

Accessing Non-Relational Data with SQL

Dan Cruikshank
dcrank@us.ibm.com

1

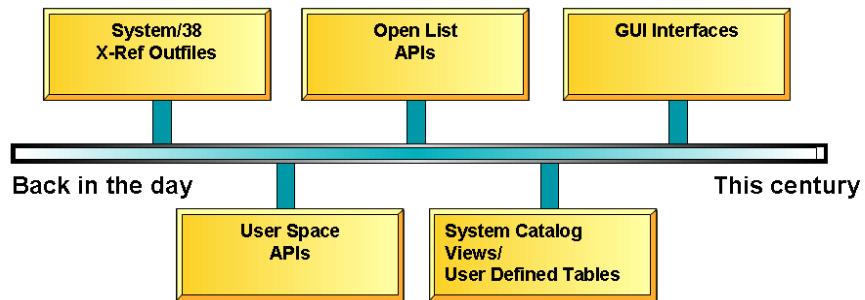
Agenda

- Shortcuts to the good stuff
 - The evolution of metadata
 - Where's the beef?
- Building your own views
 - i-openers using Open List apis
 - Displaying and modifying objects using SQL

2

The Evolution of Metadata

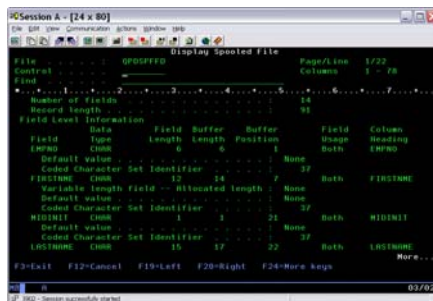
- Metadata is data about data
 - Tables, views, indexes, procedures
 - Relationships
 - Statistics
 - Etc.



3

System/38 Xref Commands

- Command output could be directed to:
 - Green Screen Display File
 - Spool File
 - Output files
- Example
 - DSPFFD
DB2_SAMPLE/EMPLOYEE
OUTPUT (*, *PRINT or *OUTFILE)
- Output files could be analyzed using queries:
 - Where used
 - How used?
 - How many?
 - Etc.



4

Outfile Templates

- A file containing the format of the actual file generated using the *OUTFILE option
- The templates are stored in QSYS
- They are traditional physical file (PF) objects
- Example of some DSPFD *OUTFILE templates

And I still use them, by golly!



File Name	Description
QAFDACCP	System outfile for DSPFD TYPE(*ACCPATH)
QAFDBASI	System outfile for DSPFD TYPE(*BASATR)
QAFDCST	System outfile for DSPFD TYPE(*CST) FILEATR(*PHY)
QAFDJOIN	System outfile for DSPFD TYPE(*JOIN)
QAFDMBR	System outfile for DSPFD TYPE(*MBR)
QAFDMBRL	System outfile for DSPFD TYPE(*MBRLIST)
QAFDRFMT	System outfile for DSPFD TYPE(*RCDFMT)
QAFDSELO	System outfile for DSPFD TYPE(*SELECT)
QAFDTRG	System outfile for DSPFD TYPE(*TRG) FILEATR(*PHY)

5

Database Modernization 101

- What if the *OUTFILE physical file template was used to create a DB2 table?
- Would the DSP command no longer work?
- Is there a benefit?
- Let's find out.

6

Reverse Engineering Legacy to DB2

- Use template as traditional PF
CL:DSPFD FILE(DBRSHEMA/QLABSRC)
TYPE(*MBRLIST) OUTPUT(*OUTFILE)
OUTFILE(DB2SANDBOX/DSPFD_DDS);



And it can
partitioned!

- Use template to create DB2 PF (table)

```
CREATE TABLE DB2SANDBOX.DSPFD_DDL  
LIKE QSYS.QAFDMBRL RCD_FMT  
QWHFDML;
```

```
CL:DSPFD FILE(DBRSHEMA/QLABSRC)  
TYPE(*MBRLIST) OUTPUT(*OUTFILE)  
OUTFILE(DB2SANDBOX/DSPFD_DDL);
```

RCD_FMT Added in V5R4

- Joblog results
Output file DSPFD_DDS created in library
DB2SANDBOX.
Member DSPFD_DDS added to output file
DSPFD_DDS in library DB2SANDBOX.
21 records added to member DSPFD_DDS in
file DSPFD_DDS in DB2SANDBOX.
Statement ran successfully

- Joblog results

Statement ran successfully

21 records added to member DSPFD_DDL in
file DSPFD_DDL in DB2SANDBOX.
Statement ran successfully

It
works!

