Temporary Tables in DB2 SQL Using for Effective SQL

Brian Laube

Manulife Financial

Session code: F06

06/04/2019, 09:20

Platform: DB2 for z/OS

IDUG
Leading the Db2 User Community since 1988

Welcome to my presentation on TEMPORARY TABLES

I will use the different types of temporary tables... And some examples of how to effectively use temporary tables for different SQL coding requirements

And importantly, I will review some important lessons I have learned!

Agenda

- Introduction
- DGTT: DECLARED GLOBAL TEMPORARY TABLE
- CGTT: CREATED GLOBAL TEMPORARY TABALE
- A real life example of mis-use of DGTT vs CGTT
- Use cases for DGTT vs CGTT
- A simple example with sample code of using DGTT vs CGTT
- Other temporary concepts in DB2 SQL:
  - Common table expressions (CTE)
  - Derived tables – named tables
  - Table function

Welcome to my agenda

Introduction: What is a Temporary Table (1|4)

- A temporary table (in most RDMS) is a place to keep data for the duration of some application process. The data is basically private and only available to the application user.
- The typical end-user of a temporary table ranges from ad-hoc dynamic end-users (like you or me) up to real application programs (native SQL stored procedures or COBOL).
  - Temporary tables are used in both dynamic and static SQL.
- There are two types of temporary table:
  - DECLARED GLOBAL TEMPORARY TABLE (DGTT)
  - CREATED GLOBAL TEMPORARY TABLE (CGTT)

3

What is a temporary table? In a RDMS sense. Very useful for creating SQL scripts (independent of a progrmmin language). Handy to temporarily store SQL result sets for later reference.

And to be clear… the temp table contents are private to the user!

And to expand. Two types

DGTT – declared by anyone. In the moment it is needed! Declared and used and then it disappears! Forgotten by DB2!

CGTT – created in advance. It is just used at the moment it is needed. DB2 knows the layout! Because it was created in advance!

The original purpose of this whole presentation is to outline the pros and cons of each!

Introduction: What is a Temporary Table (2|4)

- A temporary table is a place to store relevant to your needs. Whether you are an application or some end-user. It is a place to keep data.
- It helps give YOU some SQL coding options.
- Temporary Tables can help reduce SQL complexity! You can break down potentially complex and confusing and large SQL into smaller SQL chunks via temporary tables.

4

In a sense, by breaking big SQL into multiple smaller SQL referencing a temporary table.
Hopefully using the temporary table and having several separate SQL then you make the overall SQL functionality easier to understand.
And you have some control of access path… you almost force or encourage DB2 to go in the order you specify because of your SQL chunks.

Introduction: Examples of use of Temporary Tables (3|4)

- a high-volume online transaction might need to hold working data in a temporary table for a short time to make the transaction logic simpler
- a low-volume batch reporting application might have complex reporting requirement which might be simply implemented by using a temporary table
- and sometimes we end users (DBA or developer or support) have some totally dynamic and ad-hoc complex query requirement where a temporary table provides a simple way to hold data to make a multi-step process easier.
  - A temporary table can be used dynamically (in SPUFI, batch DSNTEP2, or workstation tools like Data Studio) to hold data and make very complex almost procedural queries to a series of tables. This can help YOU avoid writing a program! Very handy

5

The IBM knowledge center provides excellent documentation on HOW to use temporary tables. This article summarizes the HOW but emphasizes, the WHEN to use and in what scenario. And importantly, the real-life example below highlights the pros and cons of both types of temporary tables.

Introduction: What is a Temporary Table (4|4)

Other types of temporary tables or concepts in DB2 SQL:

- COMMON TABLE EXPRESSIONS (CTE)
- derived tables

6

I include these types of "temporary tables" to complete my conversation on this topic. Really, temporary tables should be just the two types of (DB2) objects that I will mostly discuss here.

But sometimes people include or confuse CTE and Derived Tables… so I give a few words in this presentation. And in a sense, they are super-temporary too… so they do count!

I suppose I could or also should take about TABLE functions… but I don't use them much…

CGTT – CREATED GLOBAL TEMPORARY TABLES (1|3)

- As always, the best reference is the knowledge center (link in notes)

- The CGTT is created once (often by the DBA) in advance… and then other (applications) can use it. Authorization to create a CGTT is required!
- Each application has its own instance of created temporary table while it is using it! The data is private to the user of the instance of the CGTT.
- A CGTT instance exists when an application uses it by referencing it in OPEN, SELECT, DELETE or INSERT. Most likely you will use INSERT to stuff it with data the first time!
- The instance of the temporary table will exist until the application process ends OR The remote server connection terminates, OR the unit of work completes. ROLLBACK or COMMIT.
  - With COMMIT, Db2 will delete the instance of the temporary table UNLESS it is referenced by an OPEN cursor that was defined 'WITH HOLD'.

https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/sqlref/src/tpc/db2z_sql_createglobaltemptable.html

CGTT – CREATED GLOBAL TEMPORARY TABLES (2|3)

- Each application schema/database/environment has its own version of the CGTT. Remember, they are created in advance! The same CGTT name could theoretically be different in different schemas. The CGTT could change over time like any other table. You would migrate the changes up as the CGTT changes.

CGTT are created in advance! Used in some later moment! As always, the data inside the temporary table is private to the user!

