IDUG
Leading the DB2 User Community since 1988

IDUG DB2 EMEA Tech Conference
Lisbon Portugal | October 2017

#IDUGDB2

DB2 11 Performance: BLU Hits and Misses

Mark Gillis
Triton Consulting

Session code: D15

Date: Wednesday, 5 October 2017

Time: 17:30 – 18:30

DB2 for LUW

1

IDUG
Leading the DB2 User Community since 1988

IDUG DB2 EMEA Tech Conference
Lisbon Portugal | October 2017

#IDUGDB2

Db2 11 Performance: BLU Hits and Misses

Mark Gillis
Triton Consulting

Session code: D15

Date: Wednesday, 5 October 2017

Time: 17:30 – 18:30

Db2

2

Intro

- Mark Gillis – Principal Consultant with Triton Consulting
- Originally DB2 on mainframe (starting in 1990 with v2.1) but mid-range since 2000

IBM Certified Advanced Database Administrator -
IBM Certified Database Administrator - DB2 10.5 f
IBM Certified Designer – Cognos 10 BI Reports
IBM Certified Developer - Cognos 10 BI Metadata
IBM Certified Database Adminstrator - DB2 11.1 f

- http://db2geek.triton.co.uk/

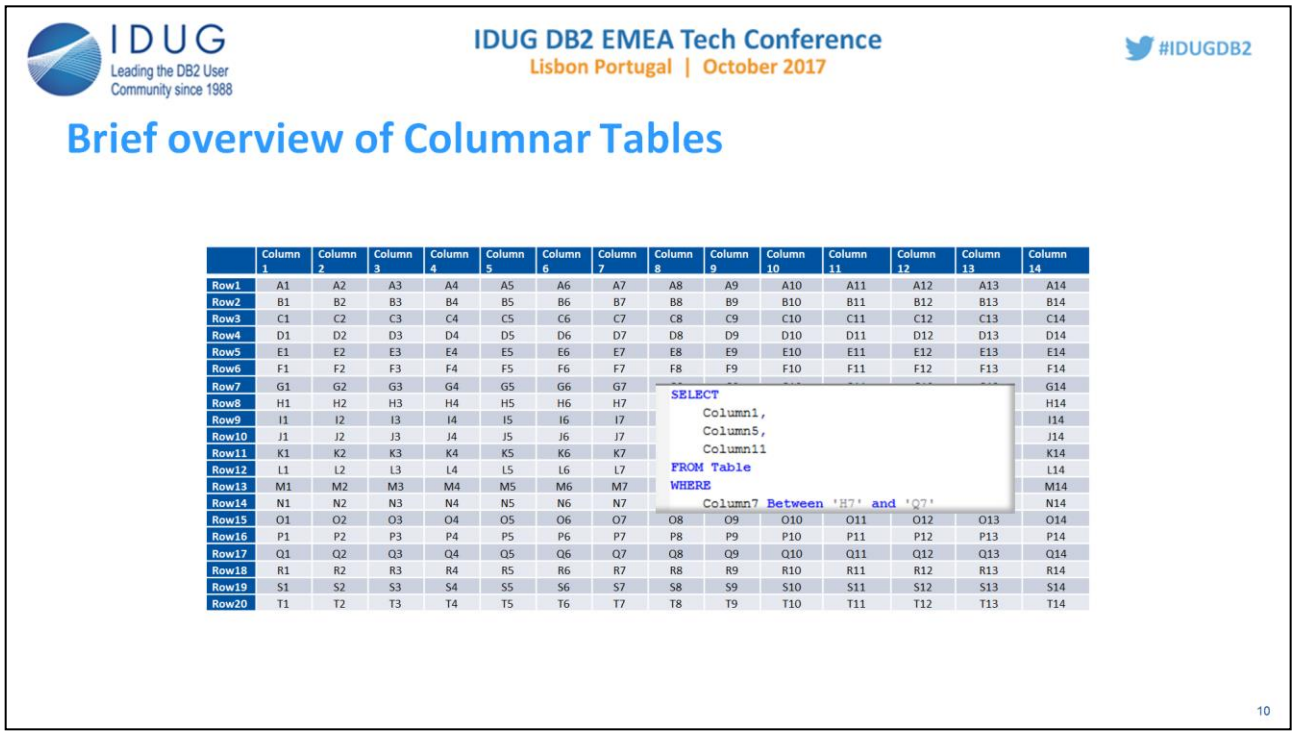
Agenda

- Brief overview of BLU technology
- Examples of workload performance in V10.5
- The benefits of V11.1 for your workload
- Examples of where the savings are
- Gotchas : where you might not get the advertised benefits

That is my advertised agenda, but I might skip about a bit

The idea is to share observations gleaned from experience of using systems that have been upgraded to V11.1

It is not going to present any blinding insights into why it works, or doesn't work



	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12	Column 13	Column 14
Row1	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14
Row2	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14
Row3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
Row4	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
Row5	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
Row6	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
Row7	G1	G2	G3	G4	G5	G6	G7							G14
Row8	H1	H2	H3	H4	H5	H6	H7							H14
Row9	I1	I2	I3	I4	I5	I6	I7							I14
Row10	J1	J2	J3	J4	J5	J6	J7							J14
Row11	K1	K2	K3	K4	K5	K6	K7							K14
Row12	L1	L2	L3	L4	L5	L6	L7							L14
Row13	M1	M2	M3	M4	M5	M6	M7							M14
Row14	N1	N2	N3	N4	N5	N6	N7							N14
Row15	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14
Row16	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
Row17	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Row18	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
Row19	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Row20	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14

The advantage of column-organized tables over row-organized is that far less data needs to be retrieved for a query to be satisfied. This is particularly aimed at OLAP / BI queries where large volumes of data need to be processed to return the result set; often aggregated into more compact, summarized details. With column-organized tables only the pages that contain the columns requested by the query will be retrieved. Data skipping will further reduce the result set by eliminating pages that do not match the predicates from the WHERE clause. And the data will be significantly compressed, both on disk and in memory to compact the data even further.

