
DYNAMICSCON
POWERED BY
DUG
20-23 Sept 2021
DynamicsCon.com

# AL Language: Coding for Performance

- **Basics**
  - Find (-/first/last/set), IsEmpty, Count
  - Table Keys
  - Pass-by-value & Pass-by-reference
- **Partial Records**
  - Impact on performance
  - Problems & Best practises

# Find (first/last/set), IsEmpty, Count

# FindFirst()
# FindLast()

**Both methods find the first/last record in a table based on the current key and filters.**

***Find(-) / Find(+) return also first/last record but load set of records (not just the first/last one) hence always use FindFirst/FindLast.***

SQL:

```
SELECT TOP X
*
FROM xxx
```

```al
/// <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;
```

# FindSet()

FindSet() returns a set of records from the table based on the current key and filters.

SQL FindSet():

```
SELECT
*
FROM xxx
```

```
/// <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;
```

# Table Keys

# Keys (till v18)

- **Important Properties**
  - Clustered
  - Unique
  - Enable

- **Know about**
  - MaintainSiftIndex
  - MaintainSqlIndex

```
keys
{
    - reference
    key(PK; "Entry No.")
    {
        Clustered = false;
    }
    - reference
    key(CK; "Posting Date", "G/L Account No.")
    {
        Clustered = true;
    }
    - reference
    key(UK; "Unique Identifier")
    {
        Unique = true;
        Enabled = false;
    }
}
```

# Keys (new with v19)

- **Included fields for indexes**
  - New Table Key property
  - Creates SQL keys with included columns
  - Could be especially useful with partial records

- **ColumnStoreIndex**
  - New Table property
  - Possible replacement (beta!) to SIFT indexes

```
table 50100 "TKA Table Example"
{
    Caption = 'Table Example';
    ColumnStoreIndex = Amount;
    DataClassification = CustomerContent;

    fields
    { ...
    }

    keys
    {
        - reference
        key(PK; "Entry No.")
        { ...
        }
        - reference
        key(CK; "Posting Date", "G/L Account No.")
        {
            Clustered = true;
            IncludedFields = Amount;
        }
        - reference
        key(UK; "Unique Identifier")
        { ...
        }
    }
}
```

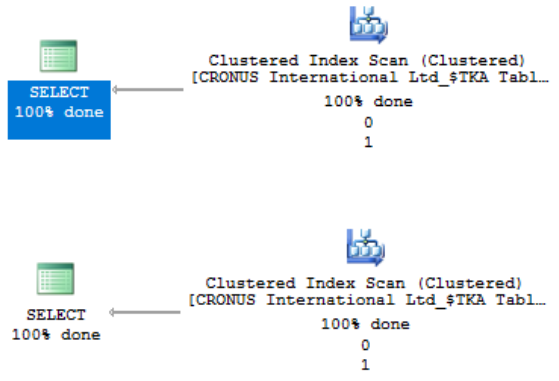
# Included fields for indexes

With BC <19 no included fields exist

To get non-index value -> (Non) Clustured Index **SCAN**

```
SELECT
    [Amount]
FROM [default].[dbo].[CRONUS International Ltd_$TKA Table Example$08292c4a-b44a-4cc6-8993-1558a379a059]
WHERE
    [Posting Date] >= GETDATE()

SELECT
    [Amount],[Unique Identifier]
FROM [default].[dbo].[CRONUS International Ltd_$TKA Table Example$08292c4a-b44a-4cc6-8993-1558a379a059]
WHERE
    [Posting Date] >= GETDATE()
```



# Included fields for indexes

```
table 50100 "TKA Table Example"
{
    Caption = 'Table Example';
    ColumnStoreIndex = Amount;
    DataClassification = CustomerContent;

    fields
    { ...
    }

    keys
    {
        - reference
        key(PK; "Entry No.")
        { ...
        }
        - reference
        key(CK; "Posting Date", "G/L Account No.")
        {
            Clustered = true;
            IncludedFields = Amount;
        }
        - reference
        key(UK; "Unique Identifier")
        { ...
        }
    }
}
```

```
CREATE UNIQUE NONCLUSTERED INDEX [$CK] ON [dbo].[CRONUS Internatio
(
    [Posting Date] ASC,
    [G_L Account No_] ASC,
    [Entry No_] ASC
)
INCLUDE ([Amount])
WITH (
    PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON
) ON [PRIMARY]
```