CGTT – CREATED GLOBAL TEMPORARY TABLES (3|3)

- DB2 does not perform locking and locking operations for CGTT
- You cannot use MERGE with CGTT.
- You can use DELETE ... but without WHERE clause. Basically, you can only use DELETE to empty the CGTT
- No UPDATE of Temporary Table.
- Basically, use INSERT to get data inside. You can use SELECT INTO.
- SELECT is used to query the CGTT. Or OPEN cursor
- No index allowed on CGTT.
- If you are truly finished with a CGTT and no application program or end-user uses it anymore… then you must DROP TABLE to get rid of it!

Many subtle details about CGTT. For best resource, use the knowledge center and read the official documentation!

This presentation is really to describe WHEN to use TT (not the HOW > for that … read the documentation … the documentation is purposefully not clear on WHEN we use temp tables)

DGTT – DECLARED GLOBAL TEMPORARY TABLES (1|2)

- As always, the best reference is the knowledge center (link in notes)

- The DGTT is defined or declared as it is needed by an application or end-user. No special authorization is required to use DGTT.
- The DGTT is not stored in the DB2 catalog. It is not persistent. It cannot be shared. You DECLARE and then you use it!
  - If some other process/person wants to use the 'same' DGTT then they must declare it when they need it!
- The DGTT is unique to the application/user that defined it.
  - Theoretically, multiple parallel users could define the same DGTT with the same name… but they only use their own DGTT!
- Subsequent references to the DGTT name must use "session" as the schema.
- When the application process terminates, the temporary table is implicitly dropped.

https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/sqlref/src/tpc/db2z_sql_declareglobaltemptable.html

DGTT – DECLARED GLOBAL TEMPORARY TABLES (2|2)

- A DGTT can have zero or one or many indexes.
- A DGTT allows UPDATE.
- On COMMIT the DGTT can be auto-emptied (optionally, also drop the DGTT) OR on COMMIT, the DGTT can preserve the ROWS.
- A DGTT can be logged or not logged (as of V11). If not logged, then on rollback it can delete the rows or preserve the rows.

As with CGTT, the DGTT has many subtle characteristics that you can consider. Read the documentation to be clear.
The concept is pretty simple… but as always.. Read the knowledge center to be very clear on exactly what is possible!

Again, my presentation is to go over WHEN to use DGTT

An example of mis-use of DGTT vs CGTT

- At a high level both types of temporary tables provide a simple temporary table functionality for simple SQL needs.
- And at a super-high level, DGTT seems much easier than CGTT.
- And some people might even say DGTT are “better”
  - although one should always remember… “it depends”).
- No DBA required for DGTT. DGTT allows indexes. You can UPDATE a DGTT. What is not to love?

At first blush… DGTT seem better (compared to CGTT)

An example of mis-use of DGTT vs CGTT

- The simplest examples of using a temporary table are ad-hoc dynamic queries that a DBA might come up with at his desk.
- And in that case, the DBA might use DGTT because they are easy and can be re-done over and over until the SQL script is perfected.
- And then the DBA might tell a bunch of his favorite developer-buddies about DGTT and CGTT. With an emphasis about how DGTT is pretty cool and easy for said developers to use! No need to bother the DBA. The DBA likes that.

All the initial examples that one can think of using a TT are best suited by using DGTT.

A classic reason to use DGTT is to dynamically gather lots of data into one result set -> temporary table. And then query the DGTT multiple times to produce multiple reports or views of the same data!

I do this all the time!

So this how DGTT became used in my shop

An example of mis-use of DGTT vs CGTT

- And then a few years might past. The DBA continues to use DGTT but is unaware how much the developers are using temporary tables in applications. As far as the DBA knows, the usage of temporary tables is limited and non-existent by the applications and developers.

- One day, an exciting and new application is developed in-house and implemented. It goes to prod. It is a futuristic and modern application almost entirely written with DB2 native SQL stored procedures.

This explains how DGTT snuck into a new production application

An example of mis-use of DGTT vs CGTT

- After a few days, we realized that the CPU life was being sucked out of our mainframe.
- Some investigation showed the culprit was this new application.
- Further investigation quickly led to the new and high-cost procedures
- A quick review of these top 3 or 5 procedures showed source code with some less-than-ideal-and-crazy-complex SQL (stupid-stupid SQL) along with a reliance of DGTT.
- Each of these high-cpu/high-resource procedures used a DGTT used to gather data from various application tables.
  - They stuffed the data into the DGTT (and counted the rows inside the DGTT) and then returned the contents of the DGTT to the caller (via open cursor).

15

Investigation was done with a combination of online monitor thread history PLUS my DB2 performance database. The investigation was quick .. The culprit was easily identified.

What is “stupid-stupid” SQL? Hand-coded SQL that went for several scrolling pages…. Using tons of joins .. Inner and outer… and derived tables.

Technically… they did not need to count(*) after inserting into the DGTT. They should have used SQLCA information to find how much was inserted into the DGTT. A bit of lazy programming there.

An example of mis-use of DGTT vs CGTT

- Apparently, the project got through “acceptance testing” with no complaints. They never asked the DBA to review code or application performance measurements.
- In “acceptance” the response time and cpu time were well under a second. It seemed fine. People hit enter and the response came back promptly. BUT, the math would say that doing the same thing thousands of times per day would add up to something significant. No one did the math.
- Now we needed to do something.