So, here's a (very) simple illustration of a table and its data

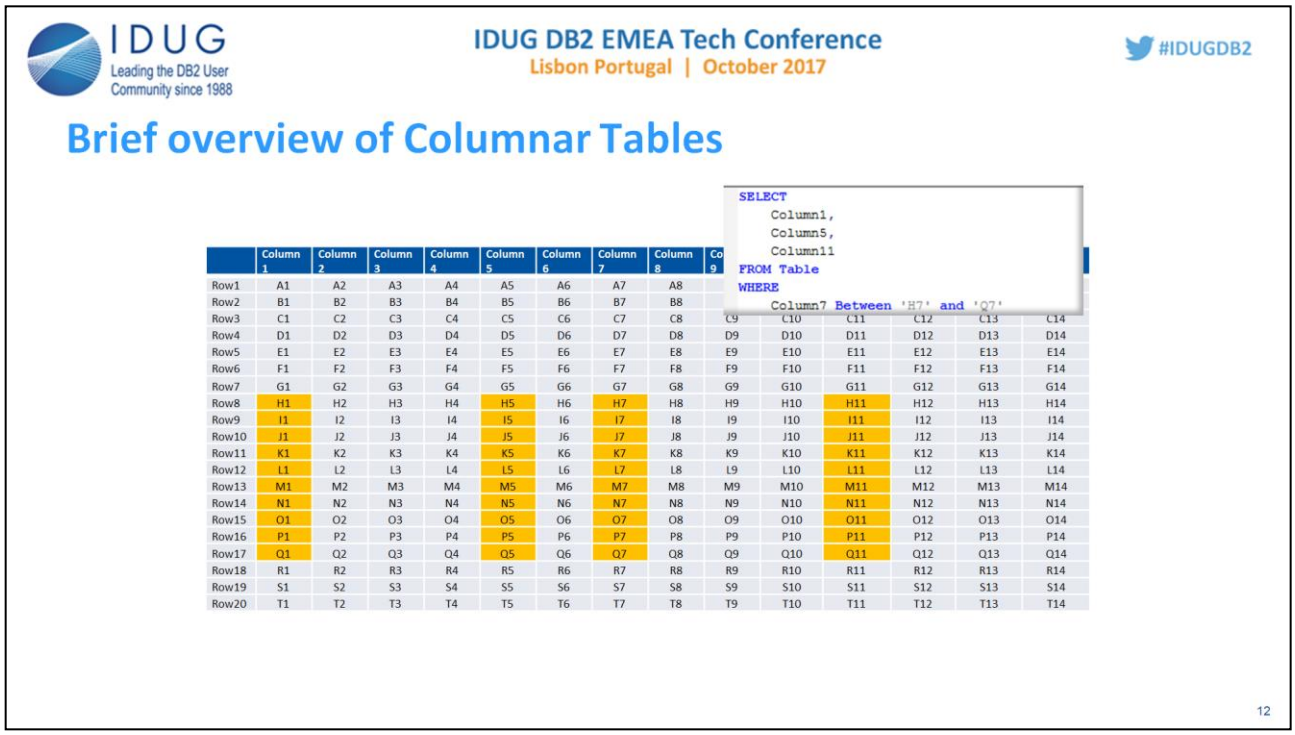
Brief overview of Columnar Tables

```
SELECT
    Column1,
    Column5,
    Column11
FROM Table
WHERE
    Column7 Between 'H7' and 'Q7'
```

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	C 9					
Row1	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14
Row2	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14
Row3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
Row4	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
Row5	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
Row6	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
Row7	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14
Row8	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14
Row9	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
Row10	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	J11	J12	J13	J14
Row11	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14
Row12	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14
Row13	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
Row14	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14
Row15	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14
Row16	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
Row17	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Row18	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
Row19	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Row20	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14

11

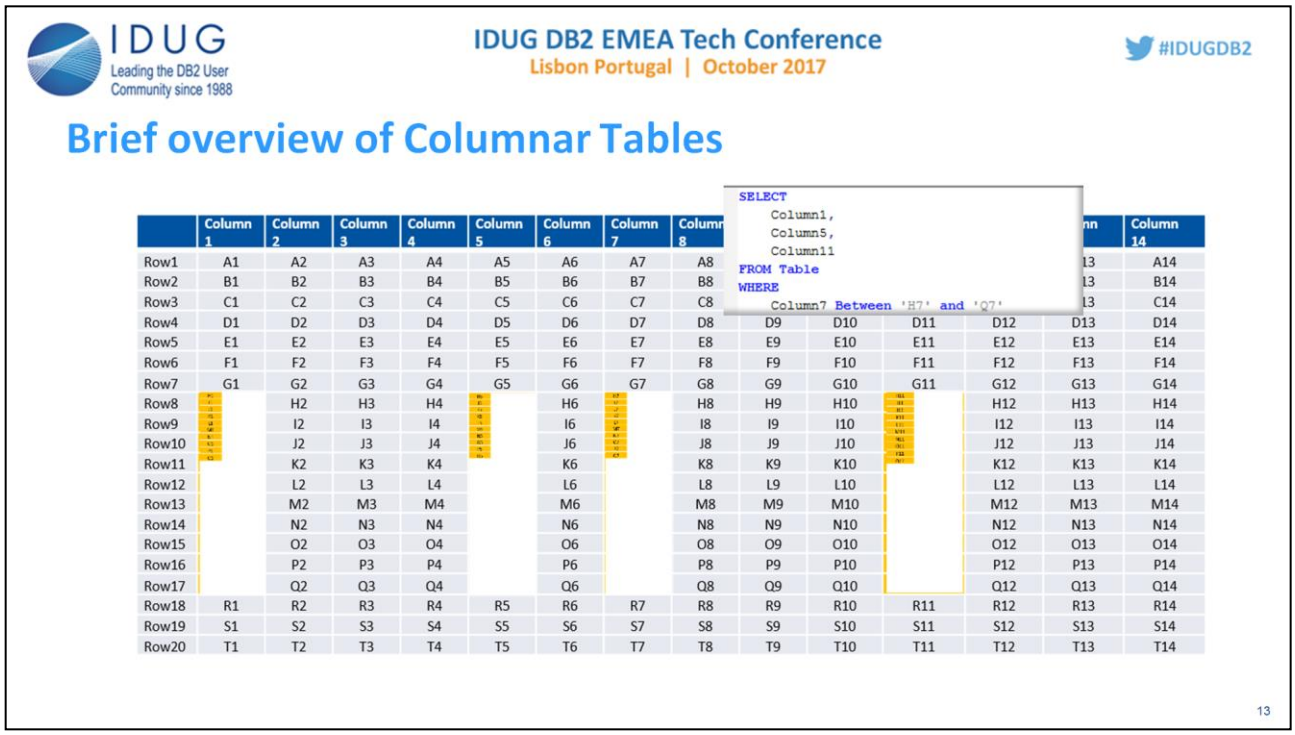
With a row-based table the ‘area’ of data that the query would have to access is shown by the shaded area



	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Co 9	C10	C11	C12	C13	C14
Row1	A1	A2	A3	A4	A5	A6	A7	A8						
Row2	B1	B2	B3	B4	B5	B6	B7	B8						
Row3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
Row4	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
Row5	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
Row6	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
Row7	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14
Row8	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14
Row9	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
Row10	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	J11	J12	J13	J14
Row11	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14
Row12	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14
Row13	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
Row14	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14
Row15	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14
Row16	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
Row17	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Row18	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
Row19	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Row20	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14

But if it was column-organized, the same query would access the shaded area shown

Bearing in mind that besides the reduced number of pages that are being retrieved, the column-organized data is compressed using routines that can offer 10-fold improvements over pre 10.5 algorithms, and that the database engine will be taking advantage of hardware capabilities (e.g. Parallel vector processing, multi-core parallelism and single instruction, multiple data (SIMD) parallelism) to further improve the data processing speed, and you can anticipate very rapid analytical processing of large volumes of data.

