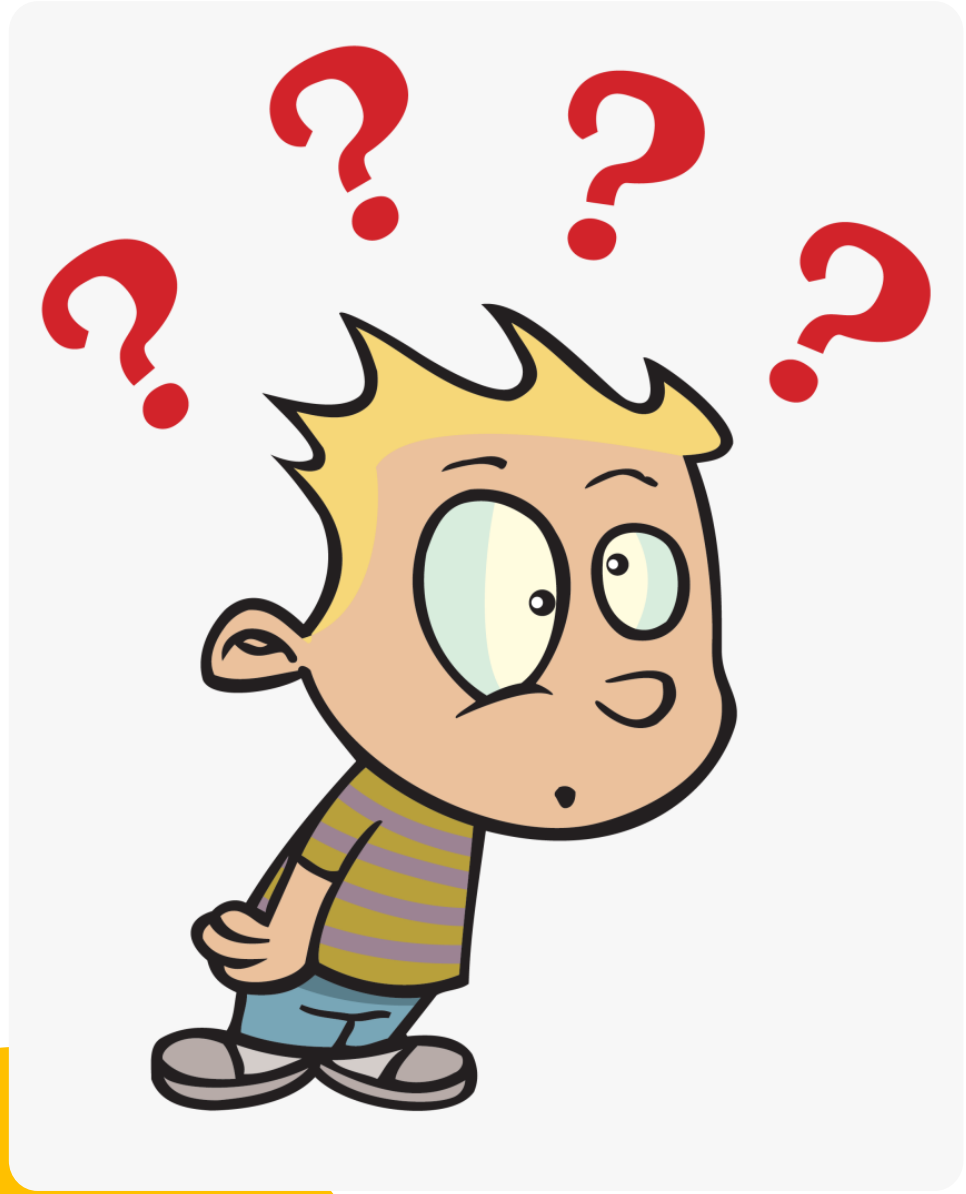
Language Enhancements In



Who am I

- Steen Dybboe
- Database Developer
- Pragmatic BI
- Plays Golf
- No dogs

Language Enhancements SQL Server 2019

```
SELECT APPROX_COUNT_DISTINCT ( id )
FROM dbo.PoolEntrance
```

Pragmatic BI

Lots of Changes in SQL Server 2022

Lots of information from the Grumpy old men

- Aaron Bertrand
- Itzik Ben-Gan

Database compatibility level set to 160

```
Alter Database Current
    Set Compatibility_Level = 160
```

IS [NOT] DISTINCT FROM

Compares the equality of two expressions and guarantees a true or false result, even if one or both operands are NULL.

Predicate used in

- search condition of WHERE clauses
- HAVING clauses
- the join conditions of FROM clauses
- other constructs where a Boolean value is required.