16

One second of elapsed wall clock time was “okay” to the acceptance end-users.

No one thought it through. This was not super high volume. But one of the procedures might be executed 10000 times per day. The overall cost was surprisingly high!

The Cons of DGTT (especially in this case)

- The DGTT is an extra SQL statement. It must be executed every time it is needed.
- The DGTT could be inside a static program (COBOL or native SQL stored procedure). The layout of the DCGTT is not really known (because it is not guaranteed) at CREATE PROCEDURE or BIND PACKAGE time. Therefore, importantly and expensively, those SQL that use the DGTT must be dynamically bound at execution time for DB2 to figure the optimal access path.
- Together, these two little things add up.
  - We had many procedures that used this concept.

Why was the DGTT expensive

I have other COBOL based application that use DGTT. In hindsight, they should be using CGTT. But this Cobol batch report program runs once a month. So let them use DGTT

> To be honest, I think a CGTT would have been “easier” for the application programmer to code… but that is just me

The Pros of DGTT (in this case)

- The DBA created a CGTT. The procedure was modified to use the CGTT instead of the DGTT. The application logic did not change at all. The actual SQL was not changed, although it could have used some review and possible, that was beyond the immediate scope.
- Switching to CGTT resulted in a 96% reduction of CPU in this one procedure. It went from something noticeable to something almost costless.
- The rest of the high cost and high volume procedures using DGTT where changed to CGTT and the overall application CPU cost went from hours to reasonable minutes. Thankfully.

And now we switched our application to use CGTT instead of the DGTT.

The program change was trivial. Something needed to change… so of course we validated and tested. But logically, it was the exact same SQL. The INSERT was into a CGTT, not a DGTT. The result had to be identical (and it was identical).

Comparison of procedure after change from DGTT to CGTT

DT	SSID	PCK_ID	PLAN_NAME	CPU_MIN	CPU_MAX	CPU_AVG	PKG_ALLOC	SQL	CNT
03-01	**PDBB**	**SRCH_FOR_PRVDR**	**YTCIRRP1**	**1.2**	**0.193633**	**0.005655**	**147057**	**248937**	**13315**
02-28	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	1.2	0.150026	0.005225	154246	260650	14285
02-27	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	1.2	0.191723	0.005337	153097	260189	14147
02-26	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	1.2	0.135898	0.005248	161023	266483	14606
02-23	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	1.3	0.168243	0.005679	156703	264599	13953
02-22	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	30.6	0.684351	0.134640	160194	468635	13645
02-21	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	29.8	0.634492	0.125066	165294	462562	14301
02-20	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	28.2	0.609018	0.118179	156760	440831	14367
02-19	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	14.2	0.573073	0.099820	88111	239414	8538
02-16	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	29.3	0.598881	0.126219	158599	455446	13936
02-15	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	30.3	0.667659	0.125824	161636	467731	14484
02-14	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	28.5	0.580629	0.114303	162520	446162	14963
02-13	PDBB	SRCH_FOR_PRVDR	YTCIRRP1	29.7	0.664645	0.118736	162726	458787	15026

19

The PKG_ALLOC and CNT (count of accounting trace in the DB2 performance database) show that the procedure package usage remained relatively consistent over the relevant time period.

But the total CPU_MIN and CPU_AVG show the obvious drop in cost. The SQL went down because the procedure had fewer SQL in the procedure.

The CPU_MAX is a bit high because some odd input data cases would cause the SQL search to be expensive (that is another issue).

Lessons Learned for CGTT

In hindsight, the lesson learned, is that CGTT are well designed for

- small result sets (no need for index)
- being used over and again. Especially in a high-volume transactional application. CICS or WEB
- static programs
- The SQL access path referencing CGTT is fixed. You can know it in advance!
- The CGTT cannot be updated. But many typical transactional type users of temporary tables have no need for UPDATE of temporary table contents
- The CGTT is pre-defined. You just use it. And then it exists for you.
  - A DGTT requires an extra SQL statement to actually declare it!

In hindsight… lessons were learned… and they are always so obvious in hindsight…

Anyways… this lesson is what inspired me to create this presentation and share this lesson with YOU!!!!!!!

Lessons Learned for DGTT

And DGTT are best designed for

- Ad-hoc query or reporting.
  - DGTT can help break up complex reporting requirements into multiple smaller and easier-to-understand SQL steps
- Low-volume regular reporting (yearly, monthly or even daily > but probably not thousands of times per day)
- Large results (or small). But especially large.
- Results that require updates
- Results that would benefit from an index for later queries.
- A unique index can guarantee uniqueness
- Easy to use by anyone -> no need to bother the DBA

21

(beyond my comments in the slide) DGTT are best for ad-hoc and low-usage!
They are so simple for anyone to use!

