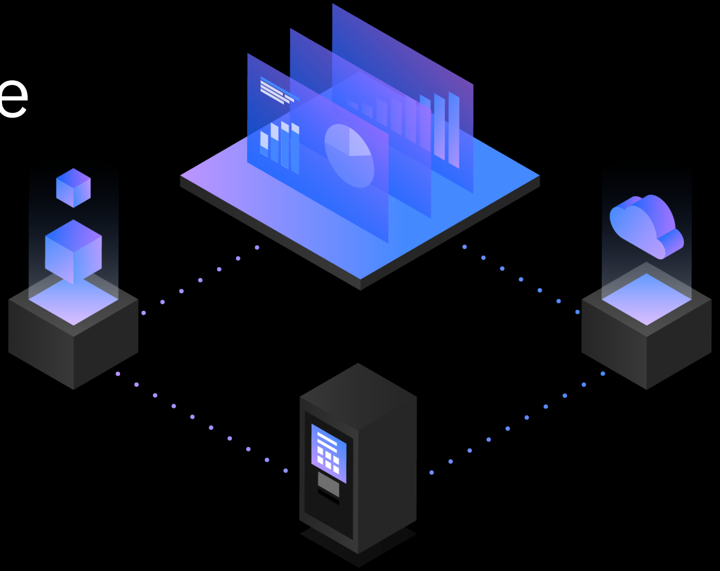
Bringing Db2 for z/OS-Based Applications Into the Modern Age



Central Canada Db2 Users Group & IDUG Data Tech Summit

September 19, 2023

Robert Catterall, IBM
Principal Db2 for z/OS Technical Specialist

Agenda

- Choosing the right interface – SQL or REST
- Leveraging Db2 for z/OS application-enabling functionality
- Putting SQL procedure language to work, and doing that in an agile way
- Synchronous versus asynchronous processing

Choosing the right interface – SQL or REST

A level-set on these two interfaces

- Until recently, only application interface to Db2 for z/OS was SQL interface
  - If an application were going to access Db2 data, it would issue SQL statements
  - Even if “table-touching” SQL statements packaged in a Db2 stored procedure, that stored procedure would be invoked by way of a SQL statement: CALL
- Db2 12 for z/OS introduced the REST interface to Db2 – when using that interface, a Db2-accessing program does not issue SQL statements
  - Application issues REST requests that invoke server-side static SQL statements
  - If stored procedures involved, difference is means of invoking stored procedures
    - SQL interface: SQL CALL – programmer knows server is relational DBMS
    - REST interface: REST request – nature of request-serving system completely abstracted from developer’s perspective

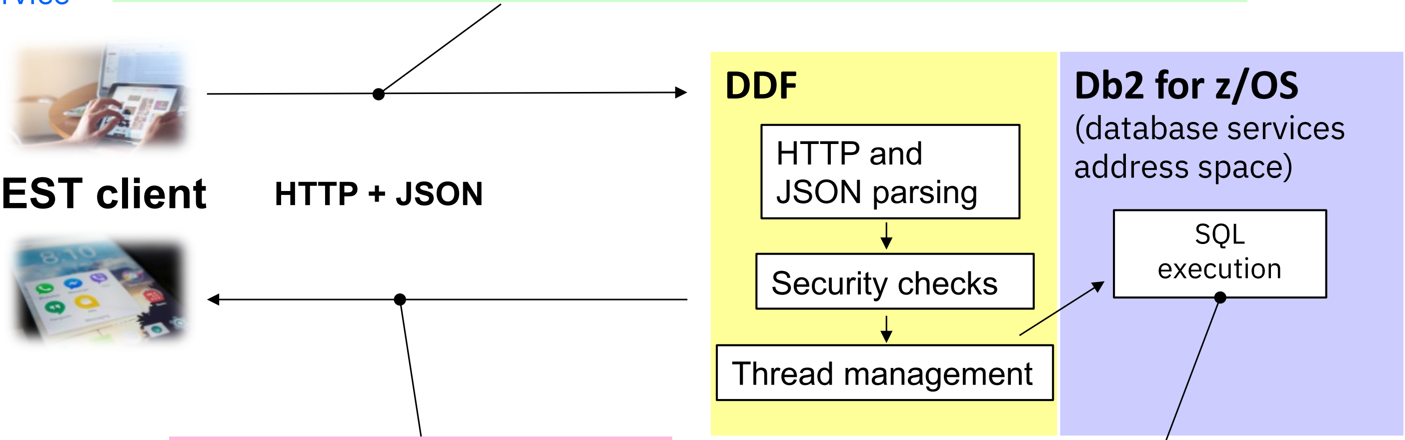
More on the REST interface to Db2 for z/OS