A	B	A = B	A IS NOT DISTINCT FROM B
0	0	True	True
0	1	False	False
0	NULL	Unknown	False
NULL	NULL	Unknown	True

Pragmatic BI

IS [NOT] DISTINCT FROM

```
SELECT X, Y
, Old = IIF(X <> Y, 'Yes', 'No' )
, New = IIF(X IS DISTINCT FROM Y, 'Yes', 'No' )
FROM ( VALUES
   (null, 'abc')
, ('abc', null)
, ('abc', 'abc')
, (null, null)
   ) t(X, Y)
```

	X	Y	Old	New
1	NULL	abc	No	Yes
2	abc	NULL	No	Yes
3	abc	abc	No	No
4	NULL	NULL	No	No

Pragmatic BI

GENERATE_SERIES()

Produce a set-based sequence of numeric values.

It supplants

- Cumbersome ***numbers*** tables
- Recursive CTEs
- On-the-fly sequence generation techniques

GENERATE_SERIES()

Methods through the time (code to remove from your databases)

```
SELECT TOP (1234) ROW_NUMBER() OVER (ORDER BY t1.number) n
FROM master..spt_values t1
CROSS JOIN master..spt_values t2
```

```
;WITH cteNumbers AS (
    SELECT 1 n
    UNION ALL
    SELECT n + 1
    FROM cteNumbers
    WHERE n < 1234
)
SELECT n
FROM cteNumbers
OPTION (MAXRECURSION 0)
```

New Method

```
SELECT *
FROM GENERATE_SERIES(1, 1234)
```

```
SELECT *
FROM GENERATE_SERIES(1, 240, 12)
```

```
SELECT ExplodedDate = DATEADD(DAY,value,'2020-01-01’)
FROM GENERATE_SERIES(0,999)
```

GREATEST / LEAST

- Basically MAX and MIN, but across columns instead of across rows.
- If only two attributes – IIF / CASE is simple
- More than 2, then !!!!

```
SELECT
  GREATEST(1, 5)
, GREATEST(6, 2)
, LEAST (1, 5)
, LEAST (6, 2)
, GREATEST(15, 800, 100)
```

```
SELECT MaxVal =
  CASE
    WHEN a > b AND a > c THEN a
    WHEN b > a AND b > c THEN b
    WHEN c > a AND c > b THEN c
  END
FROM (
  VALUES (11,7,2),(15, 800, 100)
) t(a,b,c)
```

Pragmatic BI

TRIM() – Extended functionality

LTRIM (character_expression , [characters])

RTRIM (character_expression , [characters])

TRIM ([LEADING | TRAILING | BOTH] [characters FROM] string)

```
SELECT RTRIM('x123abcx','abcx'); --> x123

SELECT LTRIM('123abcx', '21x'); --> 3abcx

SELECT TRIM('12x' FROM '12xaxbc13x')
                                --> axbc13
```

```
SELECT TRIM(LEADING '123x'  FROM '123abcx')

SELECT TRIM(TRAILING 'abcx' FROM 'x123abcx')

SELECT TRIM(BOTH '12x' FROM '12xaxbc12x’)
```

STRING_SPLIT

Documentation (old):
The output rows might be in any order. The order **is not guaranteed** to match the order of the substrings in the input string.

STRING_SPLIT has been in SQL Server for a few versions, but you had no guarantee of ordering

We need some way to reliably determine the original sequence

```
STRING_SPLIT ( string , separator [ , enable_ordinal ] )
```

```
SELECT *
FROM string_split('bbb,aaa,ccc',',')

SELECT *
FROM string_split('bbb,aaa,ccc',',',1)
```

	value
1	bbb
2	aaa
3	ccc

	value	ordinal
1	bbb	1
2	aaa	2
3	ccc	3

Pragmatic BI

STRING_SPLIT

Bravo/Alpha/Bravo/Tango/Delta/Bravo/Alpha/Delta

Return only first occurrence in same order

```
DECLARE @Delim nchar(1) = N'/';
DECLARE @List nvarchar(max) =
N'Bravo/Alpha/Bravo/Tango/Delta/Bravo/Alpha/Delta';
```

```
SELECT value, ordinal
FROM STRING_SPLIT(@List, @Delim, 1)
```

	value	ordinal
1	Bravo	1
2	Alpha	2
3	Bravo	3
4	Tango	4
5	Delta	5
6	Bravo	6
7	Alpha	7
8	Delta	8

```
SELECT value, ordinal = MIN(ordinal)
FROM STRING_SPLIT(@List, @Delim, 1)
GROUP BY value
```

	value	ordinal
1	Alpha	2
2	Bravo	1
3	Delta	5
4	Tango	4

```
SELECT STRING_AGG(value, N'/') WITHIN GROUP (ORDER BY ordinal)
FROM (
    SELECT value, ordinal = MIN(ordinal)
    FROM STRING_SPLIT(@List, @Delim, 1)
    GROUP BY value
) t
```

