



Central Canada Db2 User  
Group 2023

Sept 18/19 2023

# DB2 SQL – go beyond the usual – My current TOP 40 SQL tips, tricks, and opinions

Brian Laube

*Manulife Financial*

*Session Code: AP11*

 #IDUGDb2

Platform: Db2 z & Db2 LUW

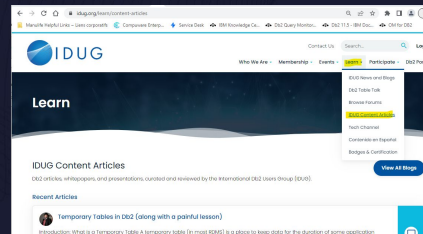
# Welcome to my presentation

- My name is Brian Laube.
- I work for Manulife Financial in Canada, known as John Hancock in the USA
  - DB2 DBA for over 20 years! Mostly zOS, some LUW. I can bluff my way through Db2 Connect
- I have presented at several IDUG NA (and two EMEA) since 2016
- Director of Central Canada (Toronto) user group since 2015. also presented
- IBM Champion for analytics since 2017
- Currently Chair of the IDUG content committee. We gather and write and organize BLOGs on topics of interest for all IDUG members!
- <https://www.idug.org/learn/content-articles>
  - Do you have any Db2 interests or experiences to share? Try blogging with us! Contact me and write a one-off blog or simply join the committee and help us get it done!

# IDUG CONTENT COMMITTEE

- I am a volunteer with IDUG. I am on the committee that organizes the CONTENT BLOG
- <https://www.idug.org/learn/content-articles>
- Visit our site. Review past blogs. There is always something interesting to learn for both Db2 Z and Db2 LUW.

*We are always looking for USERS of Db2 who are interested in writing an article or blog for IDUG Content! On any topic Db2 related! War Stories, experience with new Db2 functionality. Application! Development, Tools, **It is a great help to becoming an IBM Champion!** It is a great way to meet others and share your knowledge with the community. Contact any of us (names on website > or contact me > [brian@spufi.ca](mailto:brian@spufi.ca) )*



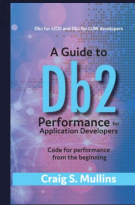
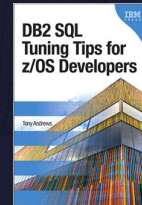
<https://www.idug.org/learn/content-articles>

## Agenda

- Modern SQL – Db2 and SQL have both evolved over the years!
- Opinions on SQL – I have opinions on what is better
- Best Practices for SQL – Beyond the usual best practices
- Questions

## Pre-amble (1 | 2)

- There are many good resources for opinions on SQL best practices. On the internet, in books and hopefully in your shop.
  - The Db2 SQL Tuning Tips for z/OS Developers by Tony Andrews has many practical short tips. Excellent.
  - Craig Mullens has several books (and blogs). I read the Db2 Developers Guide
- IDUG Content blog has many great blogs on this broader topic, including the following
- <https://www.idug.org/blogs/tony-andrews1/2022/09/28/db2-for-zos-sql-and-application-new-features-v8-v1>



<https://www.idug.org/blogs/tony-andrews1/2022/09/28/db2-for-zos-sql-and-application-new-features-v8-v1>

## Pre-amble (2 | 2)

- This presentation is not meant to restate what you can get from the previously mentioned usual sources for SQL tips
- I am trying to remind people of some new ways of doing things in SQL that I think are interesting. New functions in Db2 Z (and Db2 LUW).  
**Modern SQL!**
- I will also give what I think are important “**best practices**” for SQL.
  - Things that people STILL forget!
- I will also give my **opinion** on the right way to do things!
  - Obviously, opinions are my own and your mileage may vary.

## Modern SQL – typing FETCH FIRST nn ROWS ONLY is tiring. Use LIMIT

- FETCH FIRST NN ROWS ONLY has been around since forever.
  - We use it to tell Db2 that we only want NN rows from the table
- The new LIMIT syntax is easy to type and fast.
  - LIMIT requires APPLICATION COMPATIBILITY at 12R1M500+

```
-- TRADITIONAL FETCH FIRST NN ROWS ONLY
SELECT * FROM EMP
ORDER BY EMPNO
FETCH FIRST 15 ROWS ONLY
;
```

```
-- CHECK > APPL COMPATIBILITY SHOULD BE V12R1M500 OR
HIGHER
SELECT CURRENT APPLICATION COMPATIBILITY
FROM SYSIBM.SYSDUMMY1;
--SET CURRENT APPLICATION COMPATIBILITY = 'V12R1M500';

-- EQUIVALENT SQL SYNTAX USING LIMIT
SELECT * FROM EMP
ORDER BY EMPNO
LIMIT 15
;
```

# Modern SQL – SQL pagination

- Use OFFSET to tell Db2 to build the result set and order by as required (as always) and then skip over the first OFFSET nn rows.

```
-- THESE 3 SELECT PRODUCE THE  
SAME RESULT SET  
SELECT * FROM EMP  
ORDER BY EMPNO  
OFFSET 05 ROWS FETCH FIRST 10  
ROWS ONLY  
;  
SELECT * FROM EMP  
ORDER BY EMPNO  
LIMIT 10 OFFSET 05  
;  
SELECT * FROM EMP  
ORDER BY EMPNO  
LIMIT 05,10  
;
```

- Link to IDUG content blog on this topic:
- <https://www.idug.org/blogs/tony-andrews1/2022/10/03/sql-pagination-data-dependent-new-syntax-and-numer>

<https://www.idug.org/blogs/tony-andrews1/2022/10/03/sql-pagination-data-dependent-new-syntax-and-numer>