- Enables creation of REST service from single static SQL statement (SELECT, INSERT, UPDATE, DELETE, TRUNCATE or CALL [of stored procedure])
- It is an extension of the Db2 distributed data facility (DDF)
  - One implication of that: up to 60% of CPU cost of executing SQL statements invoked via REST interface offloaded to lower-cost zIIP engines on IBM Z server
- Secure: request must have ID and password (or certificate), and ID must have EXECUTE privilege on Db2 package associated with REST service
- Highly scalable, highly performant
- A Db2 REST service can be created with BIND SERVICE command, with Db2-provided DB2ServiceManager REST service, or with IBM z/OS Connect

A Db2 REST request round trip…

REST call (ACCOUNTS is collection name, getBalance is service name)

```
POST http://mybank.com:4711/services/ACCOUNTS/getBalance
Body: { “ID”: 123456789 }
```



REST client

HTTP + JSON

DDF

HTTP and JSON parsing

Security checks

Thread management

Db2 for z/OS (database services address space)

SQL execution

HTTP response in JSON format (JavaScript Object Notation)

```
Body:
{
  “FIRSTNAME”  :  “John”,
  “LASTNAME”   :  “Smith”,
  “BALANCE”    :  1982.42,
  “LIMIT”      :  3000.00
}
```

```
SELECT C.FIRSTNAME,
C.LASTNAME, A.BALANCE,
A.LIMIT
FROM ACCOUNTS A,
CUSTOMERS C
WHERE A.ID = ?
AND A.CUSTNO = C.CUSTNO
```

SQL statement (in form of a package)

SQL vs. REST applies mainly to client-server apps

- For “local to Db2” applications (e.g., JES batch jobs, CICS or IMS transactions), SQL interface will usually be the best choice
- For applications that will access Db2 for z/OS via DDF over TCP/IP connections, REST versus SQL is an important decision
  - Note: in client-server scenario, client using SQL interface is a DRDA requester
- In deciding between SQL and REST for new client-server application, there are several things to consider, including client-side programming language
  - IBM has drivers (e.g., JDBC, ODBC, ADO.NET) that support SQL access to Db2 for z/OS from programs written in variety of languages (e.g., Java, C#, Perl, Python)
  - What if you want to use a language for which there is not a Db2 SQL driver?
    - If program coded in that language can issue REST request, it can access Db2

Other SQL vs. REST considerations

- Even if the IBM Data Server Driver supports language that will be used on client side, is it feasible to install that driver on client-side app servers?
  - Use of REST interface to Db2 requires no client-side driver
- If desirable or necessary for client-side program to dynamically construct a SQL statement and send it to Db2, SQL interface would make sense
  - Example: build a SELECT based on combination of screen fields filled by user
- What about control over scope of a Db2 unit of work (UOW)?
  - With SQL interface, client application determines scope of UOW by issuing a commit – if that is important for application, SQL interface could be best choice
  - With REST interface, each REST request is a separate UOW
    - For multiple SQL statements with one REST request, use stored procedure

Db2’s REST interface and z/OS Connect

- A client application can directly access Db2’s REST interface; alternatively, Db2’s REST interface can be accessed using IBM z/OS Connect
  - In that case, Db2 is a REST provider to z/OS Connect
- What z/OS Connect provides:
  - GUI tooling makes it easier to create REST services from Db2 SQL statements
  - Automatically-generated Swagger descriptions of Db2 REST services (industry-standard service description specification – helpful for service discovery)
  - Flexibility in coding Db2-targeted REST request: use any HTTP verb (e.g., GET, PUT) – when Db2’s REST interface accessed directly, POST form required
  - Flexibility in formatting JSON document that is output of Db2 REST service
- z/OS Connect also enables outbound REST requests (e.g., from COBOL)

Leveraging Db2 for z/OS application-enabling functionality

A twofold objective

1. If a Db2 capability can provide functionality that an application needs, that's functionality developers don't have to provide with program code
   - Accelerates application development, reduces application maintenance
2. If particular capability can be provided either via application code or Db2 feature, highly likely that Db2 feature will provide the best performance
   - Pushing functionality into the database layer boosts efficiency