	(No column name)
1	Bravo/Alpha/Tango/Delta

Pragmatic BI

DATE_BUCKET()

Collapses a date/time to a fixed interval, eliminating the need to

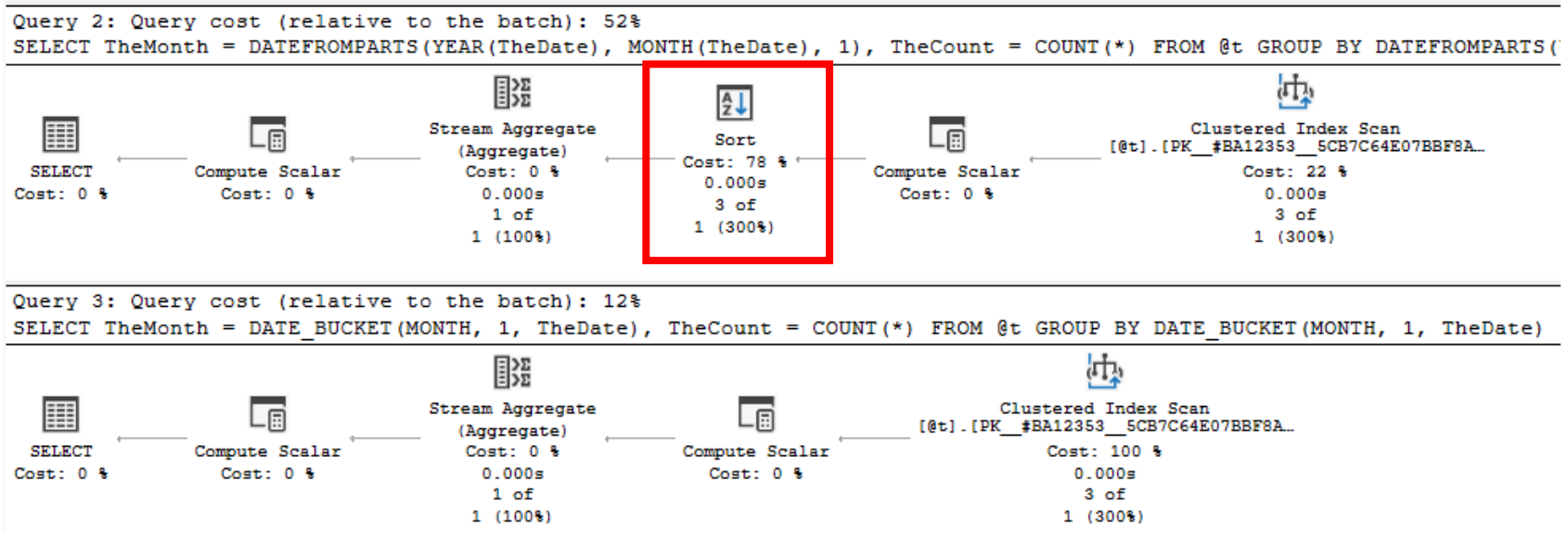
- round datetime values,
- extract date parts,
- perform wild conversions to and from other types like float

  (to make elaborate and unintuitive dateadd/datediff calculations - sometimes using magic dates from the past).

```
SELECT name, modify_date,
  ModMonth1 = DATEADD(MONTH, DATEDIFF(MONTH, '19000101', modify_date), '19000101')

, ModMonth2 = DATEFROMPARTS(YEAR(modify_date), MONTH(modify_date), 1)

, ModMonth3 = DATE_BUCKET(MONTH, 1, modify_date)

FROM sys.all_objects;
```

Date Bucket is fast – Preserves SortOrder

```
DECLARE @t table(dt date PRIMARY KEY);
INSERT @t(dt) VALUES('20220701'),('20220702'),('20220703');

SELECT m = DATEFROMPARTS(YEAR(dt), MONTH(dt), 1), cnt = COUNT(*)
FROM @t
GROUP BY DATEFROMPARTS(YEAR(dt), MONTH(dt), 1);

SELECT m = DATE_BUCKET(MONTH, 1, dt), cnt = COUNT(*)
FROM @t
GROUP BY DATE_BUCKET(MONTH, 1, dt);
```