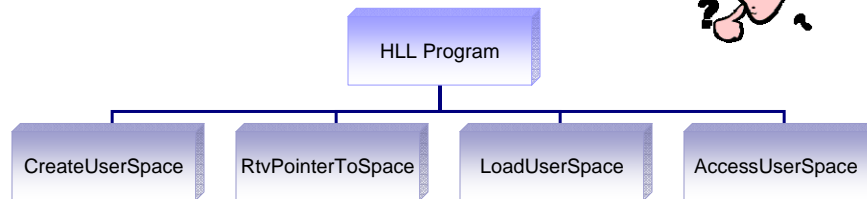


So what did we just learn?

- Moving to DB2 does not require mass program changes
- We protect our investment in legacy applications still using traditional methods
- We can take advantage of new DB2 only feature and function
 - Data Centric Development
 - Auto generation (identity, row change timestamp, etc)
 - Partitioned tables
 - Better security and ease of use
 - E.g. Implicitly hidden columns
 - New column types (lobs, XML, etc)
- But, some applications cannot keep up
 - not all DSP commands have *OUTFILE capability
 - and some *OUTFILE formats do not contain latest information

User Space APIs

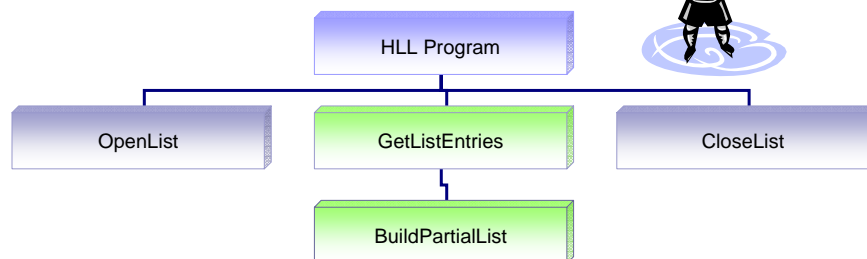
- Application Programming Interface to the good stuff
 - IBM response to growing demand for more output file capability
- User space uses less resource than *OUTFILE
 - No temporary file required
- Requires advanced programming skills
 - Understanding of API programming concepts
 - Pointers
- And it may take a long time to populate the space



9

Open List APIs

- Creates partial list of messages, spool files, objects, etc
 - No waiting for entire list to be created
- Most list APIs part of IBM i as of 5.3
 - Host Servers feature not required
- Superstar programming skills
 - Be the envy of your peers



10

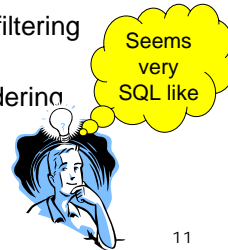
Differences between User Space and Open List APIs

■ User Space

- User space is 16MB
- Control is not returned until:
 - All data has been retrieved
 - User space fills up
- Limited filtering capability
- No ordering capability

■ Open List

- Open list is 2GB
- Control is returned when:
 - Initial number of entries is met
 - Immediately if 0 entries requested
- Enhanced filtering capabilities
- Multiple ordering capabilities

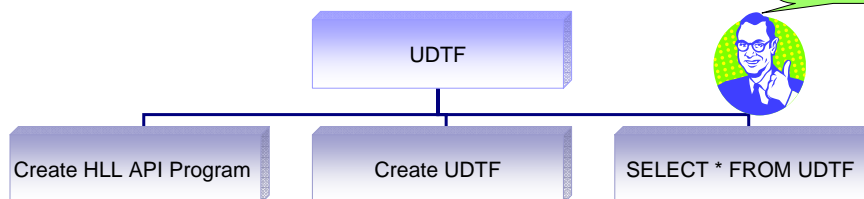


11

User Defined Table Function (UDTF)

- A UDTF returns a table from a non-relational source
 - Typically a HLL written program or service program
 - Can be written in SQL
- Allows non-relational data to be accessed via SQL queries
- Data can be accessed anywhere SQL is allowed to run
 - STRSQL, Run SQL Scripts, ODBC/JDBC, .net, etc.