The next few slides highlight some of the more important application-enabling features of Db2 for z/OS – consider whether they would be helpful for your application requirements

Temporal data support

- Comes in 2 “flavors” – one is system-time temporal (aka row versioning)
  - How that works: suppose that table T1 has been enabled for row versioning
  - If row in T1 is updated or deleted, Db2 inserts “before” image of row (i.e., row as it was prior to update or delete) in “history” table associated with T1
    - Db2 also updates timestamp column values in history table row, showing when row became “current” (i.e., when it was inserted in T1, or when it was changed via UPDATE) and when it stopped being current (when deleted or updated)
  - What this means:
    - Using temporal query syntax (easy to code), an application can see what the current version of a row looked like *at a prior point in time*, or how a row changed during a specified period of time (and who changed the row)

Second “flavor” of temporal data support

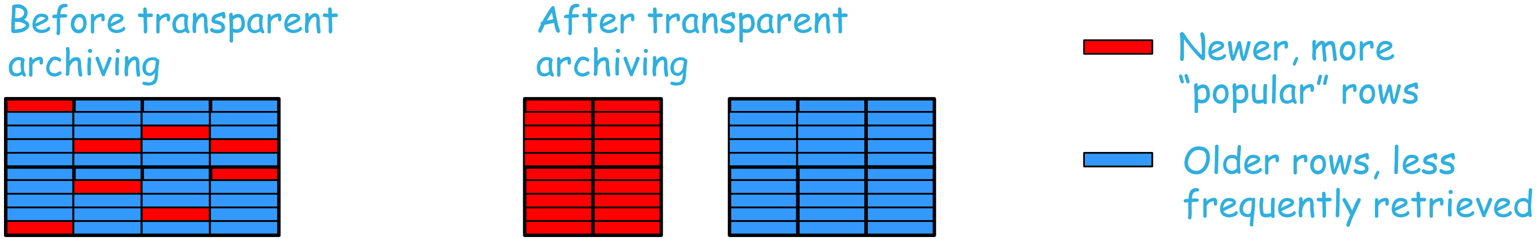
- Business-time temporal
- Allows future data changes (such as product price changes) to be inserted into a table, along with an indication of when a change will go into effect and how long it will be in effect (if not indefinitely)
  - Adding these future changes to a table does not impact applications that, by default, are accessing rows that are currently in effect
- Advantages of future data changes being added to a table beforehand:
  - Ensures that future changes will go into effect when they are supposed to
  - Allows (for example) business analysts to submit temporal queries that will show what revenue and profits would be with prices that will be in effect *at a future time*

Storing XML documents in Db2 table columns

- A column of a Db2 for z/OS table can have the XML data type
- The XML data type gives Db2 awareness of XML documents – structure of documents is understood, and supporting functionality is available:
  - Schema validation
  - Ability to retrieve and modify XML data using XQuery expressions
  - Ability to retrieve data in XML documents in tabular form
  - Ability to define indexes on elements of XML documents to speed data retrieval
  - Ability to transform an XML document with an XSL style sheet
  - And more…

Db2 transparent archiving

- Suppose table T1 holds 20 years of transaction data, and 95% of all queries target rows inserted within the past 3 months (the “popular” rows)
- If T1’s clustering key not continuously-ascending, and if inserts outnumber deletes, popular rows are separated by ever more “old and cold” rows
  - Result: performance degradation for access to popular rows
- With Db2 transparent archiving, T1 holds only most recent 3 months of data (for example) – other rows are in archive table associated with T1
  - Result: better performance for retrieval of popular rows
  - Query coding unaffected – Db2 makes T1 and archive table appear to be 1 table


Before transparent archiving
After transparent archiving
🟥 Newer, more “popular” rows
🟦 Older rows, less frequently retrieved

Result set pagination

- Makes it easier to write a program that returns parts of a query result set in “pages” as user scrolls through
- Enabled via OFFSET clause for SELECT, introduced with Db2 12 – example:
  - First page of 20 rows: `OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY`
  - Second page of 20 rows: `OFFSET 20 ROWS FETCH FIRST 20 ROWS ONLY`
- And, both OFFSET and FETCH FIRST values can be parameter markers or host variables – you could decide that after first 3 pages of 20 rows each have been returned, subsequent pages will have 30 rows apiece
  - Example: `OFFSET ? ROWS FETCH FIRST ? ROWS ONLY`