## OPINION – Why use OPTIMIZE FOR N ROWS (1 | 2)

- You can add OPTIMIZE FOR N ROWS as the final clause to tell the optimizer you want it to anticipate only N rows in the final result set.
  - This information may influence the optimizer to pick a relatively different access path!
- Just to be super clear. You are telling the optimizer to anticipate N rows in the result set. But if there are many more rows (or many less) than N, your application can get as many rows as it wants! *So maybe it will get all the rows!*

## OPINION – Why use OPTIMIZE FOR N ROWS (2|2)

- If you use OPTIMIZE FOR 1 ROWs then this is how you TELL the optimizer that you really expect 1 row and you plan on fetching 1 row. The optimizer will review and do everything it can to avoid sorting large sets. ***You can still fetch how ever many rows you want from the result set.***
- Basically, if possible, the optimizer will use an index. It will try to avoid sorting and use an index even if the static statistics would normally suggest it not do that!
  - TABLEA (COLA,COLB) has one index on COLA. And the static statistics say the table has 2 rows
  - Query1: the optimizer may decide to do a tablespace scan and just internally sort the 2 rows. That will be 1 I/O and relatively fast.
  - Query2: the optimizer may NOW decide to do I/O to index and then I/O to tablespace. Two I/O! Relatively expensive. But no sorting!

```
SELECT * -- query1
FROM TABLEA
ORDER BY COLA
;
```

```
SELECT * -- query2
FROM TABLEA
ORDER BY COLA
OPTIMIZE FOR 1
ROWS
;
```

## Best Practice – Volatile Tables

- Creating (or altering) a table with VOLATILE attribute is another way to influence the optimizer to pick index access.
- We mark certain tables as VOLATILE when we know the quantity of rows can vary from zero to a million. It is just a sequential dump table (unsurprisingly called TSEQ). We LOAD a million rows and then empty the table later.
- If we run RUNSTATS when the table is empty then later optimizer decisions may be misleading. VOLATILE helps remind Db2 to consider indexes, regardless of what the static statistics say!

```
SELECT * -- query1  
FROM TABLEA  
ORDER BY COLA  
;
```

```
SELECT * -- query2  
FROM TABLEA  
ORDER BY COLA  
OPTIMIZE FOR 1  
ROWS  
;
```

## OPINION – STATIC STATISTICS and REAL-TIME-STATISTICS [RTS] (1|2)

- Let me state it out loud.
- Static statistics are gathered by RUNSTATS (or inline STATISTICS in REORG and other utilities)
- I call them “static” statistics because they are accurate at the moment in time the statistics were gathered! But as time goes on, they may become out of sync with reality
- The static statistics are stored all over the Db2 catalog in various tables and contain lots of meta-info about the data including cardinality and distribution and common values.
- The Db2 OPTIMIZER eats all this information for making access path decisions.

I often say “static statistics” when referring to statistics gathered by RUNSTATS or inline STATISTICS. These “static statistics” are not to be confused with the real-time-statistics

## OPINION – STATIC STATISTICS and REAL-TIME-STATISTICS [RTS] (2 | 2)

- The real-time-statistics (RTS) are gathered by Db2 all the time and updated constantly. SYSTABLESPACESTATS & SYSINDEXSPACESTATS. These tables contain basic meta-data about these physical objects and big info about your physical objects. The RTS is intended to aid your decisions when to run Db2 Utilities (REORG, COPY, or RUNSTATS itself)
- Interestingly, the RTS is incredibly accurate.
- If you want to make informed decisions about WHEN to run utilities, then you can manually make your own query to the RTS.
- A popular option for keepers is to use IBM provided procedure DSNACCOX which is documented by IBM documentation. Lots of decisions and things to think about (of course)  
<https://www.ibm.com/docs/en/db2-for-zos/12?topic=db2-dsnaccox>
- ***The RTS has no influence to the optimizer!***

I often say “static statistics” when referring to statistics gathered by RUNSTATS or inline STATISTICS. These “static statistics” are not to be confused with the real-time-statistics

## Best Practice – understand RTS (1 | 2)

- The RTS provides powerful information and quickly helps you understand your physical objects. It is important to review the column descriptions and understand what info is provided by RTS
  - <https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-systablespacestats>
    - SPACE is allocated space in KB (divide by 48 to see tracks!)
    - DATASIZE is amount of space used inside the allocated space, in bytes! (not KB)
    - TOTALROWS (should be obvious)
    - UNCOMPRESSED DATASIZE – good name! But is a trick! Not used! Always zero
  - <https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-sysindexspacestats>
    - TOTALENTRIES – nbr of entries in the index (if unique -> should equal totalrows)
- Keen users of RTS will enable them as temporal tables with their corresponding “history” tables. This is fine. I did it. But think about purge strategy in the history table!

Knowing the column descriptions of RTS tables is important.

SPACE divided by 48 is the exact quantity of tracks for the dataset. It always matches for me!

In my world. My virtual DASD still has tracks and cylinders. Db2 uses 48KB per track (and gets 15 tracks per cylinder)

I sometimes convert SPACE and DATASIZE to MB or GB to make it easier to read.