Pragmatic BI

DATE_BUCKET

- Userdefined Bucket Sizes

```
SELECT dt
, BucketDateNew = DATE_BUCKET(MINUTE, 10, dt)
, BucketDateOld = DATEADD(MINUTE, ((DATEDIFF(MINUTE, 0, dt) / 10) * 10), 0)
FROM (
    SELECT dt = DATEADD(MINUTE, s.value, '2023-12-01 07:00')
    FROM GENERATE_SERIES(1, 10080) s
) t
```

```
SELECT BucketDate = DATE_BUCKET(MINUTE, 10, dt), cnt = COUNT(*)
FROM (
    SELECT dt = DATEADD(MINUTE, s.value, '2023-12-01 07:00')
    FROM GENERATE_SERIES(1, 10080) s
) t
GROUP BY DATE_BUCKET(MINUTE, 10, dt)
ORDER BY BucketDate
```

DATETRUNC()

Truncate date to desired fraction

The returned data type for DATETRUNC is dynamic!

DATETRUNC returns a truncated date of the same data type (and, if applicable, the same fractional time scale) as the input date.

- If DATETRUNC is given a datetimeoffset(3) input date, DATETRUNC will return a datetimeoffset(3).

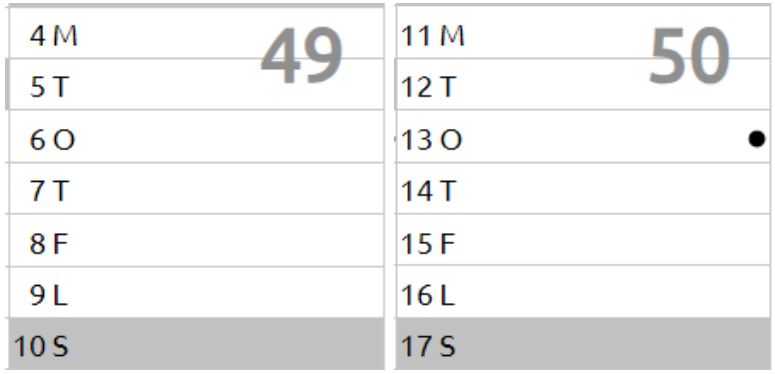
- If it is given a string literal that could resolve to a datetime2(7), DATETRUNC will return a datetime2(7).

DATETRUNC()

```
SELECT t.dt
  , year        = DATETRUNC(YEAR       , t.dt)
  , quarter     = DATETRUNC(QUARTER    , t.dt)
  , month       = DATETRUNC(MONTH      , t.dt)
  , dayofyear   = DATETRUNC(DAYOFYEAR  , t.dt)
  , day         = DATETRUNC(DAY        , t.dt)
  , week        = DATETRUNC(WEEK       , t.dt)
  , iso_week    = DATETRUNC(ISO_WEEK   , t.dt)
  , hour        = DATETRUNC(HOUR       , t.dt)
  , minute      = DATETRUNC(MINUTE     , t.dt)
  , second      = DATETRUNC(SECOND     , t.dt)
  , millisecond = DATETRUNC(MILLISECOND, t.dt)
FROM (
    SELECT dt = DATEADD(MILLISECOND,
                (ABS(CAST(CAST(NEWID() AS VARBINARY) AS INT)) % 1296000000)
                , '2023-12-01 07:00')
    FROM GENERATE_SERIES(1, 30) s
) t
```

Pragmatic BI

DATETRUNC()



Be carefull when using @@DATEFIRST

```
SELECT DATE_FIRST = @@DATEFIRST
DECLARE @d datetime2
SET @d = '2023-12-12 12:12:12.1234567';

SET DATEFIRST 1;
SELECT 'Week1', Day1 = DATETRUNC(week, @d);

SET DATEFIRST 7;
SELECT 'Week7', Day1 = DATETRUNC(week, @d);

SET DATEFIRST 6;
SELECT 'Week6', Day1 = DATETRUNC(week, @d);
```

	DATEFIRST
1	1

	(No column name)	Day1
1	Week1	2023-12-11 00:00:00.0000000

	(No column name)	Day1
1	Week7	2023-12-10 00:00:00.0000000

	(No column name)	Day1
1	Week6	2023-12-09 00:00:00.0000000

The WINDOW Clause

- Goal is simplification
- Reuse parts of—or entire—window definitions with the WINDOW clause
- New SQL WORD

```
WINDOW window_name AS (
                        [ reference_window_name ]
                        [ <window partition clause> ]
                        [ <window order clause> ]
                        [ <window frame clause> ]
                    )
```

