No_])
INNER JOIN [AAAAX$Customer$c6a52280-3748-48e3-a081-5cf71443325d] D ON (A.[No_] = D.[No_])
```

# Partial Records – Table without extensions

Kepty.CZ

```
procedure ForEachGLEntries(var GLEntry: Record "G/L Entry"): Decimal;
var
    SumTotal: Decimal;
    Counter: Integer;
begin
    if GLEntry.FindSet() then begin
        repeat
            SumTotal += GLEntry.Amount;
            Counter += 1;
        until GLEntry.Next() = 0;
        exit(SumTotal / Counter);
    end;
end;
```

```
procedure PartialRecords()
var
    GLEntry: Record "G/L Entry";
begin
    // Load all fields
    ForEachGLEntries(GLEntry);
    Clear(GLEntry);

    // Load only Amount field
    GLEntry.SetLoadFields(Amount);
    ForEachGLEntries(GLEntry);
    Clear(GLEntry);
end;
```

# Partial Records – Table extensions

Kepty.CZ

First
- Without Partial Records

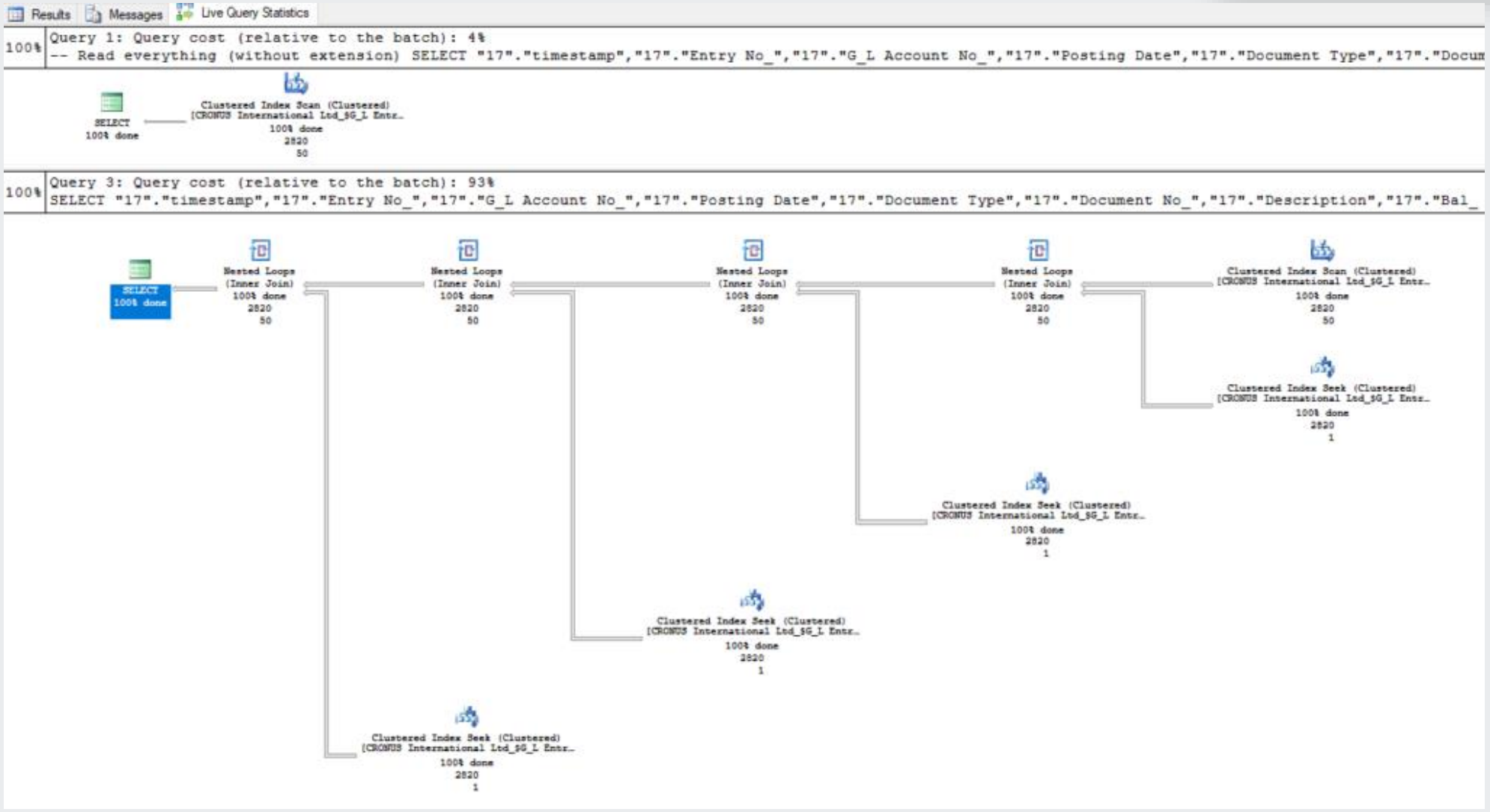
```
SELECT
    "17"."timestamp",
    "17"."Entry No_",
    "17"."G_L Account No_",
    ... All Table Fields (56 fields hidden) ...
    "17"."Last Modified DateTime",
    "17_e1"."TKA Veeery Long Description",
    "17_e1"."TKA Veeery Long Description 2",
    "17_e3"."TKA Veeery Long Description 5",
    "17_e3"."TKA Veeery Long Description 6",
    "17_e4"."TKA Veeery Long Description 7",
    "17_e4"."TKA Veeery Long Description 8",
    "17_e5"."TKA Veeery Long Description 3",
    "17_e5"."TKA Veeery Long Description 4",
    "17"."$systemId",
    "17"."$systemCreatedAt",
    "17"."$systemCreatedBy",
    "17"."$systemModifiedAt",
    "17"."$systemModifiedBy"
FROM "CRONUS".dbo."CRONUS International Ltd_$G_L Entry$437dbf0e-84ff-41
JOIN "CRONUS".dbo."CRONUS International Ltd_$G_L Entry$425df973-0cda-41
JOIN "CRONUS".dbo."CRONUS International Ltd_$G_L Entry$fa14d844-0ad9-42
JOIN "CRONUS".dbo."CRONUS International Ltd_$G_L Entry$fa14d844-0ad9-42
JOIN "CRONUS".dbo."CRONUS International Ltd_$G_L Entry$fa14d944-0ad9-42
ORDER BY "Entry No_" ASC OPTION(OPTIMIZE FOR UNKNOWN, FAST 50)
```

Second
- With Partial Records
- SetLoadFields(Amount)

```
SELECT
    "17"."timestamp",
    "17"."Entry No_",
    "17"."G_L Account No_",
    "17"."Amount",
    "17"."Bal_ Account Type",
    "17"."Source Type",
    "17"."FA Entry Type",
    "17"."$systemId",
    "17"."$systemCreatedAt",
    "17"."$systemCreatedBy",
    "17"."$systemModifiedAt",
    "17"."$systemModifiedBy"
FROM "CRONUS".dbo."CRONUS CZ s_r_o_$G_L Entry$437dbf0e-84ff-417
ORDER BY "Entry No_" ASC OPTION(OPTIMIZE FOR UNKNOWN, FAST 50)
```

# Partial Records – Table extensions

Kepty.CZ



Kepty.CZ

# Q & A

www.kepty.cz
kapitan@kepty.cz