I SPACE by DATASIZE

## Best Practice – understand RTS (2|2)

DBNAME	TSNAME	SPACE_MB	SPACE_TRKS	DATASIZE_MB	PCT_USED	TOTALROWS	MAX_PART	CNT
----	----	-----	-----	-----	-----	-----	-----	-----
----	-	1905292	40646239	1399102	73.43	16908212253		70
3434								
DIL01P	ZHH	319903	6824610	242061	75.66	2382176145		70
70								
DYR01P	ZDDDDD	171765	3664335	143262	83.40	614870209		64
64								
DYR01P	ZRGSD	92813	1980015	76292	82.20	717490535		32
32								
DYR01P	ZNTSD	80002	1706730	66596	83.24	494017180		32
32								
DYB01P	ZCLM	71658	1528725	57888	80.78	743865554		20
20								
DIL04P	ZUHCO	61457	1311090	43522	70.81	1099519653		14
14								
DIL01P	ZUHCO	56670	1208970	39470	69.64	1103649819		14
14								
DCL01P	ZAH1	55122	1176165	11165	75.10	146448200		20
20								
DYD01P	ZD10P							
14								
DYR01P	ZDNTD							
32								
DIL04P	ZDCLM							
14								
DYR01P	ZDCLM							
14								
DYR01P	ZDCLM							
32								
DYR01P	ZDCLM							
14								

Knowing the column descriptions of RTS tables is important.  
 SPACE divided by 48 is the exact quantity of tracks for the dataset. It always matches for me! Look it matches in the screenshots  
 In my world. My virtual DASD still has tracks and cylinders. Db2 uses 48KB per track (and gets 15 tracks per cylinder)  
 I sometimes convert SPACE and DATASIZE to MB or GB to make it easier to read.  
 I SPACE by DATASIZE

## Modern SQL – Stop Joining in your WHERE clause. Use JOIN syntax

- Classic (old-fashioned) “joining” tables in the FROM <table list> via WHERE clause is classic. Of course, it is only for “inner join”
- Modern SQL “join” syntax is more explicit and allows left outer join which is so very often useful!

```
-- old fashioned JOIN - inner
SELECT E.EMPNO, E.FIRSTNAME,
E.LASTNAME, E.WORKDEPT, D.DEPTNAME
FROM EMP E, DEPT D
WHERE 1=1
      AND E.WORKDEPT = D.DEPTNO
ORDER BY E.EMPNO
;

-- modern JOIN syntax
-- same result as above
SELECT E.EMPNO, E.FIRSTNAME,
E.LASTNAME, E.WORKDEPT, D.DEPTNAME
FROM EMP E
INNER JOIN DEPT D
      ON E.WORKDEPT = D.DEPTNO
WHERE 1=1
ORDER BY E.EMPNO
;
```



## Opinion – join syntax should always be explicit

- I already said stop using join via FROM table list and WHERE clause joining.
- Use JOIN syntax.
- The default when joining two tables is INNER JOIN. So you don't have to write the word INNER. But I encourage to explicit and put INNER.
  - It is just clearer to say INNER when you know it is INNER
- More opinion > How often should you code "right outer join"? Never. It confuses me. LEFT OUTER JOIN is more common. And besides, if you code right outer join, Db2 will just internally transform it to LEFT. *Even Db2 dislikes right outer join!*

```
-- EXPLICIT INNER JOIN
SELECT A.COL1, A.COL2, B.COLX
FROM TABLE A
INNER JOIN TABLE B
    ON (A.COL1 = B.COL1)
ORDER BY A.COL1
;

-- MAYBE NO B ROW EXISTS
SELECT A.COL1, A.COL2, B.COLX
FROM TABLE A
LEFT OUTER JOIN TABLE B
    ON (A.COL1 = B.COL1)
ORDER BY A.COL1
;
```

# Modern SQL – match on groups of columns

- We can now match table columns in joins or where clause with sets or groups of columns. This makes the SQL easier to read and reduces overall length because I now put groups of columns on one line!

- I use it all the time in JOIN!
- It also works in correlated sub-selects!

```
SELECT Z.DBNAME, Z.NAME AS TSNAME
, Z.TYPE AS TSTYPE
, T.CREATOR AS TCREATOR, T.NAME AS TNAME
, T.TYPE AS TTYPE
FROM SYSIBM.SYSTABLESPACE Z
LEFT OUTER JOIN SYSIBM.SYSTABLES T
  ON (Z.DBNAME,Z.NAME) = (T.DBNAME,T.TSNAME)
  AND T.TYPE = 'T'
WHERE 1=1
  AND Z.DBNAME = 'DSN8D12A'
-- AND Z.DBNAME = 'DSNDB06'
-- AND Z.TYPE = 'G'
-- AND Z.CREATEDTS > (CURRENT TIMESTAMP - 999 DAYS)
ORDER BY Z.DBNAME, Z.NAME
;
```

This becomes more and more beneficial when matching on many columns. Much easier to read!

# Modern SQL - use COL groups to start in middle

Modern SQL makes it easier to read And understand WHEN you want to start in the middle of a result set!

```
1 -- FIRST, LOOK AT ALL EMP
2 SELECT LASTNAME, FIRSTNAME, EMPNO, WORKDEPT
3 FROM EMP
4 ORDER BY LASTNAME, FIRSTNAME, MIDINIT
5 ;
6 -- IMAGINE, YOU WANT TO START FROM MIDDLE
7 -- TRADITIONAL WAY TO START IN MIDDLE BASED UPON 2 COLS
8 SELECT LASTNAME, FIRSTNAME, EMPNO, WORKDEPT
9 FROM EMP
10 WHERE LASTNAME > 'GOUNOT'
11 OR (LASTNAME = 'GOUNOT' AND FIRSTNAME > 'JACK')
12 ORDER BY LASTNAME, FIRSTNAME, MIDINIT
13 ;
14 -- MODERN WAY TO START IN MIDDLE BASED UPON 2 COLS
15 SELECT LASTNAME, FIRSTNAME, EMPNO, WORKDEPT
16 FROM EMP
17 WHERE (LASTNAME, FIRSTNAME) > ( 'GOUNOT', 'JACK' )
18 ORDER BY LASTNAME, FIRSTNAME, MIDINIT
19 ;
```