“Piece-wise” DELETE

- Coding SQL for removing a large number of rows from a table hard

```
DELETE FROM T1 WHERE C1 > 7
```

  - Problem: if T1 has 500 million rows and 50 million of them have a value greater than 7 in column C1, execution of statement above will acquire a ton of locks

- Db2 “piece-wise” DELETE functionality makes it easy to code SQL that will remove a large number of rows from a table in bite-sized units of work
  - Based on including FETCH FIRST clause in DELETE statement - for example:

```
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 500 ROWS ONLY;
```

Delete first chunk of 500 rows

```
COMMIT;
DELETE FROM T1 WHERE C1 > 7 FETCH FIRST 500 ROWS ONLY;
```

Delete second chunk of 500 rows

```
COMMIT;
```

Newer built-in Db2 functions

- PERCENTILE_CONT (column values treated as points in continuous distribution) and PERCENTILE_DISC (column values treated as discrete data values) make it easy to answer questions like this one:
  - “What is the 90th percentile for salaries of people in department A02?”
- HASH_MD5 lets you get an MD5 hash of a value prior to (for example) inserting it into a table (related functions: HASH_CRC32, HASH_SHA1, HASH_SHA256)
- LISTAGG makes it easy to have a query result set column that is a comma-separated list of values (e.g., last names of employees in each department)
  - And, separator need not be a comma – can be any character string constant

Db2 global variables

- Db2 global variable: created by a DBA, versus being declared in a program
  - After creating a global variable, DBA permits IDs (e.g., of applications) to use it
- When application program references global variable, it gets its own instance of the global variable (instance exists for a Db2 session)
- A global variable makes it easy to get a value from a Db2 table and pass it to a subsequent SQL statement in the Db2 session
  - When second SQL statement references global variable (e.g., in query predicate), it is effectively referencing value placed in global variable by first SQL statement
- Global variables also make it easy for program to receive a value from Db2 advanced trigger (trigger that includes a SQL procedure language routine)
  - Trigger, when fired, places value in global variable, and when control returns to trigger-firing program it can see and use the value in the global variable

Db2 arrays

- A Db2 array is a Db2 user-defined data type created by a DBA – once created, it can be used in SQL statements
  - Db2 arrays can be useful for passing a set of values to a called stored procedure
    - Stored procedure must be native SQL procedure (written in SQL PL), caller must be either Java or .NET DRDA requester or another SQL PL routine
  - Also: a Db2 global variable can have an array data type
    - Pass set of values from one SQL statement to another within Db2 session
- Two types of Db2 array:
  - Ordinary (logically, like stack of individual values)
  - Associative (stack of pairs of values – each element has associated “index” value)
- Db2 provides built-in functions to populate and otherwise work with arrays

Application-specific lock timeout limit

- A recent enhancement, provided by Db2 13 for z/OS
- New Db2 special register, CURRENT LOCK TIMEOUT, can be set by program (like global variable, special register is relevant to a Db2 session)
- Potential use cases (assume lock timeout value for system is 30 seconds):
  - For a mobile app that accesses Db2, development team might want 5-second lock timeout limit – if reached, send "please try again" message to user
    - Preferable to having user look at spinning colored wheel for up to 30 seconds
  - If a certain mission-critical Db2 batch application runs for three hours at month-end, development team might want a 10-minute lock timeout limit
    - Might be preferable to situation in which job has been running for 2 hours and then gets a Db2 lock timeout error because it had to wait 30 seconds for a lock

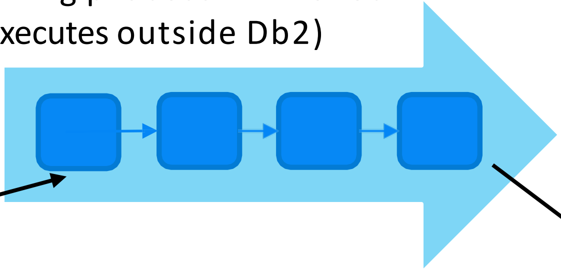
AI for data analysis – Db2 SQL Data Insights (SQL DI)

