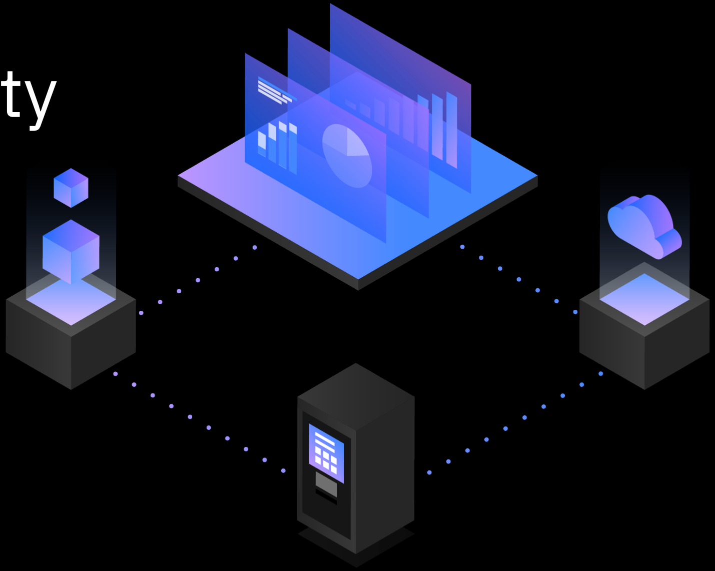
Key features of Db2 13 for z/OS delivered since general availability

Central Canada Db2 Users Group

May 13, 2024

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



Agenda

- Auto-bind phase-in
- Utility execution history – object-related information
- More-granular specification of security requirements for DDF-using applications
- The fourth SQL Data Insights built-in function: AI_COMMONALITY
- Temporal functionality for security-related Db2 catalog tables

Auto-bind phase-in

The problem that needed a solution

- Historically, altering database objects (tables, table spaces, indexes) has often been disruptive for Db2-accessing applications
- Why? Because these ALTER actions often require regeneration of dependent packages
  - For a program that issues static SQL statements, you can think of its package as the compiled, executable form of those SQL statements
  - If package has statements that reference a table, the package is dependent on that table (and associated table space and – maybe – associated indexes)
  - When an ALTER invalidates a package, that's Db2 saying, "I need to regenerate the code for this package, to accommodate the ALTER action"
  - When a package is invalidated, the next request to execute the package triggers an autobind, which regenerates the package

ALTERs can be disruptive because autobind can be disruptive

- How is it that autobind can be disruptive? Two ways (referring to the pre-Db2 13 situation):

  1. When an application requests execution of an invalidated package, it must wait for the resulting autobind to complete (and so must other applications that request execution of the package, before the autobind has completed)
  2. If autobind fails for some reason, package is marked "inoperative" by Db2

     - *In that case, the package cannot be executed until it is <u>explicitly</u> rebound*

Db2 13 post-GA feature: autobind phase-in

- Functionality delivered with Db2 13 function level 504 – groundwork laid with function levels 500 and 502
  - With FL500 activated, you can execute CATMAINT to take the catalog level to V13R1M501
    - At that level, catalog has new table, SYSPACKSTMTDEP
  - When FL502 is activated, packages can bound or rebound with a new option: DEPLEVEL
    - When package bound or rebound with DEPLEVEL(STATEMENT), Db2 will record statement-level dependency information in SYSPACKSTMTDEP
    - Db2 *does something with that information* beginning with FL504

Function level 504: what is the same

- Assume a package is bound with DEPLEVEL(STATEMENT)
  - If the package has one or more statements that are dependent on a database object, and that object is altered in a way that requires package regeneration, the package will be marked "invalid" by Db2
  - If the package has been invalidated, the next request to execute the package will trigger an autobind

Function level 504: what is different

- Again, assume a DEPLEVEL(STATEMENT) package – that package has been invalidated as a result of an ALTER
  - One difference: the autobind is a background process – *no program will wait for it to complete*
  - If programs requesting execution of the invalidated package do not have to wait on the autobind, that must mean that the invalidated package is still executable, even before the package code has been regenerated by Db2

    How is that possible?

How an invalidated package remains executable

- Suppose the invalidated DEPLEVEL(STATEMENT) package has 20 SQL statements, and 2 of those were invalidated by the ALTER action
    - When the invalidated package is requested for execution before the autobind completes (in the background), the 18 statements that were not invalidated by the ALTER *continue to execute as before*
    - The 2 invalidated statements are executable because Db2 incrementally binds them when they are executed
        - That means some additional CPU cost (similar to dynamic SQL preparation), but it preserves statement execute-ability
- Once autobind completes, requests to execute package will get the newly generated copy – that new instance of the package is phased into use

This is why the feature is referred to as “autobind phase-in”

But wait – there’s more!