LASTNAME	FIRSTNAME	EMPNO	WORKDEPT
ADAMSON	BRUCE	000150	D11
ALONZO	ROY	200340	E21
BROWN	DAVID	000200	D11
GEYER	JOHN	000050	E01
GOUNOT	JASON	000340	E21
HAAS	CHRISTINE	000010	A00
HEMMINGER	DIAN	200010	A00
HENDERSON	EILEEN	000090	E11
JEFFERSON	JAMES	000230	D21
JOHN	REBA	200220	D11
JOHNSON	SYBIL	000260	D21

LASTNAME	FIRSTNAME	EMPNO	WORKDEPT
GOUNOT	JASON	000340	E21
HAAS	CHRISTINE	000010	A00
HEMMINGER	DIAN	200010	A00
HENDERSON	EILEEN	000090	E11
JEFFERSON	JAMES	000230	D21
JOHN	REBA	200220	D11
JOHNSON	SYBIL	000260	D21
JONES	WILLIAM	000210	D11

Here I use the IBM provided EMP table which most of us already have installed in our Db2.

I query the whole EMP and order it. The first result is the beginning

The second two queries want to start in the middle of that result. They both produce the same result. The second query is easier (to me) to read!

## Modern SQL – HOW to code conditions on LEFT OUTER table

- When using LEFT OUTER join, it can get complicated if the LEFT OUTER join table has conditions in the WHERE clause.

- If you want them to be always true, then the join essentially turns into INNER JOIN
- If you want the condition to only be evaluated if the OUTER table row qualifies then it is tricky

```
-- SELECT FROM TABLE TABLE FOR SOME
-- CONDITION ON COLB
-- >> WHERE P.COLB MUST BE IN SOME LIST
-- LEFT OUTER JOIN TO SHOW COL_CD1 DESCRIPTION
-- FROM CHILD TABLE TCHILD. BUT WE WANT DESC
-- WHEN LANG_CD = E
-- > SOMETIMES CHILD TABLE IS MISSING ROWS
-- > WE DO LEFT OUTER JOIN TO STILL SEE ROWS
-- > FROM MAIN TABLE
--
-- THIS SQL MAY NOT WORK AS YOU DESIRE
SELECT P.COLA, P.COLB, P.COL_CD1
, C.CDDESC
FROM TABLE P
LEFT OUTER JOIN TABLE C
ON P.COL_CD1 = C.COL_CD1
WHERE 1=1
AND P.COLB IN (1,2,3)
AND C.COL_LANG_CD = 'E'
;
```

```
-- THIS COMPLICATED SQL SHOULD
-- WORK AS DESIRED
SELECT P.COLA, P.COLB, P.COL_CD1
, C.CDDESC
FROM TABLE P
LEFT OUTER JOIN TABLE C
ON P.COL_CD1 = C.COL_CD1
WHERE 1=1
AND P.COLB IN (1,2,3)
AND (C.COL_LANG_CD = 'E'
OR C.COL_LANG_CD IS NULL
)
;

-- BETTER!! THE LEFT OUTER JOIN TABLE
-- CONDITIONS ARE IN ON CLAUSE,
-- NOT THE WHERE !!
SELECT P.COLA, P.COLB, P.COL_CD1
, C.CDDESC
FROM TABLE P
LEFT OUTER JOIN TABLE C
ON P.COL_CD1 = C.COL_CD1
AND C.COL_LANG_CD = 'E'
WHERE 1=1
AND P.COLB IN (1,2,3)
;
```

If you code a WHERE equal condition then it must be true. But if the outer join table has no row then the value is essentially null and not that desired value  
So if you keep the condition in the WHERE clause you have to add a OR NULL check.

Better is to put the condition the JOIN clause

## Modern SQL – grouping sets

- Use of GROUP BY provides a quick way to do things like SUM many rows. We all use it all the time
- Use of GROUP BY GROUPING SETS provides a way to group by multiple sets of columns all at once. It basically provides a *super quick way to produce easy to read reports with just SQL!*
  - > I often use COALESCE to convert grouping columns from NULL to something pretty to read

```
SELECT COALESCE(DBNAME, '-ALL') AS DBNAME
, SUM(SPACE) AS SPACE
FROM SYSIBM.SYSTABLESPACESTATS
WHERE 1=1
GROUP BY GROUPING SETS ((DBNAME), ())
ORDER BY SPACE DESC
;
```

DBNAME	SPACE
-ALL	2147037664
DIL01P	535818992
DYR01P	527364512
DIL04P	177562720
DSNDB07	166626432
DCL01P	131756736
DYB01P	122052960

Thanks to presentations from Tony Andrews and David Simpsons from years ago to remind me OF THE POWER OF GROUPING SETS

## Opinion – use WHERE 1=1 in ad-hoc SQL

- I often use Data Studio to build my ad-hoc SQL.
- First, it has intellisense for popping up table-names and column-names. Very useful.
- Second, Data Studio has super easy function to change select text to SQL comment and add dash-dash (or undo)
- I am now in the habit of beginning my ad-hoc SQL with WHERE 1=1.
- This allows me to build a list of AND conditions and easily go back and forth and comment them out and put them back in. Otherwise, when I want to comment out the first condition after the WHERE, it annoys me and gets complicated.

```
-- example dynamic ad-hoc SQL I might build
-- to look at Db2 catalog
SELECT Z.DBNAME, Z.NAME AS TSNAME
, Z.TYPE AS TSTYPE
, T.CREATOR AS TCREATOR, T.NAME AS TNAME
, T.TYPE AS TTYPE
FROM SYSIBM.SYSTABLESPACE Z
LEFT OUTER JOIN SYSIBM.SYSTABLES T
  ON (Z.DBNAME,Z.NAME) = (T.DBNAME,T.TSNAME)
  AND T.TYPE = 'T'
WHERE 1=1
  AND Z.DBNAME = 'DSN8D12A'
-- AND Z.DBNAME = 'DSNDB06'
-- AND Z.TYPE = 'G'
-- AND Z.CREATEDTS > (CURRENT TIMESTAMP - 999
DAYS)
ORDER BY Z.DBNAME, Z.NAME
```

The example SQL is now very easy to comment out any of WHERE AND conditions

## Best Practice – no WHERE 1=1 in applications

- It is considered best practice to not have WHERE 1=1 in any application SQL. This includes both dynamic SQL and static SQL
- Apparently, there is a common(?) hacking technique (SQL injection) to look for network traffic with those characters and then somehow infer something because it is probably SQL.