Procedure using CGTT

```
CREATE PROCEDURE TT_TEST_CGTT
                        (OUT OUT_CNT_OF_ROWS_TABLE INTEGER)
    VERSION V1  LANGUAGE SQL  APPLCOMPAT V11R1 NOT DETERMINISTIC
             DYNAMIC RESULT SETS 1 -- QUANTITY OF OPEN CURSORS AFTER EXECUTION
             QUALIFIER DBCDBD1 -- FOR UNQUALIFIED DB2 OBJECTS (SUCH AS TABLES!)
             PACKAGE OWNER DBCDBD1 -- OWNER OF PROCEDURE ... FOR AUTHORITY
             ISOLATION LEVEL CS
    WLM ENVIRONMENT FOR DEBUG MODE DDBCDBG1 -- IF DEBUGGING, THEN debug here
    ASUTIME LIMIT 800000 -- REASONABLE LIMIT ON SERVICE UNITS (CPU)

P1: BEGIN
--  DECLARE CURSOR
DECLARE CUR1 CURSOR WITH RETURN FOR SELECT * FROM CGTT01 ;
-- INSERT INTO
INSERT INTO CGTT01 (TBCREATOR, TBNAME, NAME, COLNO)
SELECT TBCREATOR, TBNAME, NAME, COLNO FROM SYSIBM.SYSCOLUMNS WHERE RAND() < 0.50
FETCH FIRST 0010 ROWS ONLY;
-- COUNT ROWS IN
SELECT COUNT(*) INTO OUT_CNT_OF_ROWS_TABLE FROM CGTT01;
-- OPEN CURSOR
OPEN CUR1;

END P1#
```

```
-- the CGTT is created in advance
-- usually by the DBA..
CREATE GLOBAL TEMPORARY TABLE
DBCDBD1.CGTT01
( tbcreator varchar(128)
, tbname varchar(128)
, name varchar(128)
, colno smallint)
;
```

22

Here is a simple example of procedure using a CGTT

Procedure using DGTT

```
CREATE PROCEDURE TT_TEST_DGTT
                        (OUT OUT_CNT_OF_ROWS_TABLE INTEGER)
    VERSION V1  LANGUAGE SQL  APPLCOMPAT V11R1 NOT DETERMINISTIC
             DYNAMIC RESULT SETS 1 -- QUANTITY OF OPEN CURSORS AFTER EXECUTION
             QUALIFIER DBCDBD1 -- FOR UNQUALIFIED DB2 OBJECTS (SUCH AS TABLES!)
             PACKAGE OWNER DBCDBD1 -- OWNER OF PROCEDURE ... FOR AUTHORITY
             ISOLATION LEVEL CS
    WLM ENVIRONMENT FOR DEBUG MODE DDBCDBG1 -- IF DEBUGGING, THEN debug here
    ASUTIME LIMIT 800000 -- REASONABLE LIMIT ON SERVICE UNITS (CPU)

P1: BEGIN

-- DECLARE DGTT
DECLARE GLOBAL TEMPORARY TABLE DGTT01
( TBCREATOR VARCHAR(128)
, TBNAME VARCHAR(128)
, NAME VARCHAR(128)
, COLNO SMALLINT)
on commit drop table
;

--  DECLARE CURSOR
BEGIN
 DECLARE CUR1 CURSOR WITH RETURN FOR
  SELECT * FROM SESSION.DGTT01
 ;
-- INSERT INTO DGTT
INSERT INTO SESSION.DGTT01
(TBCREATOR, TBNAME, NAME, COLNO)
SELECT TBCREATOR, TBNAME, NAME, COLNO
FROM SYSIBM.SYSCOLUMNS
WHERE 1=1
  AND RAND() < 0.50
FETCH FIRST 0010 ROWS ONLY
;
-- COUNT ROWS IN DGTT
SELECT COUNT(*) INTO OUT_CNT_OF_ROWS_TABLE
FROM SESSION.DGTT01;

-- OPEN CURSOR
 OPEN CUR1;
END;

END P1#
```


Here is the another example. This procedure does the same function as the previous. But it uses DGTT

Comparison of simple procedures using CGTT vs DGTT

Insert 10 rows into the TT and count the rows and return the rows.
Call each procedure many (100+) times… afterwards… compare costs

- `call dbcdbd1.tt_test_cgtt() - avg CPU 0.0004`
- `call dbcdbd1.tt_test_dgtt() - avg CPU 0.0080`

The procedure with DGTT used 20 times more cpu.

- **This is meant to simulate a very simple and fast transaction that has a need for a temporary table. Small result set.**

And here is how I used each procedure and my measurements of results.

Each procedure is relatively cheap (by some standards). But the DGTT procedure is so much cheaper.
And this is where “math matters”.
If you call it N times per day… then do the math! **How much CPU will you use?!!!**

Comparison of more procedures using CGTT vs DGTT

Insert 999 rows into the TT and count the rows and return the rows.

- `call dbcdbd1.TT_TEST_CGTT999(?) - avg cpu 0.004`
- `call dbcdbd1.TT_TEST_DGTT999(?) - avg cpu 0.018`

The procedure with DGTT used 4.5 times more CPU

- **This is meant to simulate a medium complexity transaction that has a need for a temporary table. Large result set.**

25

This is another example of obviously similar (but slightly different) procedures. Here we insert 999 rows into the TT.

So the procedure is doing more SQL work

The INSERT is relatively more expensive then the previous example which only inserted 10 rows into the TT

And here the “savings” was only 4.5 times cheaper with CGTT

Basically, I am showing that the relative savings of CGTT vs DGTT in a high-volume procedure “depends” upon the actual SQL and what you are doing. There is no hard and fast rule about how much better one type of TT is compared to the other.

IF the INSERT took 100 CPU seconds… then the cost of the two procedures would be almost the same!

Basically, the cost of the DGTT is the extra DECLARE SQL statement! And then the dynamic bind of the DGTT (it is not static like CGTT).