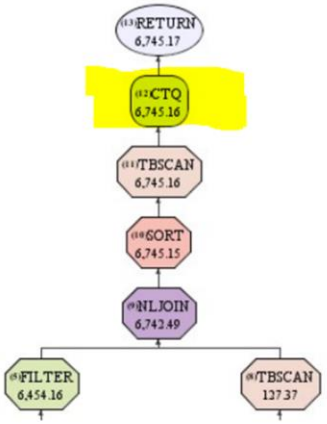


	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8						Column 14
Row1	A1	A2	A3	A4	A5	A6	A7	A8					13	A14
Row2	B1	B2	B3	B4	B5	B6	B7	B8					13	B14
Row3	C1	C2	C3	C4	C5	C6	C7	C8					13	C14
Row4	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
Row5	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
Row6	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
Row7	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14
Row8		H2	H3	H4		H6		H8	H9	H10		H12	H13	H14
Row9		I2	I3	I4		I6		I8	I9	I10		I12	I13	I14
Row10		J2	J3	J4		J6		J8	J9	J10		J12	J13	J14
Row11		K2	K3	K4		K6		K8	K9	K10		K12	K13	K14
Row12		L2	L3	L4		L6		L8	L9	L10		L12	L13	L14
Row13		M2	M3	M4		M6		M8	M9	M10		M12	M13	M14
Row14		N2	N3	N4		N6		N8	N9	N10		N12	N13	N14
Row15		O2	O3	O4		O6		O8	O9	O10		O12	O13	O14
Row16		P2	P3	P4		P6		P8	P9	P10		P12	P13	P14
Row17		Q2	Q3	Q4		Q6		Q8	Q9	Q10		Q12	Q13	Q14
Row18	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
Row19	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Row20	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14

And, bearing in mind that besides the reduced number of pages that are being retrieved, the column-organized data is compressed using routines that can offer 10-fold improvements over pre 10.5 algorithms, and that the database engine will be taking advantage of hardware capabilities (e.g. Parallel vector processing, multi-core parallelism and single instruction, multiple data (SIMD) parallelism) to further improve the data processing speed, and you can anticipate very rapid analytical processing of large volumes of data.

The CTQ Boundary

The CTQ plan operator represents the transition between column-organized data processing and row-organized data processing.




```
TBSCAN
(   3)
9912.68
 4528
  |
  10
SORT
(   4)
9912.68
 4528
  |
11.6667
CTQ
(   5)
9912.67
 4528
  |
```

Keep it columnar as long as possible

The columnar table queue : The CTQ operator represents a boundary within the DB2® query engine. Operators that appear below the boundary process data as compressed column-organized vectors and tuples, whereas operators that are above the boundary operate on tuples that are not encoded.

Advertized Benefits of V11.1

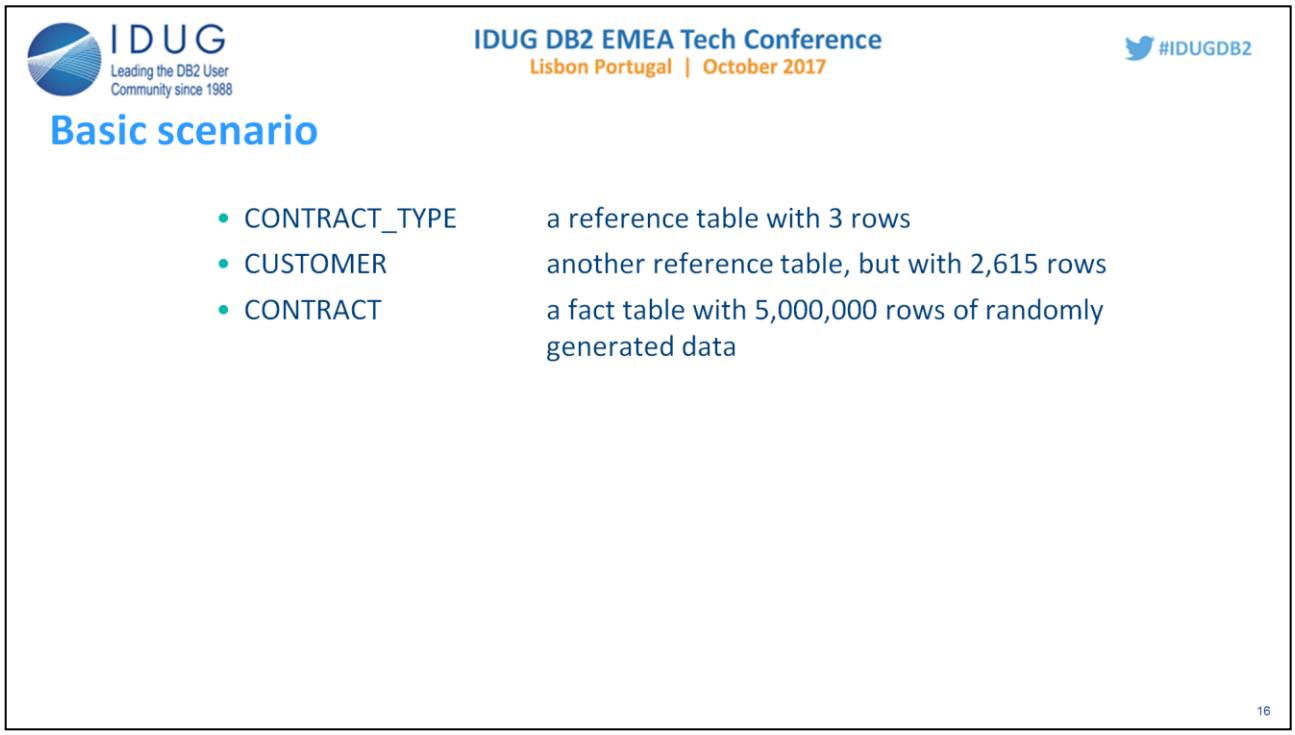
- OLAP operations
- NLJOIN
- Sorting
- IDENTITY and EXPRESSION columns
- NOT LOGGED INITIALLY

15

BLU Acceleration has also added SQL advances with richer function and compatibility including SQL compatibility with IBM PureData® System for Analytics. This enables native columnar online analytical processing for deep in-database analytics, the analytic capabilities of PureData System for Analytics, wide rows, new data types, logical character support, improved PostgreSQL compatibility, and a wide variety of additional SQL functions being incorporated in DB2 Version 11.1

enhancements include Nested Loop Join (NLJN) support, which features a fast radix sort with superior parallelism that is able to sort compressed and encoded data.

Other enhancements include BLU Acceleration support for IDENTITY and EXPRESSION generated columns, European Language support and NOT LOGGED INITIALLY support for column-organized tables.



I've defined a simple set of tables:

2 Dimension tables: very simple
1 Fact table with 5 million rows of data

we have a Primary Key on Contract_Type
A PK on Customer
And a PK on CONTRACT

Contract also has Foreign Key references to the 2 Fact tables

Test Case

```
explain plan with snapshot for
SELECT
    Name                    Customer
    ,Contract_Type
    ,SUM(Hours)             Total_Hours
FROM
    Contract
    inner join
    Customer        on Contract.Customer_ID = Customer.Customer_ID
    inner join
    Contract_Type   on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
GROUP BY
    Name
    ,Contract_Type
;
!db2exfmt -1 -d CONTRACT -e DB2I1054 -o Query_1_Shadow_EXP0076W.explain -u db2i1054
```