## Modern SQL – select all COLS from one table > and some COLS from another (easier today!)

- If you have multiple tables in your FROM clause, with “SELECT \*” you previously got all columns from all tables
- Now you can select all columns from one table and a few specific columns from another
  - I find this useful when doing ad-hoc SQL and I want add a new join to the SQL. I can quickly SELECT all the columns from the new table to review and think what specific columns I really want.

```
SELECT A.COL1, A.COL2, B.*  
FROM TABLEA A  
INNER JOIN TABLEB B  
    ON (A.COL1 = B.COL1)  
ORDER BY A.COL1  
;
```

I ONLY LEARNED THIS TRIVIA IN THE LAST FEW YEARS



## Opinion – application compatibility is hard to spell

- I constantly mis-spell compatibility. And it is so long to type.
- Because I am keen, I check the application compatibility of my tools all the time (Data Studio, DSNTEP2, SPUFI, etc).
  - I made an RFE to IBM to make an alternative special register name
  - <https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1194>
- This is minor... but why not make Db2 a bit easier...

## Modern SQL – Common Table Expressions (1 | 2)

- Common Table Expression (CTE) are powerful SQL constructs that should be moved more often. In my opinion they can help break apart complicated SQL requirements into smaller chunks that are easier to read and understand.
- is it harder for the optimizer to figure out a good access path. Sometimes. But usually not. Deal with performance as necessary!
- How does it work? IBM reference is good
- <https://www.ibm.com/docs/en/db2-for-zos/12?topic=statement-common-table-expression>
- There are good explanations... but just use them!

<https://www.ibm.com/docs/en/db2-for-zos/12?topic=statement-common-table-expression>

## Modern SQL – Common Table Expressions (2|2)

```
WITH CTE_DB AS (
  SELECT NAME AS DBNAME
  FROM SYSIBM.SYSDATABASE
  WHERE 1=1
  AND NAME NOT LIKE 'DSN%' -- IGNORE
  AND NAME NOT LIKE 'PTDB%' -- IGNORE
)
, CTE_RTS_TS AS (
  SELECT DBNAME, 'TS' AS OBJ_TYPE
  , SUM(SPACE) AS SPACE
  FROM SYSIBM.SYSTABLESPACESTATS
  WHERE 1=1
  AND DBNAME IN (SELECT DBNAME FROM
    CTE_DB)
  GROUP BY DBNAME
)
, CTE_RTS_IX AS (
  SELECT DBNAME, 'IX' AS OBJ_TYPE
  , SUM(SPACE) AS SPACE
  FROM SYSIBM.SYSINDEXSPACESTATS
  WHERE 1=1
  AND DBNAME IN (SELECT DBNAME FROM
    CTE_DB)
  GROUP BY DBNAME
)
```

DBNAME	TS_SPACE_MB	IX_SPACE_MB	ALL_SPACE_MB
~ALL	1905292	1321737	3227029
DYR01P	515004	559251	1074255
DIL01P	523263	189371	712635
DYB01P	119192	163476	282668
DIL04P	173401	64985	238386
DJC01P	98064	122092	220157
DCL01P	128668	54434	183103
DYD01P	90056	505	90562
DDB01P	59104	26476	85581
DCL04P	53501	26396	79897
DYC01P	34623	16507	51130

```
, CTE_F AS (
  SELECT COALESCE(Z.DBNAME,X.DBNAME) AS DBNAME
  , COALESCE(Z.SPACE,0) AS TS_SPACE
  , COALESCE(X.SPACE,0) AS IX_SPACE
  , COALESCE(Z.SPACE,0)
  +COALESCE(X.SPACE,0) AS ALL_SPACE
  FROM CTE_RTS_TS Z
  LEFT OUTER JOIN CTE_RTS_IX X
  ON (Z.DBNAME = X.DBNAME)
)
--SELECT * FROM CTE_F; -- UNCOMMENT TO SEE
CTE_F
SELECT COALESCE(DBNAME,'~ALL') AS DBNAME
, SUM(TS_SPACE)/1024 AS TS_SPACE_MB
, SUM(IX_SPACE)/1024 AS IX_SPACE_MB
, SUM(ALL_SPACE)/1024 AS ALL_SPACE_MB
FROM CTE_F
GROUP BY GROUPING SETS ((DBNAME),())
ORDER BY ALL_SPACE_MB DESC
```

*The report requirement is complicated (and useful). The SQL looks scary. But CTE makes each section of the SQL easy to understand! (IMHO)*

Here is a SQL which produces a fun-to-me report on db allocated space size  
I use many CTE to break up this big report requirement into small chunks. Each chunk is easy to understand

Build a list DBNAME to look at

Use RTS to SUM SPACE in that DB for TS and IX

Join the two CTE together to get space unified by dbname.

Use coalesce to turn nulls to appropriate value

Do a final grouping by grouping sets on the final cte, cte\_f, to get the sums for all!