- New feature of Db2 13 for z/OS – advanced machine learning technology incorporated with the Db2 database "engine"
- No data scientist required to activate and utilize the feature
- Three new associated built-in Db2 functions:
  - AI_SIMILARITY
  - AI_SEMANTIC_CLUSTER
  - AI_ANALOGY
- SQL DI provides ability to execute "fuzzy" queries
  - Example: "Here is the account ID of someone who engaged in fraudulent activity – show me the 10 account IDs most like this one"

The key: you don't have to tell Db2 what you mean by "like"

SQL DI – the big picture

Model training process – invoked via GUI, executes outside Db2)



Model is a Db2 table containing encoded vectors for each distinct entity in the source table

SQL DI built-in functions retrieve the vectors to calculate their results

USER.DATA_TABLE

CUSTOMER_ID	GENDER	BILLING	…
3668-QPYBK	F	auto	…
…	…	…	…

Data Engineer

DBA

DSNAIDB.<generated vector table name>

Column	Value	vector
CUSTOMER_ID	3668-QPYBK	<1280 byte vector>
…	…	…
CUSTOMER_ID	8923-VFGHT	<1280 byte vector>
GENDER	F	<1280 byte vector>
…	…	…

SQL:

```
SELECT CustomerID,
AI_SIMILARITY(CUSTOMER_ID, '3668-QPYBK')
FROM USER.DATA_TABLE
WHERE ...
```

Business Analyst

App Developer

Putting SQL procedure language to work, and doing that in an agile way

Some background on SQL procedure language

- SQL PL effectively introduced with Db2 9 for z/OS
- It's a way to write Db2 routines (stored procedures, user-defined functions and advanced triggers) using only SQL statements
  - That is do-able thanks to a set of Db2 SQL statements known as control statements – a reference to logic flow control
  - SQL control statements include GOTO, ITERATE, LOOP, WHILE
    - Additionally, variables can be declared in a SQL PL routine
- Terminology for SQL PL routines in a Db2 for z/OS system:
  - Stored procedure written in SQL PL is called a native SQL procedure
  - User-defined function written in SQL PL is called a compiled SQL scalar function
  - Db2 trigger that includes a SQL PL routine is called an advanced trigger

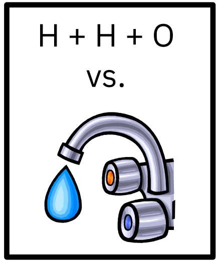
Key characteristics of SQL PL routines

- A SQL PL routine's one and only executable is its Db2 package; therefore:
  - No associated z/OS load or object module
  - A SQL PL routine runs in the Db2 database services address space (where all SQL executes) – not in an external-to-Db2 address space
  - A SQL PL routine never has its own task – always runs under task of invoker
  - When routine invoked by network-connected application (DRDA requester or REST client), task is a preemptable SRB in Db2 DDF address space
    - That makes SQL execution up to 60% zIIP-eligible (reduces cost of computing)
  - Also, no need to switch Db2 thread from caller's task to task of Db2 routine
    - Helps performance when SQL routine invoked many times for a process (e.g., used in inner SELECT of correlated subquery)

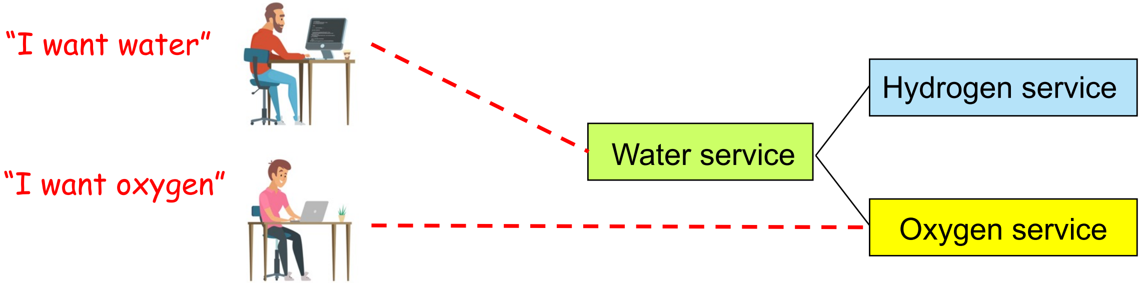
Functional advantages of native SQL procedures