CGTT vs DGTT

In the previous comparisons, it was shown that for moderately high-volume transactions using temporary tables, the CGTT is “cheaper” than DGTT.

- *CGTT should be first considered for static programs. High volume transactions*
- *DGTT should be considered for low volume.. Reporting…*
- BUT, there is no fixed rule for the benefit or cost of CGTT vs DGTT.

  As always, it depends.

26

Again.. It always depends

But as a default. Use CGTT in static and DGTT for reporting. Other cases may require a tiny bit of thought by the developer. So think about it! But it should not be hard. You don’t want to regret your decision later.

CGTT

- And of course… to be clear… you could use CGTT in some low-volume (monthly) reporting application. The CGTT will be moderately cheaper than a DGTT.
- And a CGTT would be static so you can more confidently review access path.
- BUT, a DGTT might be more appropriate because the usage is low and you could add indexes
  - Maybe a DGTT is used because it is "easy" for the programmer! If they thought about and have a reason for DGTT then I will probably be happy

27

CGTT do have their use and place

And it is interesting that DGTT allow indexes

- although… I have not often put index on DGTT. If the result set is typically (or always) small then the index won't help performance. If the DGTT contains many rows then it could be useful… but to be honest… I have often done it.
- My DGTT

DGTT – with indexes

- DGTT allow indexes. CGTT do not allow indexes
- Why put an index on DGTT?
  - A unique index can prevent duplicate rows. The user of the DGTT should have a reason to understand and control duplicates. Although, if you know your INSERT into the DGTT is "good" then don't put unnecessary unique index
  - If you later read a large DGTT then an index may help that later SQL performance! But maybe not necessary
- You can create the index on the DGTT before or after insert into it. There are pros and cons! It depends

28

Don't' go willy nilly adding indexes to the DGTT.

They seem useful.. But I found I don't use them much.

You can often ensure your INSERT into the DGTT is appropriately unique. But if you doubts and a need then use unique index.

And if the DGTT has many rows then indexes may help later SQL SELECT performance. But think about it. Often the subsequent involves a join of two tables (the DGTT and a real application table). And often you are producing a report. And it is a big report and something must be scanned.

For example, suppose the DGTT has pol_id as a logical key field. But if your big SELECT report off the DGTT is logically going to be a scan then you don't need an index on pol_id.

But if you join from an application table to the DGTT then maybe the index will help

It always depnds!

And finally, the index can be made before or after the INSERT into the DGTT. The create index has a cost. But the cost can be paid when you insert into it… or when you create the index after the DGTT has data.

EXTRA: DGTT and LOGGED vs NOT LOGGED (1|2)

- A DGTT allows you to optionally specify LOGGED vs NOT LOGGED (since V11)
- AS usual with DB2, the default is LOGGED. Logging is generally good.
- The NOT LOGGED will have a performance benefit when you populate the DGTT. You avoid writing to “log” and the CPU cost of writing to the “log”. The benefit is not as much as one might expect. Especially CPU. But it is something. But if it is many many rows then you will definitely save LOG space. You will avoid filling up the “log” with low-value log records.

I would only use NOT logged when the DGTT is going to be HUGE!!!

EXTRA: DGTT and LOGGED vs NOT LOGGED (2|2)

- An interesting caveat is about when the DGTT has a unique index.
  - If you get halfway through INSERT data into the DGTT and receive SQLCODE -803 for attempting to insert a row that will violate the unique index constraint, then the DGTT will be emptied if it is NOT LOGGED. If it is LOGGED, then the DGTT will continue to hold the previous rows. Be careful of this difference in behavior!
- *In general, one should use the default of DGTT of "LOGGED" unless one has a specific reason to use NOT LOGGED. Think about it!*

This is an example of how reading the fine-print of the documentation is important. But because this difference of LOGGED and NOT LOGGED behavior is not "obvious"

Reading the documentation and experimentation show the difference!

But again, use LOGGED as default or you have some other reason to not use LOGGED

Other "temporary" tables.. But only within SQL

- Derived tables
- Common table expressions (CTE)

- I want to bring up these concepts for my developers and end-users… to remind them about the power and options available in SQL (not just DB2 but most SQL RDMS)
- And in particular… CTE are a favorite of mine. But many forget about them! CTE became available in DB2 V8 (back in 2005). They are super!

31

I don't know why people don't use CTE more often!
Maybe they learned SQL before 2005?
Or the books they read are old.
Or people assume CTE is new so it must be complex.
But really… CTE are easy… and I find it makes SQL much more clear!

Derived table

- Derived tables are created on-the-fly within your SQL as named table expressions following a FROM statement

```
select rts_sum.dbname, rts_sum.tsname, rts_sum.totalrows
, t.creator as tb_creator, t.name as tb_name, bigint(t.cardf) as card_bigint
from (select dbname, name as tsname
            ,sum(coalesce(totalrows,0)) as totalrows
      from sysibm.SYSTABLESPACESTATS
      group by dbname, name
      ) rts_sum  -- this is my derived table called rts_sum
inner join sysibm.systables t
  on t.dbname = rts_sum.dbname and t.tsname = rts_sum.tsname
order by totalrows desc, t.cardf desc
;
```

32

Derived tables are fine… but sometimes you have to look carefully to find them! They are not obvious when reviewing SQL code

Derived tables have been around forever. Many people know of the idea (even if they call the concept something else – different name).