I could have included tracks -> I like knowing tracks because of DSLIST

IMHO – in my humble opinion

# Modern SQL – LISTAGG

- Db2 V12 FL501 introduced LISTAGG function. I think it is fun.

- It allows one to turn a result set sideways!
- [Link: LISTAGG - IBM Documentation](#)
- [LINK TO IDUG CONTENT BLOG: SQLTricks - Part 2 LISTAGG Function \(idug.org\)](#)
- Basically, use LISTAGG with GROUPBY to report on column(s) and put other columns sideways as a list.
- It allows one to make reports using SQL
- Select indexnames and index columns as the list
- Select workdept and employee lastnames as the list

```
SELECT WORKDEPT,  
LISTAGG(ALL LASTNAME, ', ')  
  WITHIN GROUP(ORDER BY LASTNAME)  
  AS EMPLOYEES  
FROM EMP  
GROUP BY WORKDEPT;
```

WORKDEPT	EMPLOYEES
A00	HAAS, HEMMINGER, LUCCHESI, O'CONNELL, ORLANDO
B01	THOMPSON
C01	KWAN, NATZ, NICHOLLS, QUINTANA
D11	ADAMSON, BROWN, JOHN, JONES, LUTZ, PIANKA, SCOTT
D21	JEFFERSON, JOHNSON, MARINO, MONTEVERDE, PEREZ,
E01	GEYER
E11	HENDERSON, PARKER, SCHNEIDER, SCHWARTZ, SETRIGLI
E21	ALONZO, GOUNOT, LEE, MEHTA, SPENSER, WONG

[LISTAGG - IBM Documentation](#)  
[SQLTricks - Part 2 LISTAGG Function \(idug.org\)](#)

## Best Practice – ALWAYS use ORDER BY

- If there is any chance that your SELECT result may return more than 1 row, then you should always specify ORDER BY

- Sure, today, by coincidence, you may always get the result in the desired order.

- But the fact that result happens to usually/always return the desired order TODAY is not a guarantee for the future!

- Why could order of a result change?

1. Indexes could be changed/added/removed/deleted and then the optimizer picks a new access path and the result is returned in a different order.
2. The static statistics could change significantly in the future. Again, this may influence the optimizer to pick a new access path.
3. maybe static STATISTICS changed or maybe Db2 optimizer changes to be more "clever".

If you do not have ORDER BY then explain to me why not.

> And document it. Because it is not normal.

- Why do I care about this? As you might expect. Once (a few years ago) we had some obscure SQL that returned a result. No order by on the SELECT. Coincidentally, it always returned in desired order (for many many years). After a Db2 release and some rebinds, the access path changed. The result might have been returned slightly faster, but in a new more random order and the application updated things out of sequence. A bit of a disaster

## Opinion – never use DISTINCT

- Never use DISTINCT. I have rarely met a DISTINCT that I liked!
- SELECT DISTINCT causes Db2 to SORT the result set to remove duplicate rows in the result set.
- You may as well do a GROUP BY and accomplish the same final result set. GROUP BY does essentially the same SORT as DISTINCT. And then you can get the bonus of some classic GROUP BY functions like MAX(colx) or COUNT(\*)
- My problem when I see SELECT DISTINCT is that the SQL developer is getting results with duplicate rows, and they do not understand why and they use DISTINCT to remove them. But better to understand why! For example, they forget to look at rows with a status code of inactive. Or they include some other expired or past due effective date rows from the source. IF they investigate properly, they can change the SQL properly and not use DISTINCT.

## Modern SQL – UNION and UNION ALL

- When we have two result sets (two SELECT) that we want to concatenate together we have the choice of UNION and UNION ALL
- There is too much usage of UNION instead of UNION ALL
- UNION will combine the two sets and then sort and remove duplicates
- UNION ALL will combine the sets and be done. Duplicate rows could remain. Perhaps we want duplicate rows. Or perhaps we know duplicates are impossible > so we use UNION ALL and avoid that extra SORT to look for duplicates.

## Modern SQL – UNION, EXCEPT, INTERSECT

- We all know UNION. But there are other SET operations! ***Did you forget?***
- EXCEPT will take the top/first result set and “subtract” the identical rows from the second result set
- INTERSECT will take the top/first result and return only the rows that are common with the second result set.
- These additional SET operations are powerful ways to produce useful info or reports just with SQL without additional reporting languages
- <https://www.ibm.com/docs/en/db2-for-zos/12?topic=queries-fullselect>
- *I used EXCEPT in an interesting case. I built a result set containing all column names and types of tables in schema A. Another result had the same thing but for schema B. I then used EXCEPT and the result was the empty set! This quickly confirmed the two sets were identical.*



## Modern SQL – use SQL to build more SQL or commands (1|3)

- In the past, I would need to extract data from Db2 tables (catalog or application) and then use other tools to format the data to look like specific input to a subsequent step in my process.
- The subsequent step could be Db2 commands OR more SQL or input to some application program
- Now I realize it is not so hard to use SQL itself to extract the data and format it immediately. A generous helping of CTE and CHAR functions and concatenation can together produce a final result set that can meet most subsequent needs!