- If the autobind of a DEPLEVEL(STATEMENT) package fails, the package will NOT be marked as “inoperative”
  - Instead, it will continue to be execute-able, as described on the previous slide (non-invalidated statements execute as before, invalidated statements are incrementally bound and executed)
  - A subsequent explicit rebind of the package will accomplish what autobind would have accomplished had it not failed – the package will be regenerated, and it’s status will return to “valid”
    - After failed autobind and before explicit rebind, a DEPLEVEL(STATEMENT) package’s status will be rebind-advisory (OPERATIVE = ‘R’ in SYSPACKAGE)

Utility execution history – object-related information

Some background

- Initial phase of utility execution history functionality was there at general availability of Db2 13 – usable once function level 501 has been activated
  - SYSUTILITIES table added to catalog when catalog level goes to V13R1M501
  - When function level V13R1M501 has been activated, you can set the value of the new ZPARM parameter UTILITY_HISTORY to UTILITY – this will cause Db2 to write a row to SYSUTILITIES every time a utility executes
  - That row provides a lot of useful information, such as:
    - Name of utility (REORG, LOAD, COPY, etc.), user ID of utility invoker, starting logpoint, start and end timestamps, general-purpose CPU consumption, zIIP CPU consumption, return code, and more
  - That's really nice, *but something's missing…*

The missing piece: object information

- Suppose you want to know not only what utilities have executed, but also what objects (table spaces, indexes) those utilities acted on?
- With the initial phase of utility execution history functionality, you could sometimes get that information for a given utility
  - I say, “sometimes,” because initially object information was obtained by joining SYSUTILITIES with SYSCOPY, using EVENTID column present in both tables
  - The problem there: *not every utility drives SYSCOPY insert activity* – examples there include RUNSTATS and UNLOAD
  - What that meant: if you wanted to know (for example) who executed UNLOAD for a particular table over the past week, Db2 utility execution history information was not going to be much help

Missing no more, thanks to ‘04 (function level 504, that is)

- When catalog goes to V13R1M504 level, table SYSOBJEVENTS is added
- When ZPARM parameter UTILITY_HISTORY is set to OBJECT (do-able when FL504 has been activated), this is what happens:
  - In addition to inserting utility execution information in SYSUTILITIES, Db2 will insert a row into SYSOBJEVENTS for each object (table space or index, or partition) processed by a utility
  - Now, the question, “Who executed UNLOAD for table space XYZ over the past week?” is easy to answer:
    - Just issue query that joins SYSUTILIIES and SYSOBJEVENTS (on EVENTID column), and that includes predicates for SYSUTILITIES.NAME = ‘UNLOAD’ and SYSOBJEVENTS.SPACENAME = ‘XYZ’

More-granular specification of security requirements for DDF-using applications

The problem(s) that needed a solution

- When it came to security for DDF-using applications (e.g., whether or not AT/TLS encryption, aka SSL encryption), the ZPARM parameter TCPALVER specified requirements for the entire DDF workload
  - That, in turn, presented some difficulties
    - You might have different security requirements for different types of DDF client applications
    - Implementing something such as SSL encryption might involve flipping a “big switch” – it could be challenging to “phase in” such a security change, and that increased the risk of problems

Db2 profile tables to the rescue

- APAR PH48764 (March 2023) provided a new capability for the Db2 profile tables: more-granular management of DDF security requirements
- Works by way of a new specification for the DSN_PROFILE_ATTRIBUTES table: MONITOR *product-type* CONNECTIONS FOR SECURITY
  - "Product type" is a category of DDF-using applications – possible values are:
    - REST
    - JDBC
    - CLI (basically refers to the ODBC driver)
    - DB2CONNECT (refers to a Db2 Connect gateway server)
    - DSN (refers to a client that is another Db2 for z/OS system)
    - * (means, "otherwise", i.e., "for categories not explicitly specified")

Before getting to the attribute, a word about the profile

- The profile entry used with MONITOR *product-type* CONNECTIONS FOR SECURITY needs to look like this (except that LOCATION can also be * or 0.0.0.0, and PROFILEID can be whatever):

DSN_PROFILE_TABLE

PROFILEID	LOCATION	ROLE	AUTHID	PRDID	COLLID	PKGNAME
1	::0	null	null	null	null	null