That's what I'm talking about



12

IBM Supplied UDTF's

- Several UDTFs exist in library QSYS2 allowing on-demand access to system data via SQL

- INDEX_PARTITION_STATISTICS
- MQT_PARTITION_STATISTICS
- OBJECT_STATISTICS
- PARTITION_STATISTICS
- PROGRAM_STATISTICS

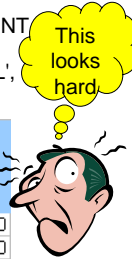
Partial Listing

Review QSYS2 for more

- Example

```
SELECT INDEX_NAME, COLUMN_NAMES, QUERY_STATISTICS_COUNT,
       QUERY_USE_COUNT
FROM TABLE (QSYS2.INDEX_PARTITION_STATISTICS('DC_DB2SMPL',
       'DEPARTMENT'))
```

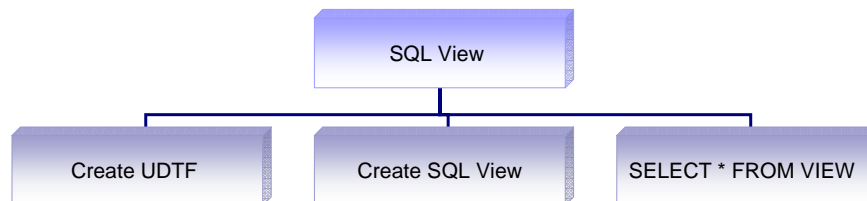
INDEX_NAME	COLUMN NAMES	QUERY STA-TISTICS COUNT	QUERY USE COUNT
Q_DC_DB2SMPL_DEPARTMENT_DEPTNO_00001	DEPTNO	4	0
	MGRNO	0	0
ROD	ADMRDEPT	1	0



13

SQL Views

- An SQL View is a container for an SQL statement
 - Contains no data-it is a virtual table
 - Masks the complexity of the underlying data model
- Can be used with a UDTF



14

IBM Supplied System Catalog Views

- Several SQL views exist to provide fast access to system information
- Three classes of catalog views:
 - IBM i specific - contained in QSYS2
 - ODBC and JDBC – contained in SYSIBM
 - ANSI and ISO – contained in SYSIBM
- The following are some QSYS2 views built over IBM supplied UDTFs:
 - SYSPARTITIONDISK
 - SYSPARTITIONINDEXES
 - SYSPARTITIONINDEXSTAT
 - SYSPARTITIONMQTS
 - SYSPARTITIONSTAT

Partial Listing
Review QSYS2 for more

■ Example SQL

```
SELECT INDEX_NAME, COLUMN_NAMES, QUERY_STATISTICS_COUNT, QUERY_USE_COUNT
FROM QSYS2.SYSPARTITIONINDEXSTAT
WHERE TABLE_SCHEMA = 'DC_DB2SMPL' AND TABLE_NAME = 'DEPARTMENT' ;
```

I'm too busy to type.
There has to be a
easier way!

INDEX_NAME	COLUMN_NAMES	QUERY STATISTICS COUNT	QUERY USE COUNT
Q_DC_DB2SMPL_DEPARTMENT_DEPTNO_00001	DEPTNO	4	0
RDE	MGRNO	0	0
ROD	ADMRDEPT	1	0



GUI Interfaces

- System i Navigator
 - Pop-ups and drill downs
- Show Indexes Example

That's so
easy
even I
can do it

SQL Name	Table Short Name	Key Columns	Query Statistics Use	Query Use Count
Q_DC_DB2SMPL_DEPARTMENT_DEPTNO_00001	DEPARTMENT	DEPTNO	4	0
RDE	DEPARTMENT	MGRNO	0	0
ROD	DEPARTMENT	ADMRDEPT	1	0

Checkpoint-Where's the Beef?