## Modern SQL – use SQL to build more SQL or commands (2|3)

- For example, I had a need to BIND all recently used packages from one collection into another collection. I wanted to use the same BIND parameter values from the original collection. The result set has to be a simple 80 CHAR wide field so that I could use DSNTIAUL with the SELECT and send the 80 LRECL dataset to the subsequent step that processed the Db2 BIND commands

```
with cte_a as (  
  select COLLID, NAME, RELBOUND  
    , OPTHINT, OWNER, CREATOR,  
      QUALIFIER, RELEASE  
    , VALID, OPERATIVE  
    , REOPTVAR  
    , LASTUSED  
  from SYSIBM.SYSPACKAGE  
 where 1=1  
       and COLLID = 'PCLD1'  
       AND LASTUSED > (CURRENT DATE - 26  
MONTHS)  
       AND OPTHINT = ' '  
  ORDER BY COLLID, NAME  
  FETCH FIRST 999 ROWS ONLY  
)
```

```
, CTE_F AS (  
  select ' BIND PACKAGE(NEW_RELEASE_COLL)  
MEM(' ||STRIP(NAME)||') -' AS BIND_TXT  
    , COLLID, NAME, 10 AS LINE_NBR  
  from cte_a  
  UNION ALL  
  select '  QUAL(' ||STRIP(QUALIFIER)||')  
OWNER(' ||STRIP(OWNER)||') -' AS BIND_TXT  
    , COLLID, NAME, 20 AS LINE_NBR  
  from cte_a  
  UNION ALL  
  select '  VALID(B) EXPL(Y) ACTION(REPLACE)'  
AS BIND_TXT  
    , COLLID, NAME, 30 AS LINE_NBR  
  from cte_a  
)  
SELECT CHAR(BIND_TXT,80) AS BIND_TXT80  
FROM CTE_F  
ORDER BY COLLID, NAME, LINE_NBR  
;
```

## Modern SQL – use SQL to build more SQL or commands (3|3)

- As you can see, the previous SQL can produce a nice-looking CHAR 80 wide result set that is easily fed into the step for Db2 batch commands

```
BIND_TXT80
-----
BIND PACKAGE(NEW_RELEASE_COLL) MEM(CFE0031) -
  QUAL(DBCCLD1) OWNER(DBCCLD1) -
  VALID(B) EXPL(Y) ACTION(REPLACE)
BIND PACKAGE(NEW_RELEASE_COLL) MEM(CFE0100) -
  QUAL(DBCCLD1) OWNER(DBCCLD1) -
  VALID(B) EXPL(Y) ACTION(REPLACE)
BIND PACKAGE(NEW_RELEASE_COLL) MEM(CFE0142) -
  QUAL(DBCCLD1) OWNER(DBCCLD1) -
  VALID(B) EXPL(Y) ACTION(REPLACE)
BIND PACKAGE(NEW_RELEASE_COLL) MEM(CFE0151) -
  QUAL(DBCCLD1) OWNER(DBCCLD1) -
  VALID(B) EXPL(Y) ACTION(REPLACE)
BIND PACKAGE(NEW_RELEASE_COLL) MEM(CFE0152) -
```

# Modern SQL – RID function

- The RID function returns the location of a row in a table in RID (BIGINT) format.
  - SQLCODE -803 failed insert duplicate because row already exists – reports the RID of the row already inside the table
  - Db2 Utilities such as CHECK DATA and LOAD DATA report conflicting rows inside the table with the RID of row.
- Now with RID function, you can confirm and double check what is at that RID location. Then you can compare with your INSERT or utility and understand really what went wrong!
- *The RID points to a record location inside the table(space). After a REORG or INS/UPD/DEL, the row at that RID location could move. The RID does not belong to a row!*
- The annoying thing is that Db2 -803 and utilities report RID in HEX and RID function requires BIGINT. Windows calculator is an easy way to convert. Other methods exist too. Or vote for my RFE: <https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1455>

```
-- LOOK AT EACH ROW
-- SHOW EACH RID
SELECT RID(E) AS RID, E.*
FROM EMP E
ORDER BY 1
;
-- KNOWING A RID(AS
BIGINT)
-- , LOOK AT THAT ROW
SELECT E.*
FROM EMP E
WHERE 1=1
AND RID(E) = 4883
```

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1455>

# Modern SQL – temporary tables

- First, temporary tables are NOT temporal tables!
- Temporary tables are declared or created (in advance, by a dba). They are private and are used during the life of your application activity.
- Both have their place and can allow SQL alone to produce complex reports or store results for (slightly) later.
- IDUG content has a blog on the topic:  
<https://www.idug.org/blogs/brian-laube1/2023/04/23/temporary-tables-and-painful-lesson>

## Modern SQL – temporal tables (1|2)

- First, temporal tables are not temporary tables!
- Temporal tables allow you to query the table and see the result as of a point in time in the past. Powerful!
- Basically, you create a duplicate “history” table of the original and associate the history table with the original. All DEL/UPD are moved to the history before they occur in the original. You can query the history table directly or the parent/original and specify what time.
- I use temporal tables on some key application tables about clients. Now myself or support team can query the history table and EASILY understand changes over time. Especially, who set colx to value Y? When? And what program? Easier than log analysis!
- See IDUG Content blog on this topic:
- <https://www.idug.org/blogs/brian-laube1/2022/03/15/travel-through-time-with-db2-sq>

## Modern SQL – temporal tables (2|2)

- A universal example of temporal tables are the REAL-TIME-STATISTICS tables of SYSTABLESPACESTATS and SYSINDEXSPACESTATS
- IBM has made it “easy” to turn them into temporal tables. They provide the DDL and all the instructions and the JCL. **You just have to think about potential purge strategy on the history table.**
- > The result is very useful to review the history of table change (growth) over time!
- <https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-systablespacestats>
- <https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-sysindexspacestats>
- *As an exercise for myself, one day I may figure out how to summarize the history.. Into daily chunks into another table? Or just within the history table itself. So then I can purge old history but keep it longer*

<https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-systablespacestats>  
<https://www.ibm.com/docs/en/db2-for-zos/12?topic=tables-sysindexspacestats>