- A native SQL procedure can be coded as an autonomous procedure
  - What that means: suppose a program calls an autonomous procedure and the procedure does some data-change work (e.g., inserts a row into a table), and after control returned to calling program that program fails
    - In that case, Db2 rolls back data-change work done by program in the unit of work, but data-change work done by autonomous procedure is not rolled back
    - True because autonomous procedure has its own unit of work
    - This can make autonomous procedures very useful for things like transaction “audit trail” functionality (insert done by autonomous procedure records fact that transaction started, and that record is preserved even if transaction fails)
- A native SQL procedure can accept a Db2 array as input

Consider usefulness of “tiered” data services



- Services that are too fine-grained can put burden on developers
  - A developer complained that an application “makes me ask for two atoms of hydrogen and one atom of oxygen – what I want is water”
- Services that are too coarse-grained limit flexibility in combining services
- An arrangement that lets you have it both ways: coarse-grained services that are comprised of finer-grained services, with latter being directly invoke-able by programs that require only narrow-scope services

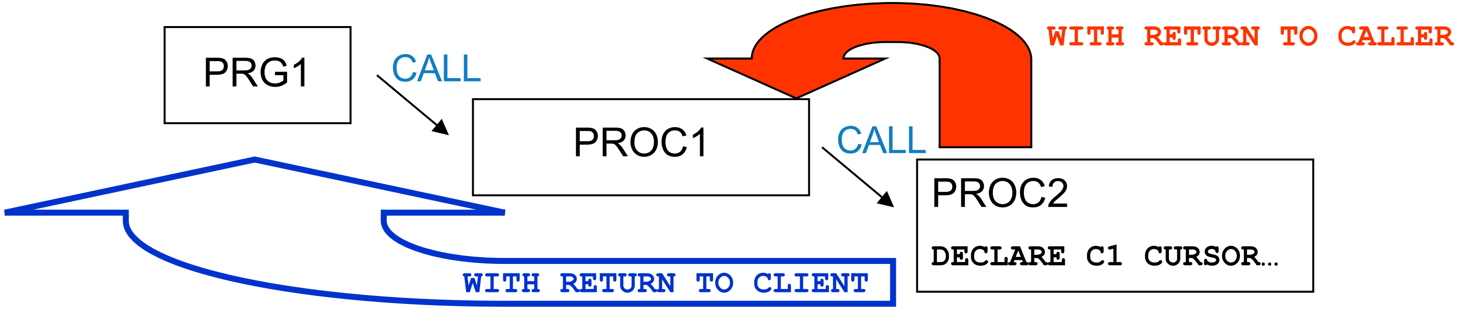


Tiered data services and Db2 stored procedures

- Db2 data services could be tiered by way of nested stored procedures
  - A nested stored procedure is one that is called by another stored procedure
  - Db2 for z/OS supports up to 64-deep nesting of stored procedures
- Native SQL procedures can boost efficiency in nested procedure situation
  - External stored procedures (written in languages other than SQL PL – e.g., COBOL) always run under their own tasks in z/OS system, and run in address spaces outside of Db2 – a lot of "moving parts" when procedures are nested
  - As previously noted, a native SQL procedure runs in the Db2 database services address space and never has its own task – always runs under task of its invoker
    - Result: more streamlined execution environment when procedures are nested

Nested stored procedures and result sets

- Suppose client program PRG1 calls Db2 stored procedure PROC1, and PROC1 calls stored procedure PROC2, and PROC2 generates a result set that is needed by PRG1 – how can PRG1 retrieve those rows?
  - Clunkier, less-efficient way: have PROC2 declare a Db2 temporary table and insert result set rows into that table – PRG1 will fetch rows from the temp table
  - Better way: cursor for PROC2 result set declared WITH RETURN TO CLIENT
    - That makes result set rows generated by PROC2 directly fetch-able from PRG1



Agility advantages of SQL procedure language

- SQL PL routines can be deployed via SQL statements (e.g., CREATE PROCEDURE), and it's much more likely that an application deployment tool can issue SQL statements versus Db2 commands
  - The fact that Db2 for z/OS SQL PL routines have no associated load or object modules (a SQL PL routine's package is its only executable) eliminates an "other-ness" factor that could complicate DevOps single-streaming
- If you really want to maximize deployment agility, use CREATE OR REPLACE syntax for Db2 stored procedures (introduced with function level 507 of Db2 12)
  - Especially useful for native SQL procedures – can specify version ID in that case