- DSPxx does not contain all information
 - Logical page size, query statistics, etc
- API programming is not for the faint of heart
- GUI Interfaces are continuously enhanced
 - Simple but limited access
 - Cumbersome for the power user
- System Catalog Statistical Views
 - This is where it's at

17

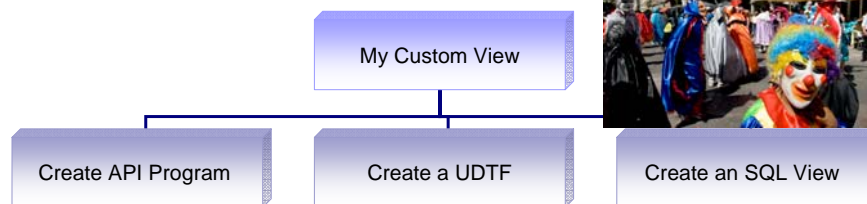
Agenda

- Shortcuts to the good stuff
 - ☑ The evolution of metadata
 - ☑ Where's the beef?
- Building your own views
 - i-openers using Open List apis
 - Displaying and modifying objects using SQL

18

Building Your Own Views

- What if an IBM supplied view or UDTF:
 - Is not complete?
 - Does not exist for some object types
 - Spool and message files, jobs, etc.
- Then build your own



19

Steps to Building your own System View

- Choose your API
- Write a program in your HLL of choice
 - Code the UDTF interface parameters
- Create the UDTF
- Wrap the UDTF in an SQL view or views
 - One API can return multiple types of lists

20

Choose Your API

- Below are some of the Open List APIs:
 - Open List of Job Log Messages (QGYOLJBL)
 - Open List of Messages (QGYOLMSG)
 - Open List of Objects (QGYOLOBJ)
 - Open List of Printers (QGYRPRTL)
 - Open List of Spooled Files (QGYOLSPL)
- Some other interesting APIs
 - Retrieve Journal Information (QjoRetrieveJournalInformation)
 - Retrieve User Information (QSYRUSRI)

21

API Concepts

- List APIs use external spaces to hold the list data
- Retrieve APIs use program defined receiver variables to hold the retrieved data
- Parameter, list and receiver formats are defined in structures
- IBM library QSYSINC
 - contains source include files for all APIs
 - Contain structure definitions for parameters, lists and receiver variables

22

Includes for QGY* APIs

- Get member list from system catalog view

```
SELECT TABLE_NAME,
       TABLE_PARTITION
FROM QSYS2.SYSPARTITIONSTAT
V1
WHERE TABLE_SCHEMA =
      'QSYSINC'
      AND TABLE_NAME =
      'QRPGLSRC'
      AND TABLE_PARTITION LIKE
      'QGY%';
```

- Above statement executed from Rational Developer for Power
 - Output exported to Excel

TABLE	
TABLE NAME	PARTITION
QRPGLSRC	QGYFNDF
QRPGLSRC	QGYOLAFA
QRPGLSRC	QGYGTLE
QRPGLSRC	QGY
QRPGLSRC	QGYOLJOB
QRPGLSRC	QGYOLOBJ
QRPGLSRC	QGYOLSPL
QRPGLSRC	QGYOLMSG
QRPGLSRC	QGYOLJBL
QRPGLSRC	QGYRATLO
QRPGLSRC	QGYRHRCM
QRPGLSRC	QGYRPRTA
QRPGLSRC	QGYRTVSJ
QRPGLSRC	QGYRPRTL



This is just like what I learned in MySQL class!

RPGLE QSYSINC Code Example

- The following is sample code from member QGYOLOBJ

```
D*****
D*      Type Definition for the Receiver Variable
D*NOTE: This definition only defines fixed portions of the
D*      format. Any varying length fields must be specified
D*      by the user.
D*****
DQGYORV01      DS
D*      Qgy Olobj RecVar
D QGYON00      1      10
D QGYOL03      11     20
D QGYOT        21     30
D QGYIS05      31     31
D QGYERVED69   32     32
D QGYNBRFR02   33     36B 0
```

- System code is not very user friendly
 - Consider copying include members and modifying to your tastes

Open List API Process

- Open the list using an API from prior slide
 - Provides a “Request Handle” for subsequent access of the list data
- Get data from the list
 - Get List Entries (QGYGTLE) API
 - “Request Handle” determines which list to access
- Close the list
 - Close List (QGYCLST) API
 - “Request Handle” determines which list to close