SELECT
FROM
WHERE
GROUP BY
HAVING
WINDOW
ORDER BY

Pragmatic BI

T-SQL Windowing Improvements in SQL Server 2022 - SQLPerformance.com

Window Clause

```
SELECT CustomerID, OrderID, OrderDate, Quantity, SumTotal
, RunQty = SUM(Quantity) OVER (PARTITION BY CustomerID
                               ORDER BY OrderDate, OrderID
                               ROWS UNBOUNDED PRECEDING)
, SumTotal
, RunTot = SUM(SumTotal) OVER (PARTITION BY CustomerID
                               ORDER BY OrderDate, OrderID
                               ROWS UNBOUNDED PRECEDING)
FROM dbo.SomeSales
```

```
SELECT CustomerID, OrderID, OrderDate, Quantity
, RunQty = SUM(Quantity) OVER win
, SumTotal
, RunTot = SUM(SumTotal) OVER win
FROM dbo.SomeSales
WINDOW win AS (PARTITION BY CustomerID
               ORDER BY OrderDate, OrderID
               ROWS UNBOUNDED PRECEDING)
```

Pragmatic BI

WINDOW Clause

Mixed Usages of Window definitions

```
SELECT CustomerID, OrderID, OrderDate,
Quantity, SumTotal
, RunQty = SUM(Quantity) OVER win
, RunTot = SUM(SumTotal) OVER win
, RowNum = RANK() OVER (ORDER BY OrderDate)
FROM dbo.SomeSales
WINDOW win AS (PARTITION BY CustomerID
               ORDER BY OrderDate, OrderID
               ROWS UNBOUNDED PRECEDING)
```

Cross referencing Window clauses

```
SELECT CustomerID, OrderID, OrderDate
, Quantity , SumTotal
, RunQty = SUM(Quantity) OVER winF
, RunTot = SUM(SumTotal) OVER winF
, RowNum = ROW_NUMBER()  OVER winO
FROM dbo.SomeSales
WINDOW
  winF AS (winO ROWS UNBOUNDED PRECEDING)
, winO AS (winP ORDER BY OrderDate, OrderID)
, winP AS (PARTITION BY CustomerID)
```

Pragmatic BI

WINDOW Clause – The missing part

SELECT
FROM
WHERE
GROUP BY
HAVING
WINDOW
ORDER BY

SELECT

FROM
WHERE

GROUP BY
HAVING

WINDOW

ORDER BY

Pragmatic BI

Window functions Updated

NULL treatment clause available to window functions
FIRST_VALUE, LAST_VALUE, LAG and LEAD

Syntax

```
<function>(<scalar_expression>[, <other args>]) [IGNORE NULLS | RESPECT NULLS] OVER( <specification> )
```

RESPECT NULLS option is the default

IGNORE NULLS option – **Long-awaited!**

Pragmatic BI

LAG – With IGNORE NULLS

```
SELECT id, col1,
    Prev      = LAG(col1) OVER ( ORDER BY id )
  , PrevKnown = LAG(col1) IGNORE NULLS OVER ( ORDER BY id )
FROM dbo.T1;
```

```
INSERT INTO dbo.T1(id, col1, col2)
VALUES
  ( 2, NULL,  200),
  ( 3,   10, NULL),
  ( 5,   -1, NULL),
  ( 7, NULL,  202),
  (11, NULL,  150),
  (13,  -12,   50),
  (17, NULL,  180),
  (19, NULL,  170),
  (23, 1759, NULL);
```

	id	col1	Prev	PrevKnown
1	2	NULL	NULL	NULL
2	3	10	NULL	NULL
3	5	-1	10	10
4	7	NULL	-1	-1
5	11	NULL	NULL	-1
6	13	-12	NULL	-1
7	17	NULL	-12	-12
8	19	NULL	NULL	-12
9	23	1759	NULL	-12

Pragmatic BI

Last known (non-null) value – ”Fill the gaps”

```
INSERT INTO dbo.T1(id, col1, col2)
VALUES
  ( 2, NULL,  200),
  ( 3,   10, NULL),
  ( 5,   -1, NULL),
  ( 7, NULL,  202),
  (11, NULL,  150),
  (13,  -12,   50),
  (17, NULL,  180),
  (19, NULL,  170),
  (23, 1759, NULL);
```

	id	col1	lastknowncol1
1	2	NULL	NULL
2	3	10	10
3	5	-1	-1
4	7	NULL	-1
5	11	NULL	-1
6	13	-12	-12
7	17	NULL	-12
8	19	NULL	-12
9	23	1759	1759

```
;WITH MyCTE AS (
  SELECT id, col1
  , grp = MAX(CASE WHEN col1 IS NOT NULL THEN id END)
          OVER (ORDER BY id ROWS UNBOUNDED PRECEDING)
  FROM dbo.T1
)
SELECT id, col1
, lastknowncol1 = MAX(col1)
                  OVER (PARTITION BY grp
                        ORDER BY id
                        ROWS UNBOUNDED PRECEDING)
FROM MyCTE;
```


```
SELECT id, col1
, lastknowncol1 = LAST_VALUE(col1) IGNORE NULLS
        OVER( ORDER BY id ROWS UNBOUNDED PRECEDING )
FROM dbo.T1;
```