17

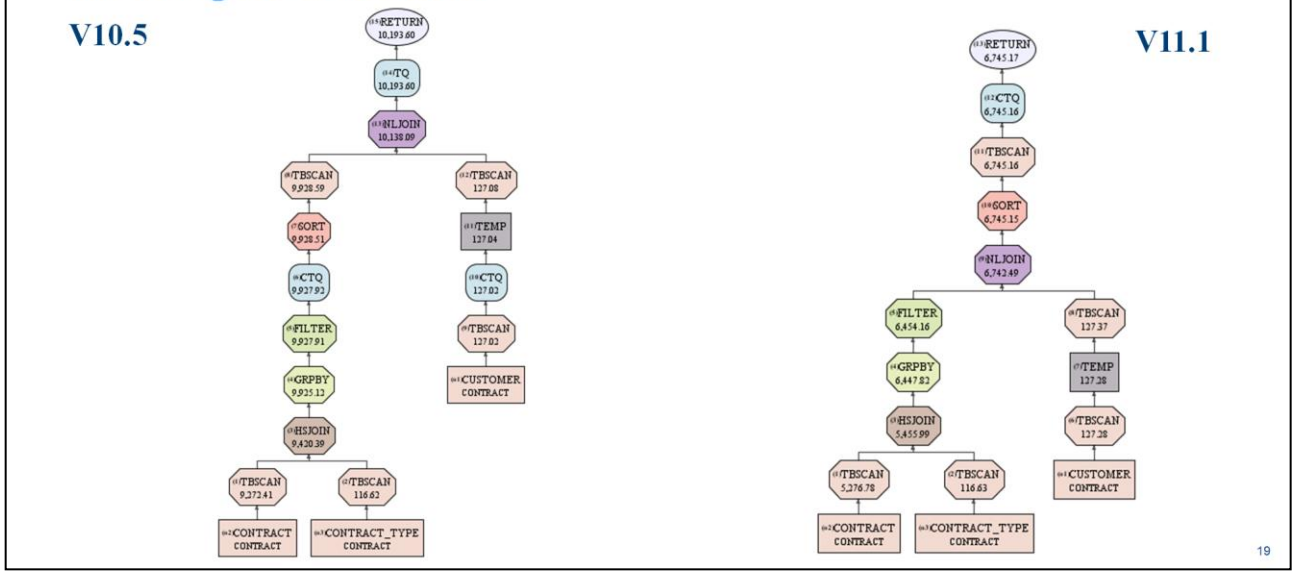
1) Take a simple 3-way join query
2) Put an explain on it
3) And format the output, so we can see what the optimizer is proposing to do

Working as advertised

```
WITH Contract_Agg AS
(
    SELECT
       Contract_Type,
       'H' as PC1, 'P' as PC2,
       YEAR(End_Date) AS Ending_in_Year,
       COUNT(Contract_Number) AS Number_of_Contracts,
       SUM(Hours) AS Total_Hours ,
       rank() over (partition by Contract_Type order by YEAR(End_Date) ) rn
  FROM
  Contract
  INNER JOIN Contract_Type
  ON Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
  GROUP BY Contract_Type, Customer_ID, YEAR(End_Date)
)

SELECT
    Contract_Type,
    Total_Hours,
    name,
    SUBSTR(Postcode,1,3)
FROM Contract_Agg ,
    Customer
WHERE Ending_in_Year BETWEEN 2005 AND 2015
AND SUBSTR(PostCode,2,1) = PC2
ORDER BY Total_Hours DESC
FETCH FIRST 40 ROWS ONLY
```


This is pretty horrible SQL but it is intended to show a particular access path, not be as efficient as possible.

Working as advertised

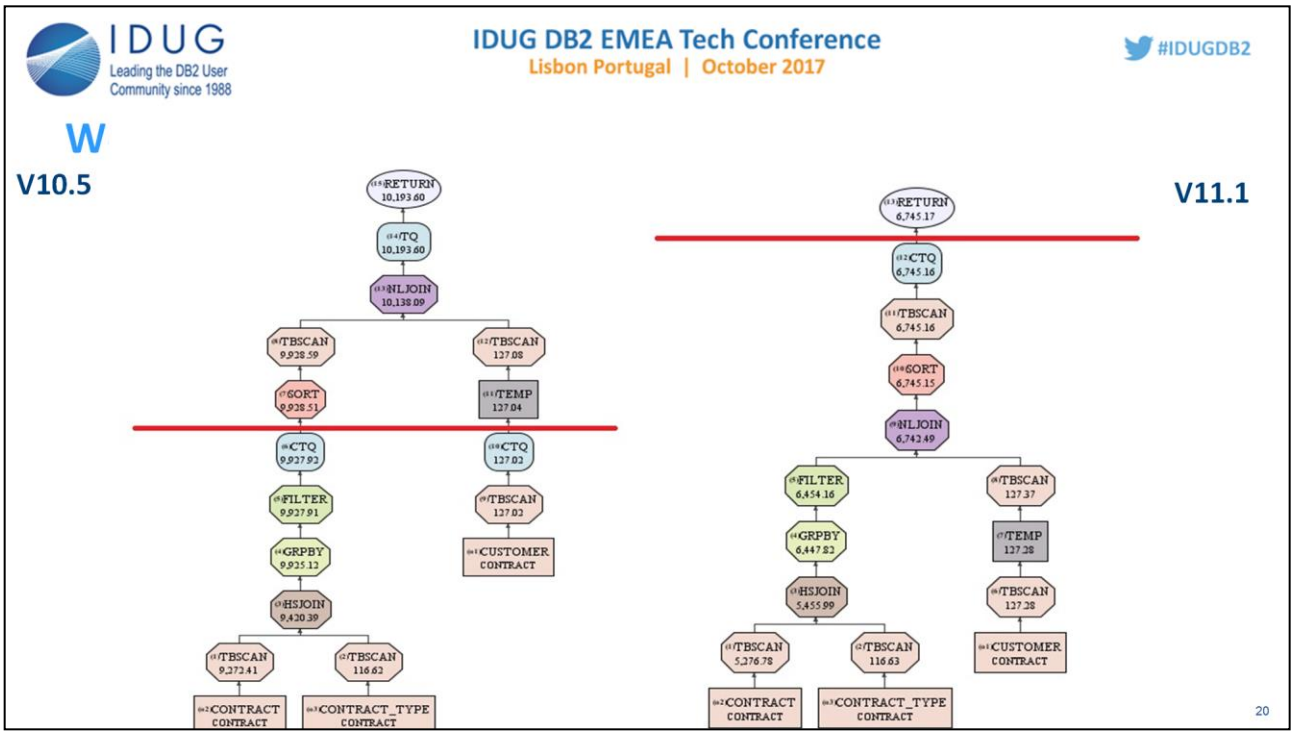


Explaining columnar based queries is a lot harder than explaining row-based ones; the access path often shows nothing more than TBSCAN

Q5 : V10.5 has the CTQ for both legs half way down

V11.1 has the CTQ right at the top, so has done Nested Loop Join, the Table Scan of the CUSTOMER temp table and the final sort, all in columnar format