- This makes sense, when you think about it
  - “Location” is generic, and all other identifiers are null, because this profile will be related to categories of DDF-using applications that are over and above auth ID, product ID, collection ID, etc. (e.g., “for ALL clients connecting via JDBC driver”)

And here is what associated attribute entries can look like

DSN_PROFILE_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3
1	MONITOR REST CONNECTIONS FOR SECURITY	EXCEPTION	5	null
1	MONITOR JDBC CONNECTIONS FOR SECURITY	WARNING	1	1

- Some things to note:
  - MONITOR *product-type* CONNECTIONS FOR SECURITY attributes can have warning (good for discovery) or exception (for enforcement) behavior
  - ATTRIBUTE2 indicates security requirement for DDF application category (see next slide)
  - ATTRIBUTE3 indicates whether AT/TLS (i.e., SSL) encryption is required
    - Null: encryption not required
    - 1: encryption required

Ignored when certificate-based client authentication used – in that case, AT/TLS encryption always required

ATTRIBUTE2 – security requirement for DDF app category

- Possible values:
  - Null – Honor TCPALVER value ("For this category, act as though MONITOR *product-type* CONNECTIONS FOR SECURITY functionality does not exist")
  - 1 – Basic authentication (ID + password or passphrase)
  - 2 – Basic authentication, with additional requirement: multi-factor authentication
  - 4 – Client certificate required for authentication (requires AT/TLS encryption)
  - 5 – Combination of 1 and 4: use basic authentication or client certificate
  - 6 – Combination of 2 and 4: use basic authentication with MFA, or use client certificate-based authentication

An example

DSN_PROFILE_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3
1	MONITOR REST CONNECTIONS FOR SECURITY	EXCEPTION	5	null
1	MONITOR JDBC CONNECTIONS FOR SECURITY	WARNING	1	1

- What this means:
  - Applications accessing this Db2 subsystem via the REST interface can authenticate using a client certificate (in which case AT/TLS encryption is required) or with password or passphrase (and in that case, AT/TLS encryption is NOT required)
  - Applications accessing this Db2 subsystem via JDBC driver can authenticate with a password or passphrase, and in that case AT/TLS is required
    - If a JDBC-using client application authenticates in some other way, issue a warning message
  - Everyone else: whatever TCPALVER indicates applies

Some ways in which this functionality could be used

- The obvious: you might in fact have different security requirements for different categories of DDF-using applications
- Other possibilities:
  - With warning behavior in effect, you could discover whether any REST clients are NOT authenticating with a client certificate
  - What if you wanted to “phase in” use of AT/TLS encryption?
    - You could let ODBC-using clients be the “canaries in the coal mine” – get AT/TLS encryption working for them, then move on to other categories
    - And, for the ODBC-using clients, if problems were encountered in the transition, you could disable the profile, fix the issues, then re-enable profile
    - Bottom line: nice to have options other than “the big switch”