# Included fields for indexes

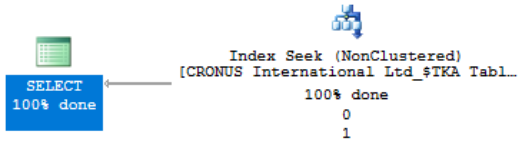
With BC 19

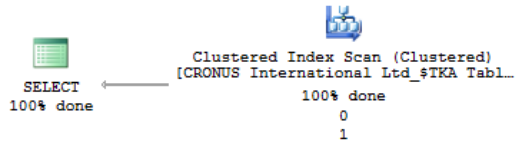
To get non-index value that is included in the key -> Index **SEEK**

(all other fields still cause Index Scan)

```
-- Amount included in key PostingDate,GLAccountName
-- Only Included column
SELECT
     [Amount]
FROM [default].[dbo].[CRONUS International Ltd_$TKA Table Example$08292c4a-b44a-4cc6-8993-1558a379a059]
WHERE
    [Posting Date] >= GETDATE()

-- Included column & non included column
SELECT
     [Amount],[Unique Identifier]
FROM [default].[dbo].[CRONUS International Ltd_$TKA Table Example$08292c4a-b44a-4cc6-8993-1558a379a059]
WHERE
    [Posting Date] >= GETDATE()
```



# Pass-by-value & Pass-by-reference

# Pass-by-

- **Value**

  procedure DoSum(Amount: Decimal): Decimal

  - Passing only values (copy) of original variable
  - Changes done to the value of the variable are visible in the procedure only
  - For record variables, only current record is available. Filters are not passed!

- **Reference**

  procedure DoSum(var Amount: Decimal): Decimal

  - Passing the variable itself
  - Any change done to the value of the variable in the procedure is visible from the caller
  - Everything is passed with the source variable including applied filters

# Pass-by-

- **Value**

  procedure DoSum(Amount: Decimal): Decimal

- **Reference**

  procedure DoSum(Amount: Decimal) Results: Decimal

```
0 references
local procedure MyProcedure(Amount: Decimal): Decimal
var
    Results: Decimal;
begin
    Results := Amount * 0.75;
    exit(Results);
end;
```

```
0 references
local procedure MyProcedure(Amount: Decimal) Results: Decimal
begin
    Results := Amount * 0.75;
end;
```

# Partial Records

# Partial Records

**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 subset of all fields is loaded (primary keys, system & audit fields and defined fields)
  - Non-loaded fields are loaded automatically using the Just-In-Time (JIT) mechanism

```
1 reference
local procedure DoPartialRecordsSumAmountField(): Decimal
var
    GLEntry: Record "G/L Entry";
    AmountSum: Decimal;
begin
    GLEntry.SetLoadFields(Amount);
    GLEntry.FindSet();
    repeat
        AmountSum += GLEntry.Amount;
    until GLEntry.Next() < 1;
    exit(AmountSum);
end;
```

# Partial Records

**[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

- **Task**
  - Sum the Amount field in G/L Entries using loop
  - Standard approach
  - SQL
    - All fields from G/L Entry table are loaded

Standard approach

```
1 reference
local procedure DoNormalSumAmountField(): Decimal
var ···
begin
    ...
    GLEntry.FindSet();
    repeat
        AmountSum += GLEntry.Amount;
        ...
    until GLEntry.Next() < 1;
    ...
    exit(AmountSum);
end;
```

(optimal solution for this task is CalcSums ☺ not looping through all records…).