Note it is also a more efficient retrieval of data from CONTRACT



Q5 : V10.5 has the CTQ for both legs half way down

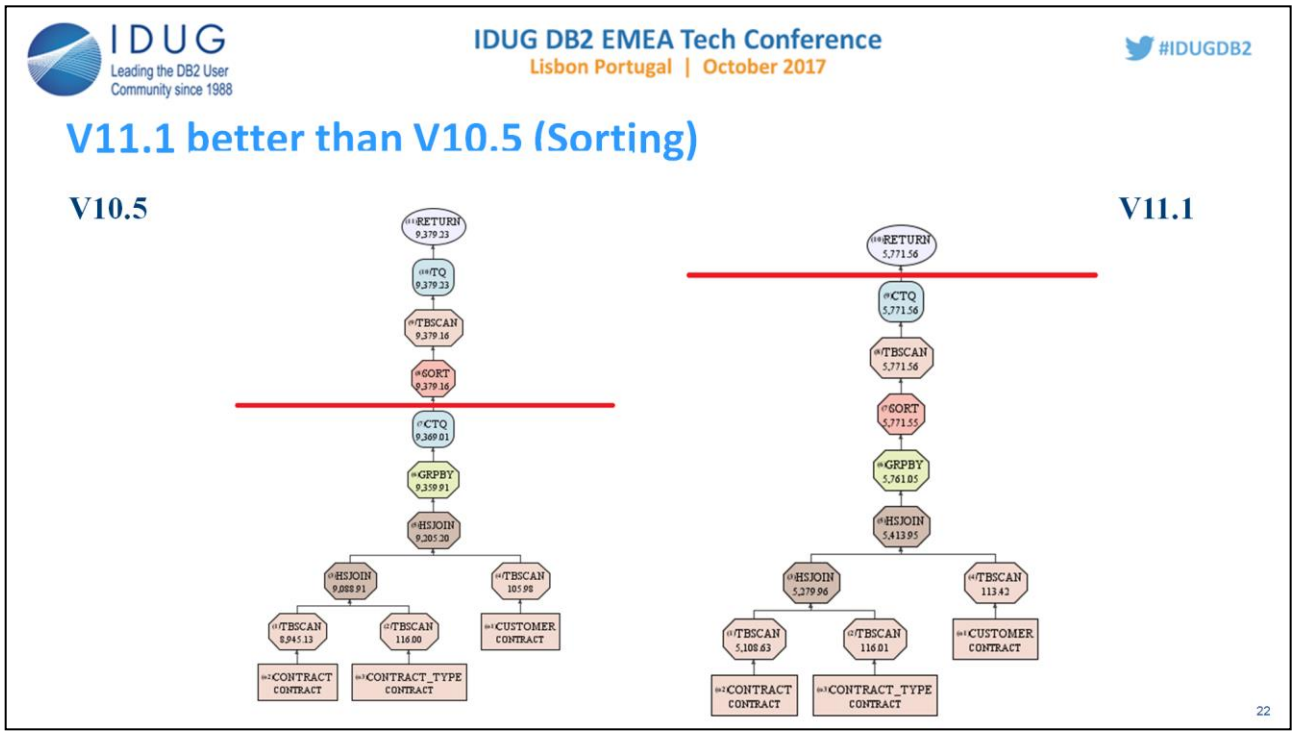
V11.1 has the CTQ right at the top, so has done Nested Loop Join, the Table Scan of the CUSTOMER temp table and the final sort, all in columnar format

Note it is also a more efficient retrieval of data from CONTRACT

V11.1 better than V10.5 (Sorting)

```
WITH    Contracts_CTE AS
(
SELECT
    Contract_Type,
    RTRIM(SUBSTR(Name, locate(' ', name ) + 1 ))    Business_Type ,
    Start_Date,
    Contract_Length
FROM
    Contract
    inner join
    Customer        on Contract.Customer_ID = Customer.Customer_ID
    inner join
    Contract_Type   on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
)

SELECT
    Business_Type,
    Contract_Type,
    AVG(CONTRACT_LENGTH)                    Average_Contract_Length,
    YEAR(Start_Date)                        Start_Year
FROM    Contracts_CTE
WHERE
    Contract_Type = 'COD'
AND Business_Type NOT IN ('Ltd.','& Sons')

GROUP BY Business_Type, Contract_Type, YEAR(Start_Date)
ORDER by Start_year

FETCH FIRST 20 ROWS ONLY
```


Establish the company type by trapping the last word of the company title



Q2 : SORT has moved above the CTQ boundary AND it's now a much more efficient sort:

a radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines.[1]

Basically this is enabling the highly-compressed columnar data to be sorted without unpacking it and / or using extra memory

IDUG
Leading the DB2 User Community since 1988

IDUG DB2 EMEA Tech Conference
Lisbon

#IDUGDB2

Paradis Sort

PARADIS: An Efficien
for In-place R

In-place radix sort is a popular distributic
algorithm for short numeric or string keys
run-time and constant memory complexity
cient parallelization of in-place radix sort i
ing for two reasons. First, the initial pha
elements into buckets suffers read-write de
ent in its *in-place* nature. Secondly, load l
recursive application of the algorithm to the
ets is difficult when the buckets are of ver
which happens for skewed distributions of

Table 1: Notations in this paper

$\mathcal{N}$	set of array indices $\{0, 1, ..., \lvert\mathcal{N}\rvert - 1\}$
$d[\mathcal{N}]$	the array of size $\lvert\mathcal{N}\rvert$ to be sorted
n, h, t	array index $\in \mathcal{N}$
$\mathcal{P}$	set of processor indices $\{0, 1, ..., \lvert\mathcal{P}\rvert - 1\}$
p, q	processor index $\in \mathcal{P}$
$p_0, p_1, ...$	shorthand for "processor 0", "processor 1", ...
$\mathcal{B}$	set of bucket indices $\{0, 1, ..., \lvert\mathcal{B}\rvert - 1\}$
i, j, k	bucket index $\in \mathcal{B}$
$\mathcal{L}$	set of recursion levels $\{0, 1, ..., \lvert\mathcal{L}\rvert - 1\}$
l	recursion level $\in \mathcal{L}$
$b(v)$	index of the bucket where element v should belong
gh_i	head pointer of bucket i
gt_i	tail pointer of bucket i
ph_i^p	head pointer of the stripe for processor p in bucket i
pt_i^p	tail pointer of the stripe for processor p in bucket i
$\mathcal{M}_i$	$\{n \mid gh_i \leq n < gt_i\}$, i.e., the indices of bucket i
$\mathcal{M}_i^p$	$\{n \mid ph_i^p \leq n < pt_i^p\}$, i.e., the indices of stripe p, i
C_i	$\lvert\mathcal{M}_i\rvert = gt_i - gh_i$, i.e., size of bucket i
C_i^p	$\lvert\mathcal{M}_i^p\rvert = pt_i^p - ph_i^p$, i.e, size of stripe p, i
$C_i(k)$	$\lvert\{n \in \mathcal{M}_i \mid b(d[n]) = k\}\rvert$ i.e. the number elements in $\mathcal{M}_i$ belonging to $\mathcal{M}_k$
$C_i^p(k)$	$\lvert\{n \in \mathcal{M}_i^p \mid b(d[n]) = k\}\rvert$ i.e. the number elements in $\mathcal{M}_i^p$ belonging to $\mathcal{M}_k$

3

Asked John Hornibrook at IBM Toronto for some detail

He explained that it is an implementation of a RADIX sort call PARADIS

Very detailed technical paper of which this is the intro. Read and digest

But then you get to the notations and it becomes clear that this is beyond the comprehension of most mortals