Common table Expressions (CTE)

- CTE are created on-the-fly within your SQL before the main SELECT.

```
with cte_rts as (
 select dbname, name as tsname
      ,sum(coalesce(totalrows,0)) as totalrows
 from sysibm.SYSTABLESPACESTATS
 group by dbname, name
)
select cte_rts.dbname, cte_rts.tsname, cte_rts.totalrows
, t.creator as tb_creator, t.name as tb_name, bigint(t.cardf) as card_bigint
from cte_rts inner join sysibm.systables t
  on t.dbname = cte_rts.dbname and t.tsname = cte_rts.tsname
order by totalrows desc, t.cardf desc
```

33

CTE became available with DB2 V8 in 2005.

Look how clear is the CTE. I think it is easy to read.

CTE vs Derived

- Both derived and CTE provide a temporary table of data for your query for the scope or life or your query
- Derived tables have been around for a long time
- CTE have been available since DB2 V8 (2005)

- My personal opinion. CTE are better than derived tables.
- Of course… it depends.. But I see too much usage of derived tables when a CTE would be subjectively 'better'

In general. The CTE and DERIVED table SQL concepts are interchangeable.
As you can guess… I tend to use CTE

CTE vs Derived

- CTE are obvious to the reader of the SQL. CTE are at the top!
- CTE are easy to read.
- The same CTE can be referenced multiple times in the same SQL (derived cannot)
- a subsequent CTE can reference a previous CTE (derived cannot do that)
- A CTE can be used in recursive SQL (derived cannot)

- All in all… CTE are my go to SQL concept to begin to break up large and complex SQL into something easier to understand

Almost all examples of how to create “recursive sql” use CTE

CTE referencing CTE

```
with cte_a as (
 select dbname, name as tsname
      ,sum(coalesce(totalrows,0)) as totalrows
 from sysibm.SYSTABLESPACESTATS
 group by dbname, name
)
,cte_b as (
select dbname, tsname, TOTALROWS
, case when totalrows > 10000000 then 'LARGE'
       when totalrows >   100000 then 'MED'
       WHEN TOTALROWS >        1 then 'SMALL'
       WHEN TOTALROWS = 0        then 'EMPTY'
       ELSE                           '????'
  END AS TS_SIZE
FROM CTE_A
)
SELECT coalesce(DBNAME,'-') as dbname
, coalesce(TS_SIZE,'-') as ts_size
, COUNT(*) AS CNT, SUM(TOTALROWS) AS TOTALROWS
FROM CTE_B
GROUP BY GROUPING SETS ((DBNAME, TS_SIZE),())
ORDER BY DBNAME, TS_SIZE DESC
;
```

DBNAME	TS_SIZE	CNT	TOTALROWS
-	-	6838	3488253358
ALADB01	EMPTY	1	0
AOCEPDB	SMALL	17	6239
AOCEPDB	EMPTY	40	0
AOCQTDB	EMPTY	2	0
CBROAM	EMPTY	1	0
DATBA8K	EMPTY	4	0
DB2TOOLS	EMPTY	1	0
DBMAPDB	EMPTY	1	0
DBPLANTS	SMALL	1	5
DBPLANTS	EMPTY	1	0
DCL01A	SMALL	137	1521387
DCL01A	MED	19	11786797
DCL01A	EMPTY	34	0
DCL01A	????	1	1
DCL01D	SMALL	118	1118991
DCL01D	MED	40	38678455

36

Here is an example of using multiple CTE in one query. And the second CTE references the first. I find it makes the query 'easier' to understand.

CTE_A queries the RTS table of systablespacestats to find totalrows for every tablespace (for all parts)

CTE_B queries the CTE_A to add a new column "ts_size" which is then referenced in the final real SELECT

And of course… I am using my favorite SQL (OLAP) functionality for "group by grouping sets". That is another big topic… but basically.. Grouping sets are great. Allow for basic reporting! Just use SQL -> reporting tool required!

CTE – using CTE to do recursive SQL

```
--  every index name (for a particular ixcreator)
--  and then have one wide column COL_LIST with every column name for that index
 with CTE_A as  (
 select IXCREATOR, IXNAME, COLSEQ, COLNAME
 from sysibm.SYSKEYS -- this DB2 catalog table has every index & column
 where 1=1
 and IXCREATOR = 'DBCYRD1'
)
-- now use another CTE (referencing previous CTE) and this recursive
 -- CTE recursively references itself!
 , CTE_RQUERY (N, IXCREATOR, IXNAME, COLSEQ, COL_LIST) AS  (
 select 1, B.IXCREATOR, B.IXNAME, B.COLSEQ
, varchar(B.colname,999) -- CAST the source column to larger type
                         --so concatenation has space to expand!
 from cte_a B
 WHERE B.COLSEQ = 1 -- prime the pump from source table!
 UNION ALL
 select N+1, T1.IXCREATOR, T1.IXNAME, T1.COLSEQ
   , COL_LIST ||','|| T1.COLNAME -- concat previous col_list w next col_name
 from CTE_RQUERY T0 -- recursive!!!
 inner join CTE_A T1 -- join back to source table for next
    on T0.IXCREATOR = T1.IXCREATOR
   AND T0.IXNAME = T1.IXNAME
   AND T0.COLSEQ + 1 = T1.COLSEQ
 where N < 999 -- control to stop possible infinite recursion (really to stop WARNING +347)
 )
```

```
-------------------------------------------------------------------
-- now (finally) select from the now complete recursively built CTE!!!
-- this CTE has the last/ultimate row for each ixname
-- > MAKE THIS YET ONE MORE CTE
, cte_idx_w_colnames as (
SELECT IXCREATOR, IXNAME, COLSEQ AS COL_CNT, COL_LIST as all_cols
 FROM CTE_RQUERY RQ
 WHERE 1=1
 AND RQ.COLSEQ = (SELECT MAX(RZ.COLSEQ)
                    FROM CTE_RQUERY RZ
                   WHERE RZ.IXNAME = RQ.IXNAME
                     AND RZ.IXCREATOR = RQ.IXCREATOR)
)
select ixcreator, ixname, all_cols
from cte_idx_w_colnames
where 1=1
order by ixcreator, ixname, IXCREATOR, IXNAME
;
```