Pragmatic BI

Fill the Gaps – Two Columns

```
WITH C AS
(
  SELECT id, col1, col2,
    MAX(CASE WHEN col1 IS NOT NULL THEN id END)
      OVER(ORDER BY id
           ROWS UNBOUNDED PRECEDING) AS grp1,
    MAX(CASE WHEN col2 IS NOT NULL THEN id END)
      OVER(ORDER BY id
           ROWS UNBOUNDED PRECEDING) AS grp2
  FROM dbo.T1
)
SELECT id,
  col1,
  Last1 = MAX(col1) OVER(PARTITION BY grp1
                  ORDER BY id
                  ROWS UNBOUNDED PRECEDING),
  col2,
  Last2 = MAX(col2) OVER(PARTITION BY grp2
                  ORDER BY id
                  ROWS UNBOUNDED PRECEDING)
FROM C;
```

```
SELECT id, col1, col2
, Last1 = LAST_VALUE(col1) IGNORE NULLS OVER W
, Last2 = LAST_VALUE(col2) IGNORE NULLS OVER W
FROM dbo.T1
WINDOW W AS ( ORDER BY id ROWS UNBOUNDED PRECEDING );
```

	id	col1	col2	Last1	Last2
1	2	NULL	200	NULL	200
2	3	10	NULL	10	200
3	5	-1	NULL	-1	200
4	7	NULL	202	-1	202
5	11	NULL	150	-1	150
6	13	-12	50	-12	50
7	17	NULL	180	-12	180
8	19	NULL	170	-12	170
9	23	1759	NULL	1759	170

Bit Manipulation

- **New Functions**
  - **BIT_COUNT ()**
  - **GET_BIT ()**
  - **SET_BIT ()**
  - **LEFT_SHIFT ()**
  - **RIGHT_SHIFT ()**
- **Operates on:**
  tinyint, smallint, int, bigint,
  binary(*n*), varbinary(*n*)

BIT Manipulation - Usage

- Binary manipulation
- Ex. used in SET variables
  - @@options
- Extract using & operator

```
SELECT @@OPTIONS & 512  --> 512
```

```
SELECT GET_BIT(@@OPTIONS, 9) --> 1
```

```
   DECLARE @options INT = @@OPTIONS
   PRINT @options
0  IF ( (1 & @options) = 1 ) PRINT 'DISABLE_DEF_CNST_CHK'
1  IF ( (2 & @options) = 2 ) PRINT 'IMPLICIT_TRANSACTIONS'
2  IF ( (4 & @options) = 4 ) PRINT 'CURSOR_CLOSE_ON_COMMIT'
3  IF ( (8 & @options) = 8 ) PRINT 'ANSI_WARNINGS'
4  IF ( (16 & @options) = 16 ) PRINT 'ANSI_PADDING'
5  IF ( (32 & @options) = 32 ) PRINT 'ANSI_NULLS'
6  IF ( (64 & @options) = 64 ) PRINT 'ARITHABORT'
7  IF ( (128 & @options) = 128 ) PRINT 'ARITHIGNORE'
8  IF ( (256 & @options) = 256 ) PRINT 'QUOTED_IDENTIFIER'
9  IF ( (512 & @options) = 512 ) PRINT 'NOCOUNT'
   IF ( (1024 & @options) = 1024 ) PRINT 'ANSI_NULL_DFLT_ON'
   IF ( (2048 & @options) = 2048 ) PRINT 'ANSI_NULL_DFLT_OFF'
   IF ( (4096 & @options) = 4096 ) PRINT 'CONCAT_NULL_YIELDS_NULL'
   IF ( (8192 & @options) = 8192 ) PRINT 'NUMERIC_ROUNDABORT'
   IF ( (16384 & @options) = 16384 ) PRINT 'XACT_ABORT'
```

```
SELECT @@OPTIONS --> 6008 -->   0001 1110 1110 1000
```

Pragmatic BI

BIT Manipulation

40

```
SET_BIT(@val, 3)
SET_BIT(@val, 5)

SELECT BIT_COUNT(@val)   -->  2
SELECT GET_BIT(@val, 4)  -->  0
SELECT GET_BIT(@val, 5)  -->  1
```