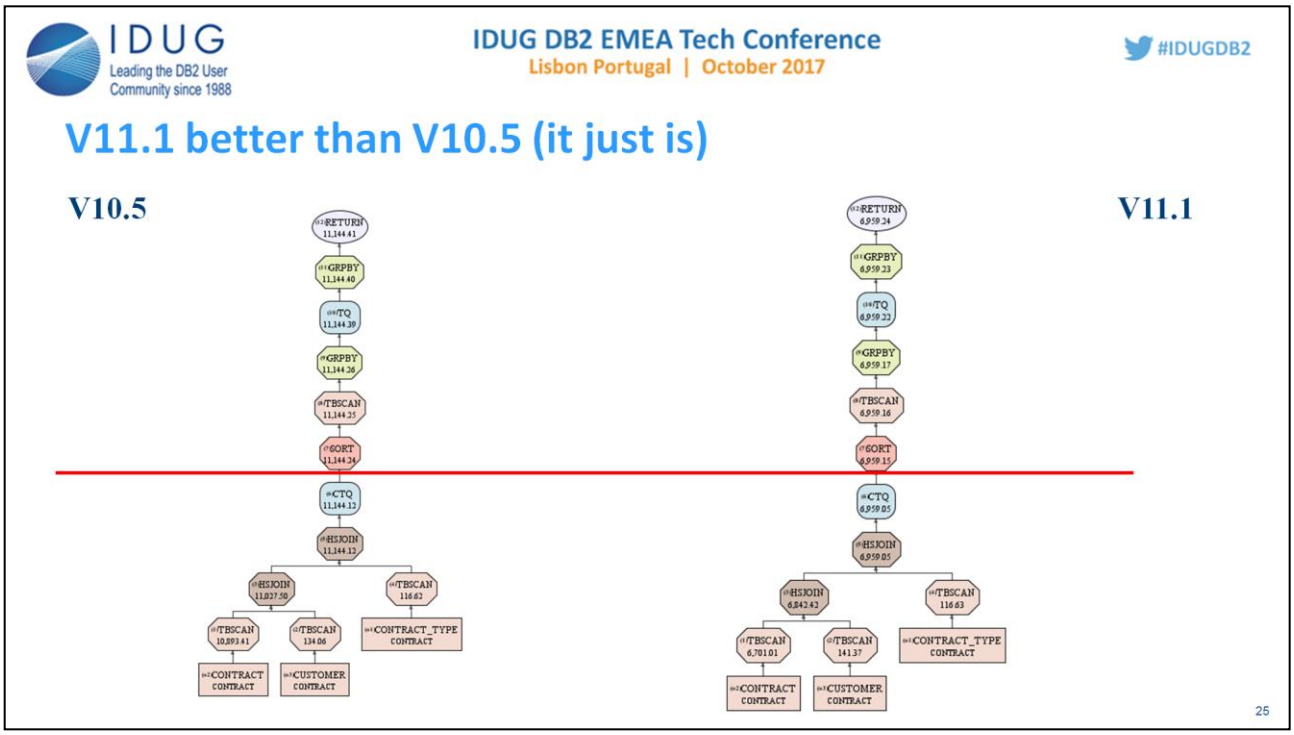
V11.1 better than V10.5 (it just is)

```
WITH    Remaining_Hours AS (
SELECT
    Name                        Customer,
    Contract_Number ,
    Contract_Type,
    MONTH(End_Date)             AS Ending_in_Month,
    YEAR(End_Date)              AS Ending_in_Year,
    end_date,
    Smallint(Days(End_Date) - Days(CURRENT DATE)) as Days_Remaining,
    Hours
FROM
    Contract
    inner join
    Customer        on Contract.Customer_ID = Customer
    inner join
    Contract_Type   on Contract.Contract_Type_ID = Con
WHERE
    End_date > CURRENT DATE
AND Contract_Number BETWEEN 120000 AND 250000
AND SUBSTR(postcode,1,1) = 'H'
)
```

```
SELECT
    Customer,
    Contract_Type,
    COUNT(Contract_Number)      AS Number_of_Contracts,
    Ending_in_Month,
    Ending_in_Year,
    SUM(Hours)
FROM Remaining_Hours
GROUP BY
    Customer, Contract_Type, Ending_in_Month, Ending_in_Year
ORDER BY 4,5,2,1
```

So, a similar query; the conventional 3-way join in a Common Table Expression but with a WHERE clause

And a select based on that CTE with OLAP functions and aggregation but no further where clause



Q3 : identical set of operations, V11.1 is cheaper

Note the CTQ boundary has not been moved above the SORT or GROUP BY functions, so V11.1 is still doing these operations more efficiently, even when the data has been returned to a row-based format

Provisos : this is db2expln output and therefore estimated costs. But it is trying to reproduce scenarios observed in actual client implementations. Often found that the before and after upgrade figures are very different, even if the actual access path is identical.

Suggestions (Calisto Zuzarte)

(a) reloading the tables and getting better compression ?

(b) new statistics … for example auto-runstats after extents were freed up ?

(c) FPAGES and NPAGES comparison?

Examining columnar table scans

MON_GET_INDEX

Query	Type	Table	Index	Index Type	BufferPool Reads	Physical Reads	Index Pages found in BP	Index Only Scans	Index Scans	Logical reads	Physical Reads
Q1	Col	CONTRACT_BLU	SQL140625033039096548	CPMA	3039	2	3037	0	0	3004	0
Q1	Col	SYN140625033039072302_CONTRACT_BLU	SQL140625033039114008	CPMA	0	0	0	0	0	0	0

MON_GET_TABLE

Query	Qtype	Table	Scans	Logical Reads	Physical Reads	Logical Pages	Logical Index Pgs	Num Cols Refd	Logical Reads	Physical Reads	Scans
Q1	Row	CONTRACT	66	5204973	23520	5354	7987	33	7299	1995	8
Q1	RoX	CONTRACT	66	5217741	25468	5354	7987	39	12768	1948	0
Q1	Col	CONTRACT_BLU	3	5533	265	1316	2886	13	2352	200	1
Q1	Col	SYN140625033039072302_CONTRACT_BLU	0	7	2	60	4	0	0	0	0

26

You can get some details; you can find the index that has been built on the CONTRACT table for instance and see how much read activity there is on that

You can find the synopsis table associated with your columnar data and see how heavily that is being used

But these are

```
WITH    Contracts_CTE AS
(
SELECT
    Name AS Custom
    Contract_Type,
    End_Date,
    COUNT(Contract
    SUM(Hours)
FROM
    Contract
    inner join
    Customer
    inner join
    Contract_Type
GROUP BY Name, Con
)
```

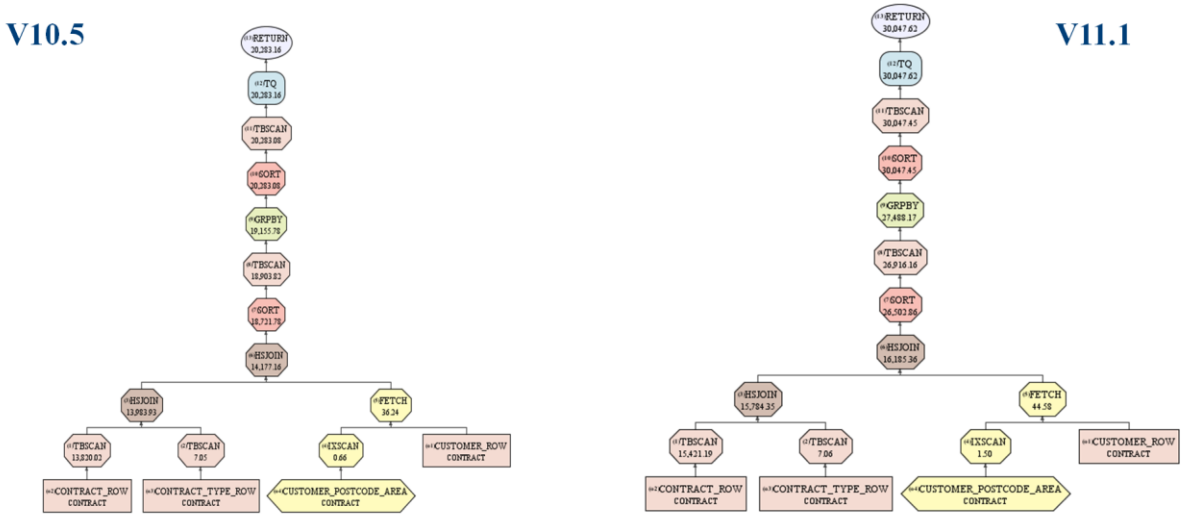
```
SELECT
    Customer,
    Contract_Type,
    Number_of_Contracts,
    YEAR(End_Date)              AS Ending_in_Year,
    MONTH(End_Date)             AS Ending_in_Month,
    CASE WHEN End_Date > CURRENT DATE
    THEN Smallint(Days(End_Date) - Days(CURRENT DATE))
    ELSE 0
    end as Days_Remaining,
    Total_Hours
FROM Contracts_CTE
WHERE YEAR(End_Date)=2017
ORDER BY Days_Remaining DESC
FETCH FIRST 200 ROWS ONLY
```