25

The User Defined Table Function and APIs

- Creating a UDTF
- Interaction with the HLL Program
 - Mapping Input and Output Parameters
 - Flow Control
 - Returning Data
- Example SQL to access data from the UDTF

26

Creating a UDTF

- The SQL CREATE FUNCTION is used to create the UDTF
- Input variables are defined first
- RETURNS TABLE makes this a UDTF instead of a function
- EXTERNAL NAME identifies the HLL program or service program and entry point
 - The HLL program does not need to exist at this time
- The SQL is converted to C code and compiled
 - PROGRAM TYPE SUB produces a service program
- PARAMETER STYLE SQL results in additional parameters being sent to the HLL
 - CALL TYPE (Open, Fetch, Close)

```

CREATE FUNCTION Object_Detail (
  Inp_Object VARCHAR(10),
  Inp_Object_Library VARCHAR(10),
  Inp_Object_Type VARCHAR(10))
RETURNS TABLE (
  Object_name CHAR(10),
  Object_library_name CHAR(10),
  Object_type CHAR(10), ...)
EXTERNAL NAME
  OPNLSTOBJ(OPEN_LIST_OF_OBJECTS)
LANGUAGE RPGLE
PROGRAM TYPE SUB
PARAMETER STYLE SQL
    
```

Mapping Input and Output Parameters

- UDTF

```

CREATE FUNCTION Object_Detail (
  Inp_Object VARCHAR(10),
  Inp_Object_Library VARCHAR(10),
  Inp_Object_Type VARCHAR(10))

RETURNS TABLE (Object_name CHAR(10),
  Object_library_name CHAR(10),
  Object_type CHAR(10), ...)

EXTERNAL NAME
  OPNLSTOBJ(OPEN_LIST_OF_OBJECTS)

LANGUAGE RPGLE
PROGRAM TYPE SUB

PARAMETER STYLE SQL
    
```

```

■ RPGLE Program OPNLSTOBJ
D Open_List_Of_Objects... PR

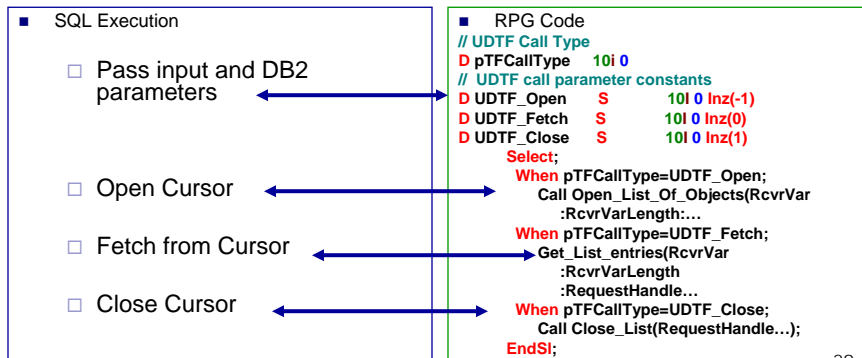
D Inp_Object 10A *VARSIZE
D Inp_ObjLib 10A *VARSIZE
D Inp_ObjectType 10A *VARSIZE)
D Out_QGYON00 LIKE(QGYON00)
D Out_QGYOL03 LIKE(QGYOL03)
D Out_QGYOT LIKE(QGYOT)

// Null Indicator Input Parameters
D NI_Input_1 LIKE(UDTF_NULLIND)
D NI_Input_2 LIKE(UDTF_NULLIND)
D NI_Input_3 LIKE(UDTF_NULLIND)
// Null Indicator Output Parameters
D NI_Output_1 LIKE(UDTF_NULLIND)
D NI_Output_2 LIKE(UDTF_NULLIND)
D NI_Output_3 LIKE(UDTF_NULLIND)

// SQL Style Parameters
D pSQLState 5A
D pFunctionName 517A CONST *VARSIZE
D pSpecificName 128A CONST *VARSIZE
D pSQLMsgText 000A *VARSIZE
// UDTF Call Type
D pTFCallType 10i 0
    
```