The fourth SQL Data Insights built-in function: AI_COMMONALITY

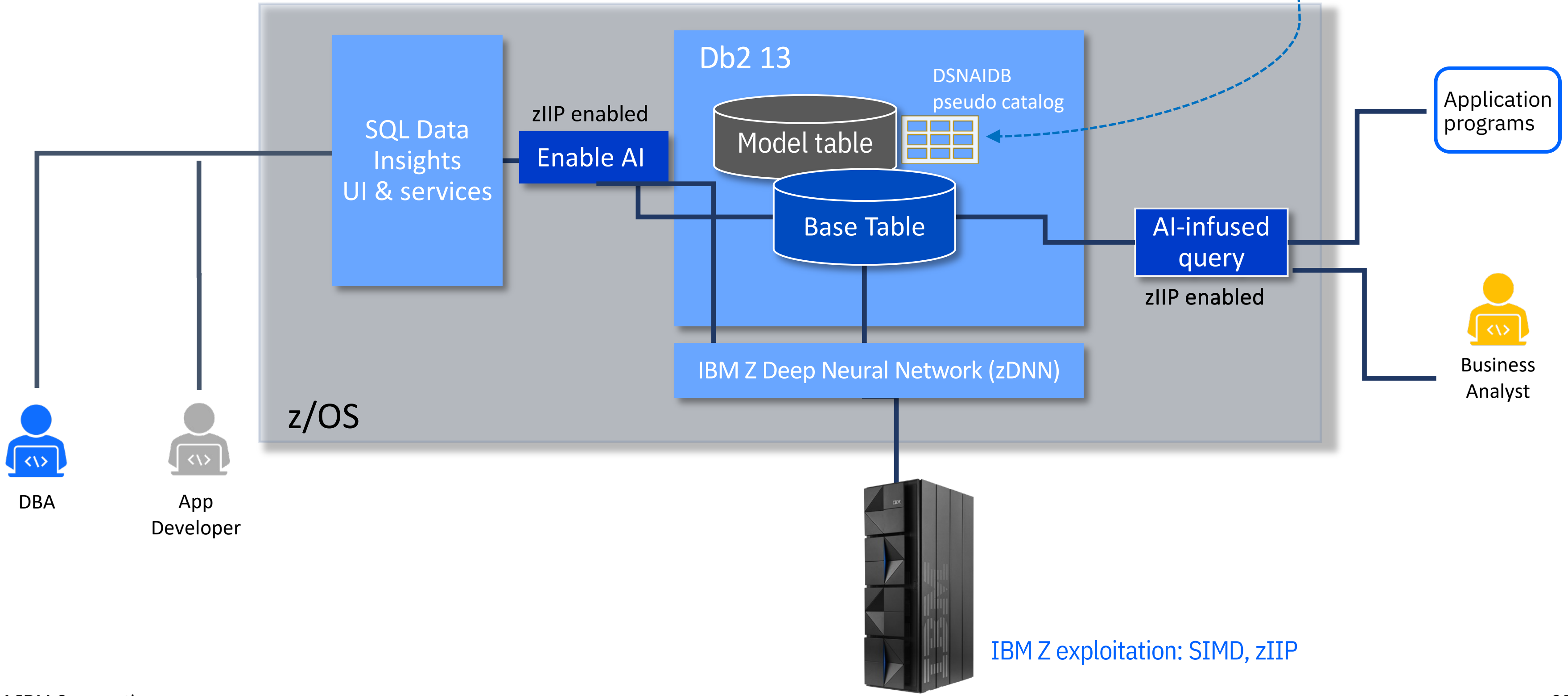
Background: SQL Data Insights

- Initial phase of this functionality present for Db2 13 at GA time
- What it does: allows Db2 to answer "most similar to this" or "least similar to that" questions, *without your having to tell Db2 what you mean by "similar to" or "not similar to"*
- How it works: via new Db2 built-in AI functions that use vectors of numerical comparative values (stored in a "model table" associated with a base table) to discern patterns of similarity (or dissimilarity) in table's data
- Easy for a DBA to enable, easy for a user (or a developer) to utilize

The big picture

Vector information

```
8879-zZna
-0.141558 -0.346767 -0.453296 0.052447 0.476916
-0.338483 0.000035 0.517277 0.191573 0.076891
-0.149729 1.036879 0.127160 -0.329846 -0.157252
-0.288485 0.243588 0.038326 -0.338862 0.173571
0.231060 0.149021 -0.328546 -0.058121 0.025713 …
```



The initial set of Db2 built-in AI functions

Function	Description
AI_SIMILARITY	Returns the entities that are most similar to (or dissimilar to) a particular entity
AI_SEMANTIC_CLUSTER	Returns the entities that are most similar to (or dissimilar to) a given set of up to three entities
AI_ANALOGY	Consider the relationship between value X in COL1 and value Y in COL2, and return the most analogous COL2 values if the COL1 value is Z

With Db2 13 function level 504: a 4th SQLDI function

- AI_COMMONALITY
- What it does: the function identifies "outliers" – the values of a column that are farthest from the column's "centroid" value
  - A similarity score is generated for each value of the referenced column
    - -1 is minimum value of similarity score – the closer to -1 the similarity score is for a given value of the column, the further that value is from the centroid
- Table T1 contains customer data for a bank – this query would return the 10 most-outlying values for account balance:

```
SELECT AI_COMMONALITY(ACCT_BALANCE) AS SCORE, ACCT_BALANCE
FROM T1
ORDER BY SCORE ASC
FETCH FIRST 10 ROWS ONLY;
```

One use of AI_COMMONALITY: data analysis

- An analyst might want to know the outlier values for a given column, for two reasons:
  - The analyst may actually want to analyze the outlier values, or the rows that contain those outlier values
  - The analyst may want to eliminate from consideration the rows with outlier values for a given column, because the outlier values would skew analytical results in an undesirable way
    - But you can’t eliminate the outliers *if you don’t know what they are* – AI_COMMONALITY tells you what they are

Temporal functionality for security-related Db2 catalog tables

The problem that needed a solution

- The security-related tables in the Db2 catalog only showed the current-state situation
- Researching “the past” (what had been true) was a time-consuming, cumbersome task
  - Often required analysis of Db2 log data to identify and document INSERT/UPDATE/DELETE records to respond to an auditor’s request

Auditor or
security administrator

“Who had access to the ACCOUNTS table since the last audit?”