80

```
SELECT LEFT_SHIFT(40, 1)  -->  80

SELECT LEFT_SHIFT(40, 2)  --> 160
SELECT LEFT_SHIFT(40, 3)  --> 320

SELECT RIGHT_SHIFT(40, 1)  --> 20
SELECT RIGHT_SHIFT(40, 2)  --> 10  25% !!
SELECT RIGHT_SHIFT(40, 3)  -->  5
```

Pragmatic BI

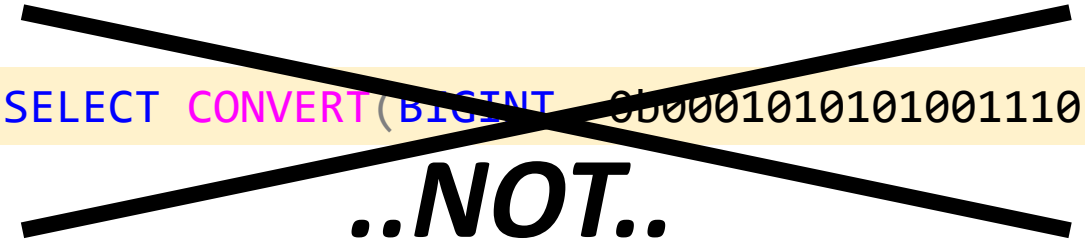
BinaryParameters

Passing explicit Hexadecimal values can be done with

```
SELECT CONVERT(BIGINT, 0x210DB12E)
```

Passing binary values can now be done with

```
SELECT CONVERT(BIGINT, 0b0001010101001110)
```



Pragmatic BI

JSON functions

- ISJSON ()
- JSON_PATH_EXISTS ()
- JSON_OBJECT ()
- JSON_ARRAY ()

JSON functions – ISJSON()

Added functionality

New type constraint – define expected type JSON part

VALUE	Tests for a valid JSON value. This can be a JSON object, array, number, string or one of the three literal values (false, true, null)
ARRAY	Tests for a valid JSON array
OBJECT	Tests for a valid JSON object
SCALAR	Tests for a valid JSON scalar – number or string

```
DECLARE @json NVARCHAR(MAX) = N'["Michael", "Kobe", "LeBron", "Magic"]';

SELECT IsValid = ISJSON(@json, ARRAY)
```

Pragmatic BI

SQL Server ISJSON for valid Value, Array or Object JSON format (mssqltips.com)

JSON functions – JSON_PATH_EXISTS ()

New function in 2022

Determine if a specific path exists in a JSON document

```
{
"Name":"Rhaenyra",
"Lastname":"Targaryen",
"Children":[
   {
     "name":"Jacaerys",
     "lastname":"Velaryon",
     "age":20
   },
   {
     "name":"Lucerys",
     "lastname":"Velaryon",
     "age":18
   }
 ]
}
```

```
SELECT JSON_PATH_EXISTS(@jsonsample,'$.name')
```
→ **0**

```
SELECT JSON_PATH_EXISTS(@jsonsample,'$.Name')
```
→ **1**

```
SELECT JSON_PATH_EXISTS(@jsonsample,'$.Children[0].name')
```
→ **1**

```
SELECT JSON_PATH_EXISTS(@jsonsample,'$..Name')
```

```
Msg 13607, Level 16, State 4, Line 91
JSON path is not properly formatted. Unexpected
character '.' is found at position 2.
```

Pragmatic BI

JSON Functions - JSON_OBJECT

- Constructs JSON Objects

```
SELECT JSON_OBJECT('Firstname':FirstName,'Lastname':LastName,'Time':getdate())
FROM [Person].[Person]
```

attribute
{"Firstname":"Syed","Lastname":"Abbas","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Catherine","Lastname":"Abel","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Kim","Lastname":"Abercrombie","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Kim","Lastname":"Abercrombie","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Kim","Lastname":"Abercrombie","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Hazem","Lastname":"Abolrous","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Sam","Lastname":"Abolrous","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Humberto","Lastname":"Acevedo","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Gustavo","Lastname":"Achong","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Pilar","Lastname":"Ackerman","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Pilar","Lastname":"Ackerman","Time":"2022-11-17T09:46:24.983"}
{"Firstname":"Aaron","Lastname":"Adams","Time":"2022-11-17T09:46:24.983"}

JSON Functions - JSON_ARRAY()