27

Conventional 3 way join with some aggregation to get the number of contracts and the total hours

And then the select to just retrieve this years data, sorted by the number of remaining days for each customer

Expected benefits not appearing



28

Q1 : identical set of operations, V11.1 is more expensive

This is, of course, a fudge. If you look at the table names you can see that these are all called xx_ROW and these are row based versions of the same tables (left over from some experiments with Shadow tables).

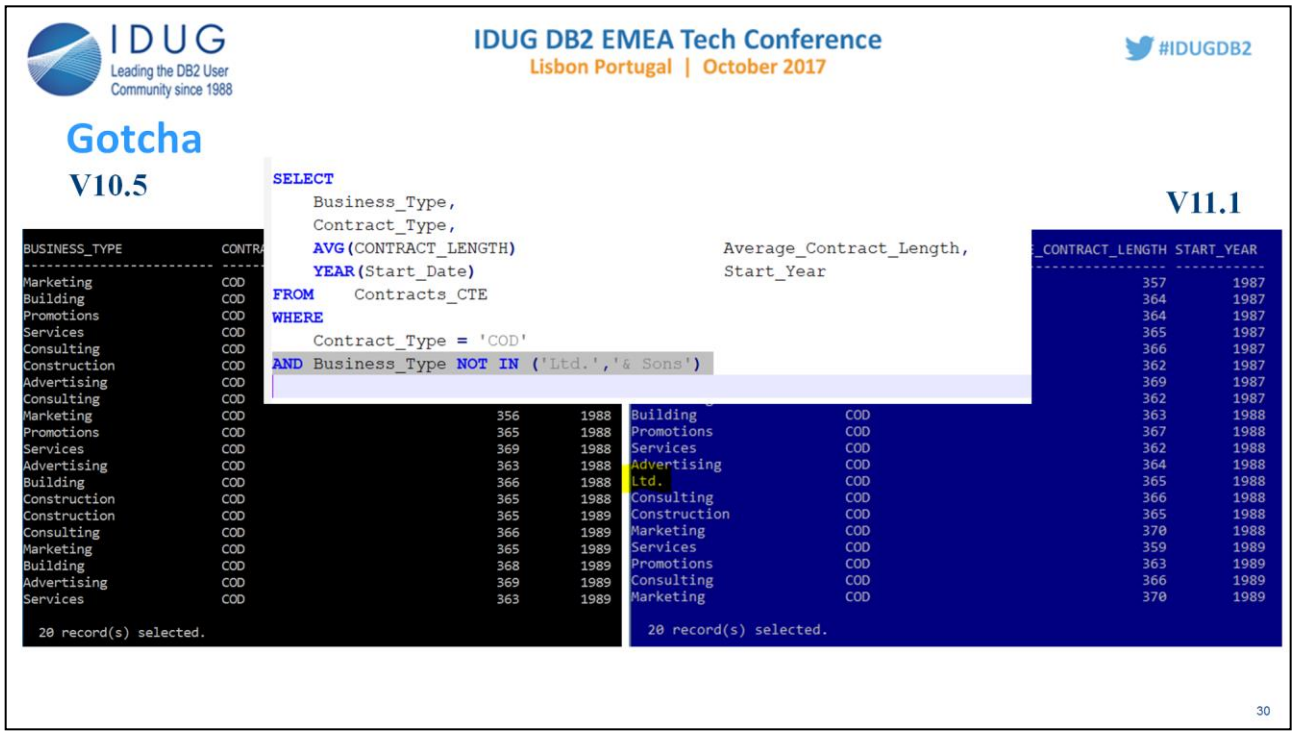
But the point is to illustrate that row-based processing may be more expensive in V11.1

Gotcha

```
WITH    Contracts_CTE AS
(
SELECT
    Contract_Type,
    RTRIM(SUBSTR(Name, locate(' ', name ) + 1 ))    Business_Type ,
    Start_Date,
    Contract_Length

FROM
    Contract
    inner join
    Customer        on Contract.Customer_ID = Customer.Customer_ID
    inner join
    Contract_Type   on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID

)

SELECT
    Business_Type,
    Contract_Type,
    AVG(CONTRACT_LENGTH)                    Average_Contract_Length,
    YEAR(Start_Date)                        Start_Year
FROM    Contracts_CTE
WHERE
    Contract_Type = 'COD'
AND Business_Type NOT IN ('Ltd.','& Sons')
```

29

Gotcha? Arguably, but this is something that stumped me enough that I ended up raising a PMR



The result set from V10.5 is pretty much what I was after

But the result set from V11.1 includes data that I specifically requested be excluded

Gotcha

```
[db2insl1@itchy:~]$ db2set
DB2_COMPATIBILITY_VECTOR=ORA
DB2_WORKLOAD=ANALYTICS
DB2_HISTORY_FILTER=G
DB2_USE_ALTERNATE_PAGE_CLEANING=ON [DB2_WORKLOAD]
DB2_ANTIJOIN=EXTEND [DB2_WORKLOAD]
DB2COMM=TCPIP
[db2insl1@itchy:~]$
```

31

Here's what IBM identified as the problem:

The registry variables include a setting for Oracle compatibility

PMR 15263,999,866

The back end engineer has created the following APAR

IT19976
ADDITIONAL SPACE IS ADDED TO "IN" CLASUE IN TABLE ORGNIZED BY COLUMN

Local Fix:

```
Rewrite the query to avoid this condition.
```

32

This is cut and pasted verbatim from the PMR as written by IBM, so any spelling mistakes are as received.

The bottom line here is that, from v10.1 onwards, there was a bug in the code that appended a space to the supplied variable, when Oracle compatibility was switched on.

This will be fixed with the APAR shown. Not sure when this will be supplied.