# Modern SQL – find lost EXPLAIN from package

- Have you lost your SQL access path from plan\_table (or maybe you forgot to create it in the first place)
- How can you confirm your access path actually being used by your package today?
- You can (since v10) use EXPLAIN PACKAGE to extract the access path from the package itself and stuff the explanation into the plan\_table.
- And once it is in the plan\_table again, you can use Visual Explain
- OR you might want the plan\_table to tweak it and use it with OPTHINTS in another schema! (to recreate the problematic SQL)

•<https://www.idug.org/blogs/brian-laube1/2023/04/23/explain-package-to-recreate-missing-plan-table>



## Modern SQL – MERGE statement

- Do not forget that SQL MERGE exists.
- SQL MERGE lets you combine two results/tables.
- The target can be inserted or updated or even deleted based upon the data in the second table!
- The classic initial example is that you can set up a table that contains data you want to change in the target. The MERGE can process the whole table and when it matches the target key, the MERGE does the update, if the key is not found then the MERGE does INSERT.
- This MERGE method allows one to avoid the classic problem of wanting to update or insert into a target but you don't know if the row already exists. In the past, one would try INSERT and if it failed then do UPDATE (or opposite). Or do SELECT to check first and then do INSERT/UPDATE. All required multiple SQL. Now MERGE is one statement

## Best Practice – SQL Formatting (1 | 2)

- Shops should have SQL standards. At minimum, everyone has their personal preferences for how SQL should look. When you get ugly SQL, you should have a way to quickly reformat so it is “easier” to look at.
- Most “database exploration” tools like IBM Data Studio, Dbeaver, DBVisualizer, AQT and even VSCODE with the extension for Db2 Z, they all have SQL reformatting capability. Use it
- Minor digression. Dbeaver is an open source “database exploration tool”. Like Data Studio and DBvisualizer and others. But it is free and open source and I think I like it. Easy install! (as opposed to Data Studio)

<https://extendsclass.com/sql-formatter.html>

<https://www.freeformatter.com/sql-formatter.html#ad-output>

## Best Practice – SQL Formatting (2 | 2)

There are also a gazillion websites for programmers that will “format” SQL for you. Including:

- <https://extendsclass.com/sql-formatter.html>
- <https://www.freeformatter.com/sql-formatter.html#ad-output>

***The problem with all existing formatting tools and websites, None of them follow all MY standards. And they probably do not follow your preferences.***

<https://extendsclass.com/sql-formatter.html>

<https://www.freeformatter.com/sql-formatter.html#ad-output>

# IBM Cloud and the free Db2 (lite)

IBM cloud provides a full and free Db2 LUW in their IBM Cloud!

It is the very latest Version 11.5. Fully functional

The “lite” plan from IBM Cloud is free! Great price.

> It is limited to one userid and 200GB of data.

> the “lite” plan is only available from IBM Cloud of Dallas or London

**The benefit? You can play with Db2 and experiment yourself at home and practice your SQL!**

Dan Luksetich provides a quick overview on how to set up this Db2!

<https://www.db2expert.com/db2expert/db2-lite-on-cloud/>

The IBM Content Committee has an article (Fall 2023) by myself. My article has painfully excessive details and screenshots on how to set up this Db2!

## Best Practice – Requests for Enhancements (RFE)

- RFE are how we ask IBM to enhance Db2 (or other software).
- <https://ideas.ibm.com/>
- RFE are not for reporting bugs! RFE are for serious suggestions. Be reasonable though!
- Join and review IBM RFE website today. Vote for ideas that you like

<https://ideas.ibm.com/>

## Best Practices – example RFE (please vote)

- Put useful statistics about index compression in Db2 catalog  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1065>
- Declare and INSERT into DGT in one statement (like every other RDBMS in the world)  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-940>
- Redirected recovery – allow schema comparison with no recovery into target  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1402>
- Redirected recovery – eliminate need for annoying REPAIR CATALOG  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1464>
- Allow LISTAGG to always work with ORDER BY  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1031>
- New function HEX2BIGINT to allow easier investigation into INSERT SQLCODE -803 and utilities who report RID in HEX. RID function accepts BIGINT  
<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1455>

Put useful statistics about index compression in Db2 catalog

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1065>

Declare and INSERT into DGT in one statement (like every other technology in the world)

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-940>

Redirected recovery – allow schema comparison with no recovery into target

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1402>

Redirected recovery – eliminate need for annoying REPAIR CATALOG

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1464>

Allow LISTAGG to always work with ORDER BY

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1031>

New function HEX2BIGINT to allow easier investigation into INSERT SQLCODE -803 and utilities who report RID in HEX. RID function accepts BIGINT

<https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1455>



The slide features a dark blue background with a large, stylized circular graphic on the left. The graphic is composed of several overlapping circles and triangles in shades of blue, creating a geometric pattern. In the center of this graphic, the text "IDUG" is written in large, white, sans-serif capital letters. Below "IDUG", the text "2023 EMEA Db2 Tech Conference" is written in a smaller, white, sans-serif font. In the top right corner, the IDUG logo is displayed, consisting of a small blue circle with a white "IDUG" text inside, followed by the text "IDUG" in large, white, sans-serif capital letters. Below the logo, the text "Leading the Db2 User Community for 35 Years" is written in a small, white, sans-serif font. On the right side of the slide, the following information is listed:

Speaker: Brian Laube  
Company: Manulife Financial

Email Address:  
> brian.laube @manulife.ca  
> brian@spufi.ca or db2@spufi.ca

Session Code: G03

 Please fill out your session evaluation!

In the bottom left corner, a small white Twitter bird icon is followed by the text "#IDUGDb2".

Yes. I own the domain "spufi.ca". I find it amusing.