IXCREATOR	IXNAME	ALL_COLS
DBCYRD1	X01AEC	POLICY_NUM,DIVISION_NUM
DBCYRD1	X01AEOO	POLICY_NUM,CERT_ID
DBCYRD1	X01CLAIM	CL_ID
DBCYRD1	X01CLASS	CON_ID,DIV_ID,CLASS_ID,BEN_TYP
DBCYRD1	X01CLG	CON_ID,CL_LST_GRP1_CD,CL_LST_GRP2_CD

37

This is my attempt to show how ‘recursive sql’ uses CTE to do “recursion”.

This particular example is a ‘simple’ example of building a list of index names with all the column names in one wide column

Not really CTE or temp tables… but V12 501 function LISTAGG replaces "complex" recursive SQL

```
-- confirm the current application compatibility
-- >> it must be V12RM501 or higher to use LISTAGG
--  >> if necessary set the special register to that value (if possible) before using LISTAGG!
select current application compatibility from sysibm.sysdummy1;
-------------------------------------------------------------------------------
-- use LISTAGG instead of RECURSIVE SQL to produce same result set as above!
-- for each index, use LISTAGG to prepare list of all colnames (as one wide columnn)
SELECT IXCREATOR, IXNAME
       ,LISTAGG(ALL COLNAME,',') WITHIN GROUP (ORDER BY COLSEQ)  AS ALL_COLS
FROM SYSIBM.SYSKEYS
WHERE 1=1
  AND IXCREATOR = 'DBCYRD1'
GROUP BY IXCREATOR, IXNAME
;
```


I think LISTAGG is fun and cool.
It replaces many/some recursive SQL using CTE

(same result as previous slide… no need to show again here)

What I would like to see: DGTT usage enhancement (1|3)

- To use a DGTT you must DECLARE it in your SQL script with column names and types and definition.
- And then, you INSERT into your DGTT and then use it.
  - ***Always two steps. DELCARE .. Then INSERT into it***
- It would be nice if you could just insert into a new DGTT name and have DB2 figure out the columns and type and size based upon the result set.
  - This would save you the hassle of thinking about how the details of the column.
    - *Isn't this idea logical? Reasonable? Obvious? Why not?*

What I would like to see: DGTT usage enhancement (2|3)

- If you read the DGTT documentation… it suggests this is idea *ALMOST* possible.
- You can DECLARE a DGTT using a SELECT to define the columns. But then you MUST include syntax WITH NO DATA

```
DECLARE GLOBAL TEMPORARY TABLE DGTTB
AS (SELECT * FROM SYSIBM.SYSTABLES)
WITH NO DATA;
```

  - Interesting usage note. The 'fullselect' that you provide does not need a where clause. You could have where clause… but you don't need it. DB2 is only looking to figure what are the columns and types in your result for deciding the columns in the DGTT
  - BUT.. You could have the WHERE clause. The 'fullselect' that you provide could be identical to the later 'fullselect' you use for the insert into the DGTT. Having the whole exact same 'fullselect' in the declare might make the SQL script "simple". The DGTT itself would be large.

40

IF you use DGTT with 'fullselect' then you don't need WHERE clause

You could use the exact same SELECT you are about to soon/later use to insert into the DGTT.

The obvious drawback is that if the SELECT is big and long and ugly then you see it twice in your SQL script. Once in the DGTT and again the INSERT into DGTT.

But then again.. It is nice and easy if you know it is the exact same SELECT

What I would like to see: DGTT usage enhancement (3|3)

- If you read the DGTT documentation… it suggests this is ALMOST possible.
- **Instead of 'WITH NO DATA' it would be nice to use something like 'WITH DATA'**
  - But that is NOT in the db2 knowledge center syntax diagram for DGTT. You must use 'WITH NO DATA'!
- It would be nice and logical if you could declare and insert in the same statement!
- **I have made RFE > DB24ZOS-I-940 to request IBM to consider this feature**
  - *If **YOU** agree with me then I would encourage you to go the ibmanalytics.ideas.aha.io website and vote for my RFE!*
  - *Link to IBM RFE website is in notes!*
- *Even worse! Or even better (for my argument). This concept exists in SQL Server and Oracle (and DB2 for LUW)… so really it is about time for DB2 for Z!*

https://ibmanalytics.ideas.aha.io/ideas/DB24ZOS-I-940

By the way, do not be confused, the IBM RFE website has moved from "developerworks" to "ibmanalytics.ideas.aha.io".
All old RFE have been moved (you want to review your old RFE <if you have any> in the new RFE website)