Although it does comes with this helpful snippet of advice

Gotcha

```
[db2ins11@itchy:~]$ db2set DB2_COMPATIBILITY_VECTOR=
[db2ins11@itchy:~]$ db2set
DB2_WORKLOAD=ANALYTICS
DB2_HISTORY_FILTER=G
DB2_USE_ALTERNATE_PAGE_CLEANING=ON [DB2_WORKLOAD]
DB2_ANTIJOIN=EXTEND [DB2_WORKLOAD]
DB2COMM=TCPIP
[db2ins11@itchy:~]$ db2stop
```

BUSINESS_TYPE	CONTRACT_TYPE	AVERAGE_CONTRACT_LENGTH	START_YEAR
Construction	COD	365	1987
Services	COD	362	1987
Marketing	COD	364	1987
Building	COD	364	1987
Promotions	COD	357	1987
Advertising	COD	369	1987
Consulting	COD	362	1987
Construction	COD	365	1988
Services	COD	362	1988
Consulting	COD	366	1988
Building	COD	363	1988
Marketing	COD	370	1988
Advertising	COD	364	1988
Promotions	COD	367	1988
Construction	COD	366	1989
Promotions	COD	363	1989
Advertising	COD	361	1989
Building	COD	361	1989
Consulting	COD	366	1989
Services	COD	359	1989

20 record(s) selected.

33

So, you just need to remove that registry variable and your V11.1 query will return the correct results

Horror story

```
DB2 Universal Database Version 9.7, 5622-044 (c) Copyright IBM Corp. 1991, 2009
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool

********************* PACKAGE ************************************

Section Code Page = 819

Estimated Cost = 4846.043457
Estimated Cardinality = 0.000137
```


It was explained before the upgrade (from V9.7) and had an acceptable estimated cost

Horror story

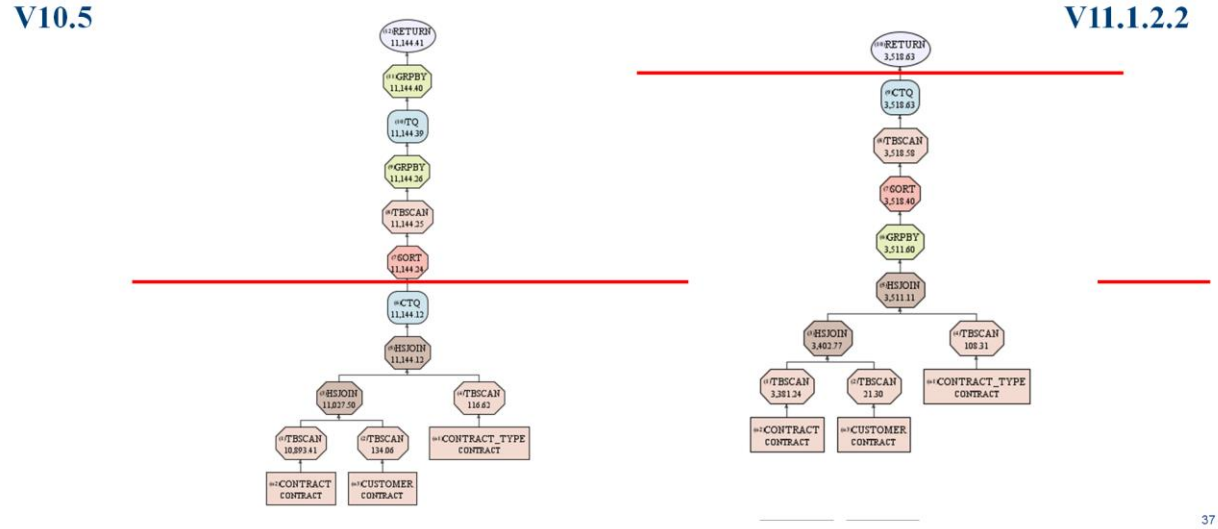
```
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2015
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL and XQUERY Explain Tool

Section Code Page = 819

Estimated Cost = 8215343923200.000000
Estimated Cardinality = 0.805647
```

36

So we've upgraded to V11.1, rebound all the procs and this one (and I think it was the only one) goes from 4,500 timerons to 8.2 trillion. Needless to say, this did not represent a significant performance improvement

The problem actually lay, not with V11.1, but with the way the database was managed with this particular client. REORGs and RUNSTATS were done rarely and REBINDS never, once the "optimum" access path had been achieved. With an upgrade, of course, all packages needed to be rebound.

V11.1 better than V10.5 (it just is)

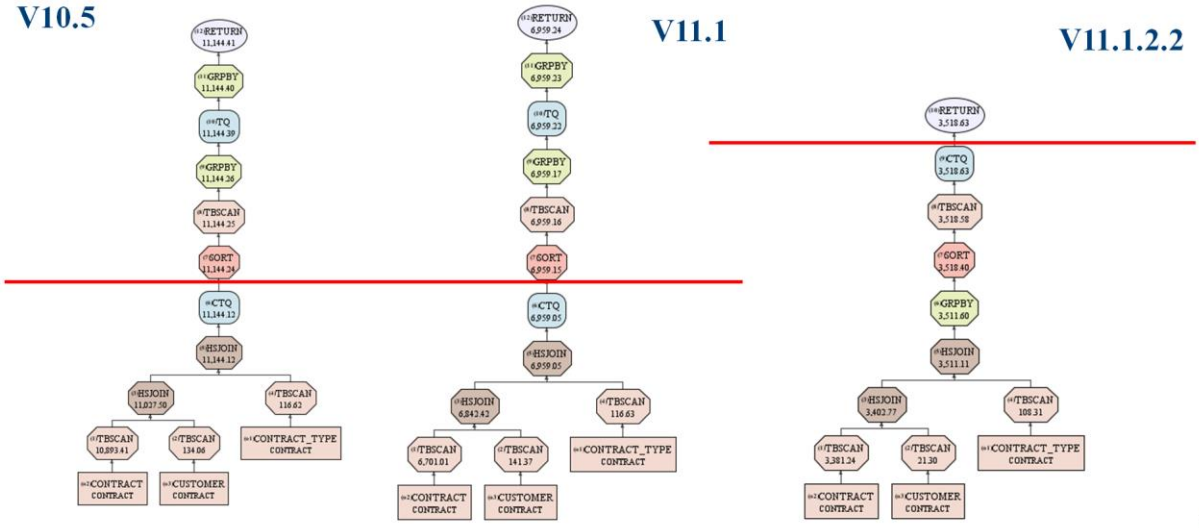


37

Q3 : identical set of operations, V11.1 is cheaper

Note the CTQ boundary has not been moved above the SORT or GROUP BY functions, so V11.1 is still doing these operations more efficiently, even when the data has been returned to a row-based format

V11.1 better than V10.5 (it just is)



38

So, looking at each access path side by side:

V10.5 compared to

V11.1 compared to

V11 FP 2

Comparison across the workload

Query	v11.1.1.1	FP2	% improvement

Summary

- There are significant performance improvements to be had by upgrading to V11.1
  - *Over 95% of code we have monitored has improved after the upgrade, with no changes to code or configuration: improvements were straight "out of the box"*
- Most improvements to be had are in the column-organized data
  - *Row-based should not suffer, unless your database is configured specifically for ANALYTICS*
- Any performance degradation is liable to be due to 'time-bombs'
  - *Record all stats, config settings and access paths before and after the upgrade, in order to help finger-pointing at the new release*

40

I'm still saying V11.1 because, although I've shared a few test results from our R&D server after upgrading to FP2, I haven't yet rolled FP2 out on any of our client sites

IDUG
Leading the DB2 User Community since 1988

IDUG DB2 EMEA Tech Conference
Lisbon Portugal | October 2017

#IDUGDB2

Mark Gillis
Triton Consulting
Mark.Gillis@Triton.co.uk

Session code: D15