# Partial Records

- Using SetLoadFields we can specify fields we want to use.
- Other fields will not be loaded. If used, the JIT must load them on-fly (expensive operation)

Partial Records

```
1 reference
local procedure DoPartialRecordsSumAmountField(): Decimal
var ...
begin
    ...
    GLEntry.SetLoadFields(Amount);
    GLEntry.FindSet();
    repeat
        AmountSum += GLEntry.Amount;
        ...
    until GLEntry.Next() < 1;
    ...
    exit(AmountSum);
end;
```

Specifies we want the value of Amount field only

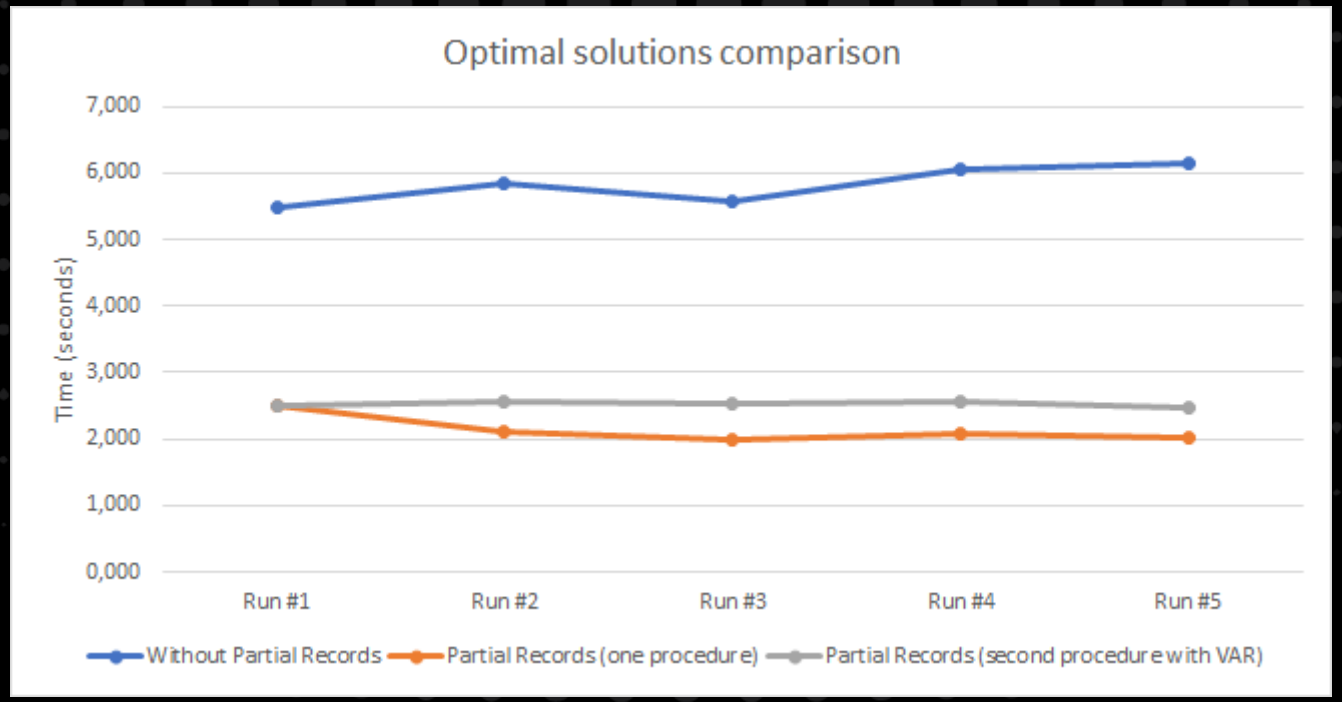
# Examples

See GitHub https://github.com/TKapitan/

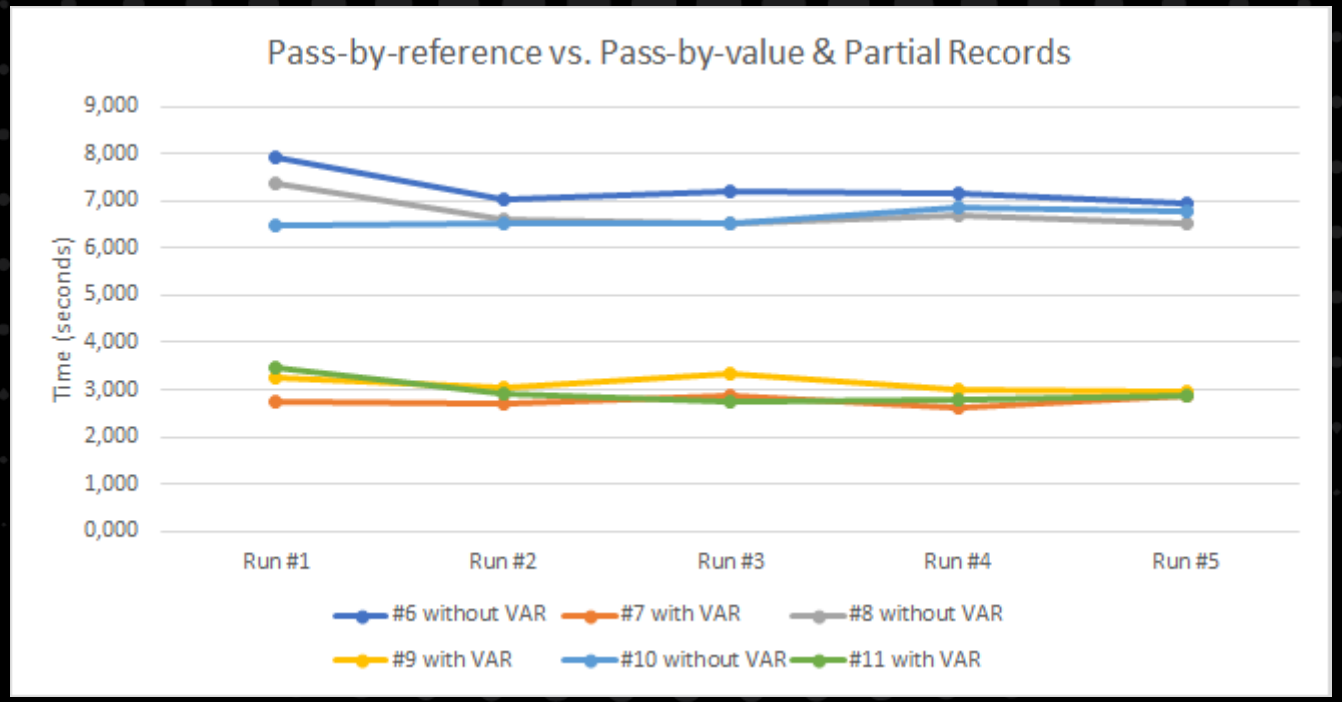
# Partial Records

| # | Implementation | Avg time (5 runs) | Optimal |
|---|---|---|---|
| 1 | Without partial records (standard approach) | 5 sec, 821 millisecs | ✓ |
| 2 | Partial recs, loaded field, one procedure | **2 sec, 048 millisecs** | ✓ |
| 3 | Partial recs, loaded & unloaded field, one procedure | 2 sec, 162 millisecs | ✗ |
| 4 | Partial recs, loaded field, second proc. without VAR | 3 sec, 520 millisecs | ✗ |
| 5 | Partial recs, loaded field, second proc. with VAR | 2 sec, 523 millisecs | ✓ |
| 6 | Partial recs, unloaded field, second proc. without VAR | 7 sec, 240 millisecs | ✗ |
| 7 | Partial recs, unloaded field, second proc. with VAR | 2 sec, 760 millisecs | ✗ |
| 8 | Partial recs, unloaded field, second proc. without VAR, Load | 6 sec, 733 millisecs | ✗ |
| 9 | Partial recs, unloaded field, second proc. with VAR, Load | 2 sec, 571 millisecs | ✗ |
| 10 | Partial recs, unloaded field, second proc. without VAR, SetLoad | 6 sec, 623 millisecs | ✗ |
| 11 | Partial recs, unloaded field, second proc. with VAR, SetLoad | 2 sec, 960 millisecs | ✗ |

# Partial Records



# Partial Records



# Q&A