CREATE OR REPLACE PROCEDURE – examples

Create procedure MYPROC1

```
CREATE PROCEDURE MYPROC1
   ( IN  P1 CHAR(5),
     OUT P2 DECIMAL(15,2) )
  BEGIN
     SELECT AVG(SALARY) INTO P2
       FROM DSN8C10.EMP
      WHERE WORKDEPT = P1;
  END
```

Replace MYPROC1 with new definition

```
CREATE OR REPLACE PROCEDURE MYPROC1
   ( IN  P1 CHAR(5),
     OUT P2 DECIMAL (15,2) )
  BEGIN
     SELECT AVG(SALARY + 1000) INTO
P2
       FROM DSN8C10.EMP
      WHERE WORKDEPT = P1;
  END
```

Change body of procedure

Add version V2 of MYPROC1

```
CREATE OR REPLACE PROCEDURE MYPROC1
   ( IN  P1 CHAR(5),
     OUT P2 DECIMAL (15,2) )
  VERSION V2
  BEGIN
     SELECT AVG(SALARY + 5000) INTO
P2
       FROM DSN8C10.EMP
      WHERE WORKDEPT = P1;
  END
```

Replace version V2 of MYPROC1

```
CREATE OR REPLACE PROCEDURE MYPROC1
   ( IN  P1 CHAR(5),
     OUT P2 DECIMAL (15,2) )
  VERSION V2
  BEGIN
     SELECT AVG(SALARY + 9000) INTO
P2
       FROM DSN8C10.EMP
      WHERE WORKDEPT = P1;
  END
```

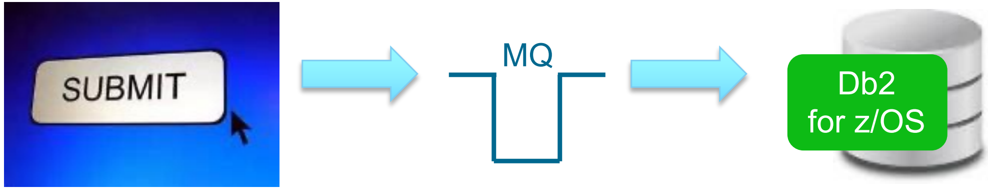
One more thing about SQL PL routines and agile development

- Source code for native SQL procedure is a CREATE PROCEDURE statement
- How should you manage the source code for native SQL procedures?
- My answer: use the same source code management tool (SCM) that you use for programs written in other languages
  - One example: GitLab
- Don't be thrown off by word CREATE in source for a native SQL procedure
  - SCM doesn't care about language in which programs are written – the SCM manages versions of source code, written in whatever language
- My point: treat SQL PL CREATE PROCEDURE statements like source code, because that's what they are

Synchronous versus asynchronous processing

Building “flex” into Db2-accessing applications

- It can be helpful to put a queue (e.g., an IBM MQ queue) between a client application and a Db2 system – Db2 processing is then asynchronous
- Scenario: application user inputs data that will lead to a Db2 data change
  - If data change will be synchronous with respect to user clicking “Submit,” and a to-be-updated table is unavailable for some reason, transaction could time out
  - Suppose instead that client program puts data provided by user on MQ queue, and server-side process takes data off queue and performs data change actions
    - In that case, if target table is temporarily unavailable, end user not impacted – data remains on queue, and data change process can proceed when table is again available



How do MQ messages become Db2 updates?

- Typically, by way of an MQ listener
  - There is a CICS MQ listener (message arriving on queue drives execution of a CICS transaction – data in the MQ message is input to the transaction)
  - Db2 also provides an MQ listener – with it you can associate an MQ queue with a Db2 stored procedure
    - When message arrives on queue, associated stored procedure is automatically invoked – message is passed as input to the stored procedure
    - You can set up several queues for different message types, and each queue can be associated with a different stored procedure

MQ and batch transactionalization

- If clients now send files of records that drive Db2 batch update jobs, would it be beneficial for clients to be able to send – and you to process – a record at a time?
  - That can be done with MQ – queues can be securely exposed as web services
  - System then functions more like a refinery – continuous flow – versus “job shop”
- One benefit: reduced latency
  - Input record can be sent and processed on your system as soon as record is available on client side
  - This as opposed to records waiting to be batched up on client side and then sent in as files a few times per day (maybe only once per day)

Robert Catterall

rfcatter@us.ibm.com