What I would like to see: declared variables! (1|2)

- The logical extension of DGTT is a something like DECLARED GLOBAL VARIABLE.
  - Then the variable could be used by some ad-hoc end-user who just wants to define a variable to hold some value and then reference that variable in several SQL in the same script (SPUFI or Data Studio).
    - Workaround is to make a DGTT with one row and one column. And then select that column from this tiny DGTT every time you want to know the value.
      - This is a kludgy solution. But it works. A variable would be more clean
    - Another workaround is to have the DBA "create" several generic variables in advance and then any ad-hoc or dynamic end-user could use them.

42

My usual example of wanting a dynamically declared variable is "ws-timestamp". I want to select from several tables and include data relative to a specific timestamp. I can set the timestamp value once up front in this declared variable and then each SQL can reference the same and consistent variable value.

What I would like to see: declared variables! (2|2)

- The declared variable idea seems logical to me. And it would seem useful for all types of ad-hoc (sophisticated) SQL script creators!
- **I have made RFE > DB24ZOS-I-941 to request IBM to consider this feature**
  - *If **YOU** agree with me then I would encourage you to go the ibmanalytics.ideas.aha.io website and vote for my RFE!*
  - *Link to IBM RFE website is in notes!*

https://ibmanalytics.ideas.aha.io/ideas/DB24ZOS-I-941

Do declared variables exist in other RDMS. I think yes.

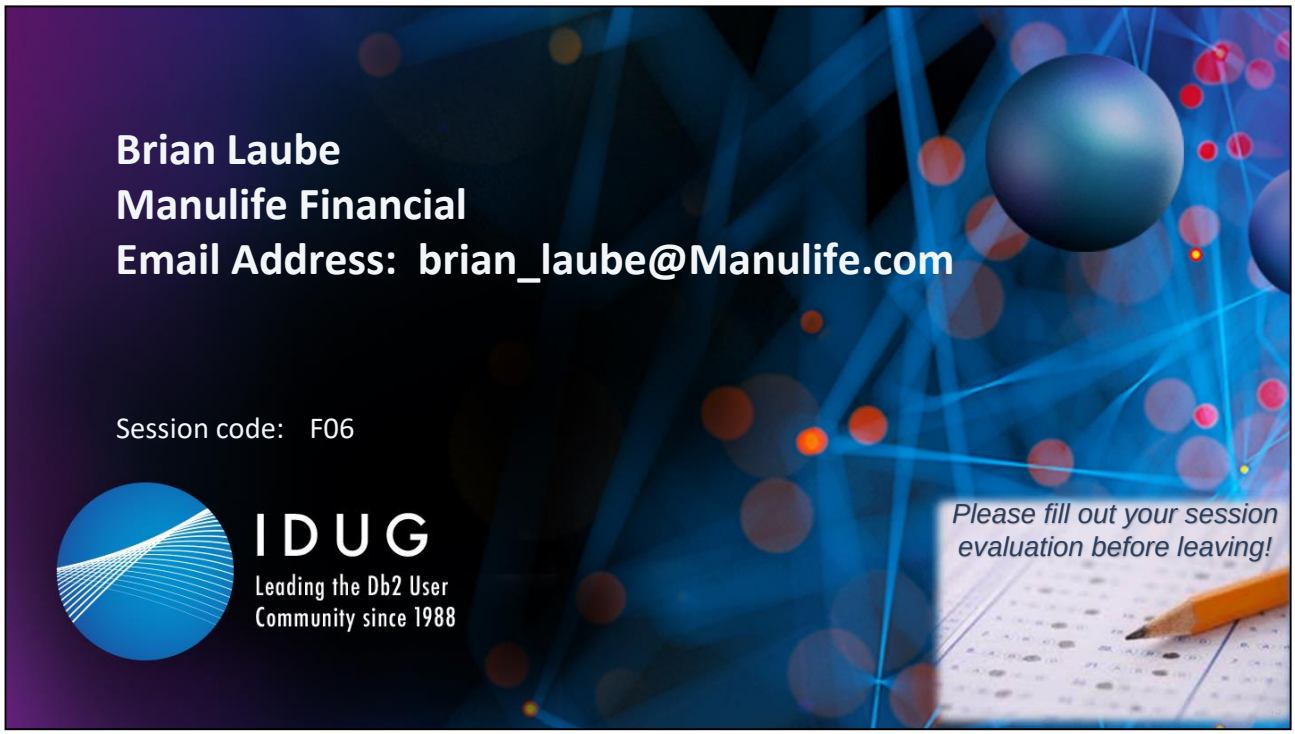
By the way, do not be confused, the IBM RFE website has moved from “developerworks” to “ibmanalytics.ideas.aha.io”.
All old RFE have been moved (you want to review your old RFE <if you have any> in the new RFE website)

The end – last chance for questions

- Any questions? Clarifications?
  - Ask me now. Ask me later. Or email me!
- See you on the DB2-listserv
  - *I lurk often, reply occasionally*

Of course, ask my questions now or anytime!

And I often read the db2-listserv. If you ask usage or scenario questions you often get a good response



The end

THANKS AGAIN