Db2 administrator
(prior to Db2 13)

“Uhh…”

Sakes alive, it’s 505!

- When Db2 13 FL505 is activated, system-time temporal functionality (aka row versioning) can be implemented for any of these catalog tables:
    - SYSAUDITPOLICIES
    - SYSCONTROLS
    - SYSCONTEXT
    - SYSCONTEXTAUTHIDS
    - SYSCTXTTRUSTATTRS
    - SYSROLES
    - SYSCOLAUTH
    - SYSDBAUTH
    - SYSPACKAUTH
    - SYSPLANAUTH
    - SYSRESAUTH
    - SYSROUTINEAUTH
    - SYSSCHEMAAUTH
    - SYSSEQUENCEAUTH
    - SYSTABAUTH
    - SYSUSERAUTH
    - SYSVARIABLEAUTH

When the catalog level goes to V13R1M505...

- Row versioning columns are added to security tables that did not already have them (SYSROLES,SYSVARIABLEAUTH)
- A GEN_SESSION_USER column is added to all security tables to track the auth ID that granted or revoked a privilege or an authority
- History tables are created for all security-related catalog tables
  - Tablespaces for the history tables created with DEFINE NO – no space is consumed until row versioning is activated for a given security table

Implementing row versioning for security tables in catalog

- Optional, and done on a table-by-table basis
- Example – for SYSTABAUTH
  - Note: successful execution of this ALTER requires that privilege set include SECADM authority (regardless of SEPARATE_SECURITY setting) – same is true for ALTER with DROP VERSIONING

```
ALTER TABLE SYSIBM.SYSTABAUTH
  ADD VERSIONING
  USE HISTORY
  TABLE SYSIBM.SYSTABAUTH_H
  ON DELETE ADD EXTRA ROW;
```

This clause enables recording of "who" (i.e., which auth ID) information when an authorization or privilege is revoked

Some additional items of information

- Clean-up of security history tables: DELETE is not allowed
  - Instead, you use REORG with DISCARD
  - Doing that requires SECADM authority (regardless of SEPARATE_SECURITY setting), plus REORG privilege
- When row versioning has been enabled for a security-related catalog table, a GRANT or a REVOKE affecting the table will fail if the data change drives a history table insert and that insert fails

Example

Initial review assessment: “The SALARY table is secure, only the ID of the HR application has access to the table”

```
TABAUTH
GRANTOR GRANTEE TCREATOR TTNAME SELECTAUTH UPDATEAUTH SYS_START             SYS_END               GEN_SESSION_USER
------- ------- -------- ------ ---------- ---------- --------------------- --------------------- ----------------
HRDBA   HRUSR   HR       SALARY G          G          2022-12-06-10.33...   9999-12-30-00.00...   PETER
```

But what happened on April 25th? Temporal security-related information reveals privilege-abuse activity. Bad guys busted!

16:56 1 ANDREW used his DB2ADM group membership to grant SELECT for a salary investigation to SMITH

18:07 2 The salary seemed way too low compared to his friends, so ANDREW granted UPDATE to resolve. First version of the row (SELECT) is archived.

```
  TABAUTH_H
  GRANTOR GRANTEE TCREATOR TTNAME SELECTAUTH UPDATEAUTH SYS_START             SYS_END               GEN_SESSION_USER
  ------- ------- -------- ------ ---------- ---------- --------------------- --------------------- ----------------
2 DB2ADM  SMITH   S133     SALARY Y          -          2024-04-25-16.56...   2024-04-25-18.07...   ANDREW_THE_ADMIN
3 DB2ADM  SMITH   S133     SALARY Y          Y          2024-04-25-18.07...   2024-04-25-18.10...   ANDREW_THE_ADMIN
4 DB2ADM  SMITH   S133     SALARY Y          -          2024-04-25-18.10...   2024-04-25-19.14...   ANDREW_THE_ADMIN
4 DB2ADM  SMITH   S133     SALARY Y          -          2024-04-25-19.14...   2024-04-25-19.14...   HENRY_THE_HIDER
```

18:10 3 ANDREW revokes the UPDATE rights to erase his traces. Row with UPDATE right is archived.

19:14 4 Already at home, ANDREW realized that he had forgotten to revoke the SELECT rights. He called his admin friend Henry to do it for him (revoker of privilege captured when versioning enabled with ON DELETE ADD EXTRA ROW).

Robert Catterall

rfcatter@us.ibm.com