- Constructs JSON Array element

```
SELECT JSON_ARRAY
  (JSON_OBJECT('name':'Robert','lastname':'Downey','age':57),
  (JSON_OBJECT('name':'Chris','lastname':'Hemsworth','age':39)))
```

```
[
  {
    "name":"Robert",
    "lastname":"Downey",
    "age":57
  },
  {
    "name":"Chris",
    "lastname":"Hemsworth",
    "age":39
  }
]
```

Pragmatic BI

Resumable Operations

- Pause and Resume
  - online primary and unique key creation
  - online index creation/rebuild

- Syntax
  - ALTER INDEX.
  - ALTER TABLE (while creating constraints)

```
ALTER TABLE sales.Invoice
    ADD CONSTRAINT PK_Invoice PRIMARY KEY CLUSTERED (InvoiceID)
    WITH (ONLINE = ON, MAXDOP = 2, RESUMABLE = ON, MAX_DURATION = 240);
```

Resumable Operations – In action

Explicit Pause and Resume. Uses ALL syntax

```
ALTER INDEX ALL ON sales.Invoice PAUSE;

ALTER INDEX ALL ON sales.Invoice RESUME;
```

When a PAUSE is executed, an ERROR (!) is thrown

Don´t worry

```
Msg 1219, Level 16, State 1, Line 711
Your session has been disconnected because of a high priority DDL operation.
Msg 1219, Level 16, State 1, Line 711
Your session has been disconnected because of a high priority DDL operation.
Msg 1750, Level 16, State 1, Line 711
Could not create constraint or index. See previous errors.
Msg 596, Level 21, State 1, Line 710
Cannot continue the execution because the session is in the kill state.
Msg 0, Level 20, State 0, Line 710
A severe error occurred on the current command.
The results, if any, should be discarded.
```

Progres and state may be validated with

```
SELECT sql_text, state_desc, percent_complete
FROM sys.index_resumable_operations;
```

	sql_text	state_desc	percent_complete
1	ALTER TABLE Warehouse.ARC_ColdRoomTemperatur...	PAUSED	28,7298453294575

Pragmatic BI

CREATE INDEX

- WAIT_AT_LOW_PRIORITY with online index operations clause added
- MAX_DURATION - is used to specify the maximum time in minutes to wait until an action
  - ABORT_AFTER_WAIT - is used to abort the operation if it exceeds the wait time
    - NONE - is to wait for the lock with regular priority
    - SELF - exits the online index operation
    - BLOCKERS - kills transactions blocking the index rebuild

```
CREATE CLUSTERED INDEX cindex
ON  dbo.salesOrderDetailTest (SalesOrderDetailID)
WITH (ONLINE = ON
        (WAIT_AT_LOW_PRIORITY
           (MAX_DURATION = 50 MINUTES, ABORT_AFTER_WAIT = SELF)
        )
     );
```

Pragmatic BI

CREATE STATISTICS

- Adds AUTO_DROP option
- Manually-created statistics may prevent schema changes being applied correctly.
  - Like dropping a column on a table with manual statistics on that column
- Such statistics can be created (or updated) with the AUTO_DROP

```
CREATE STATISTICS InvoiceStats ON sales.Invoice (CustomerKey, InvoiceDate)
WITH AUTO_DROP = ON;
```

Dynamic Data Masking

UNMASK permission adds more granularity allowing you to grant this permission at the database, schema, table, and even column levels.

```
GRANT UNMASK ON Schema.Table(Column) TO Role;
```

Aggregate functions

Computing accurate percentiles has negative implications in terms of memory footprint and performance

APPROX_PERCENTILE_CONT ()

APPROX_PERCENTILE_DISC ()

These functions can be used as an alternative for large datasets where negligible error with faster response is acceptable as compared to accurate percentile value with slow response time.

Query Store

Procedure to set up query hints for the queries in the Query Store

- *sp_query_store_set_hints*
- *sp_query_store_clear_hints*

- Extensions to Intelligent Query Processing (IQP).
  Note *Query Processing Feedback* which provides feedback for
  - Cardinality estimation
  - Degree of parallelism (DOP)
  - Memory grant
- Query Store must be enabled in read-write mode to take advantage of this feature.

Questions

Links

https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2022

My Favorite T-SQL Enhancements in SQL Server 2022 (Aaron Bertrand):
https://www.mssqltips.com/sqlservertip/7265/sql-server-2022-t-sql-enhancements/

T-SQL Windowing Improvements in SQL Server 2022 (Itzik Ben-Gan):
https://sqlperformance.com/2022/05/t-sql-queries/windowing-improvements-sql-server-2022

Pragmatic BI

Pragmatic BI Aps
Åhusene 4,9,3
8000 Arhus C

Telefon +45 42 606 202
E-mail steen@prabi.dk
Internet www.prabi.dk
CVR nr. 36974133