Flow Control

- DB2 for i will automatically pass a parameter to the HLL program specifying the type of call
 - Open, Fetch, Close
- An RPG Select statement is used to process the DB2 for i call type
- The following diagrams show how the UDTF works with the HLL program
 - `SELECT OBJECT_NAME, OBJECT_SIZE FROM TABLE(Object_Details('D*', 'MYLIB', '*FILE'))`



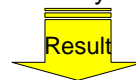
29

Returning Data

- | | |
|---|---|
| <p>DB2 has control</p> <ul style="list-style-type: none"> ■ On First Call <ul style="list-style-type: none"> □ Call Open List API □ Create cursor ■ On Fetch If first FETCH <ul style="list-style-type: none"> □ Set API Variables □ Call Get List Entry ■ On second and subsequent FETCH <ul style="list-style-type: none"> □ Call Get List Entry ■ When all rows fetched <ul style="list-style-type: none"> □ Signal End of Table ■ On Close | <p>The HLL is a slave to SQL</p> <ul style="list-style-type: none"> ■ Create Internal Buffer <ul style="list-style-type: none"> □ Open Data Path (ODP) ■ Move first row to ODP ■ Fill ODP ■ ODP buffer is now ready for consumption |
|---|---|

OBJECT NAME	LIBRARY	OBJECT TYPE	TEXT DESCRIPTION	OBJECT SIZE
CUST_MAST	ACME_DB2	*FILE		364544

DB2_TABLE	ACME_DB2	*FILE	Skeleton for new physical di	73726
DB2_100001	ACME_DB2	*FILE		188416
DB2_100002	ACME_DB2	*FILE		188416
EVFEVENT	ACME_DB2	*FILE		385024
FORMATR	ACME_DB2	*FILE		53248
JORDOTL	ACME_DB2	*FILE	Join ORDDTL to PARTMST	32768
JORDOTL_X1	ACME_DB2	*FILE		184320



30

Accessing Data using a UDTF

- List of journal objects

```
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE, TEXT_DESCRIPTION,
       BIGINT(OBJECT_SIZE *
              OBJECT_SIZE_MULTIPLIER) AS OBJECT_SIZE
FROM TABLE (OBJECT_DETAIL('ALL', 'ALL',
                           'JRN')) AS QGYOLOBJ
ORDER BY OBJECT_NAME;
```

OBJECT				
OBJECT	LIBRARY	OBJECT	TEXT DESCRIPTION	OBJECT SIZE
OBJECT NAME	NAME	TYPE		
QSQRN	ACME_DB2	*JRN	COLLECTION - created by SQL	16384
QSQRN	ACME_400	*JRN		12288

- List of journal receiver objects

```
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE, TEXT_DESCRIPTION,
       BIGINT(OBJECT_SIZE *
              OBJECT_SIZE_MULTIPLIER) AS OBJECT_SIZE
FROM TABLE (OBJECT_DETAIL('ALL',
                           'ACME_DB2', 'JRNRCV')) AS QGYOLOBJ
ORDER BY OBJECT_SIZE;
```

OBJECT				
OBJECT	LIBRARY	OBJECT	TEXT DESCRIPTION	OBJECT SIZE
NAME	NAME	TYPE		SIZE
QSQRN1040	ACME_DB2	*JRNRCV	COLLECTION - created by SQL	159744

- List of file objects

```
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE,
       BIGINT(OBJECT_SIZE *
              OBJECT_SIZE_MULTIPLIER) AS OBJECT_SIZE,
       JOURNAL_NAME, JOURNAL_LIBRARY_NAME
FROM TABLE (OBJECT_DETAIL('ALL',
                           'ACME_DB2', 'FILE')) AS QGYOLOBJ
WHERE SUBSTR(JOURNAL_NAME,1,1) <> ' '
ORDER BY OBJECT_NAME;
```

OBJECT					
OBJECT	LIBRARY	OBJECT	OBJECT SIZE	JOURNAL	JOURNAL
NAME	NAME	TYPE		NAME	LIBRARY
CUST_MAST	ACME_DB2	*FILE	364544	QSQRN	ACME_DB2
DB2_TABLE	ACME_DB2	*FILE	73728	QSQRN	ACME_DB2
LEGACYFILE	ACME_DB2	*FILE	20672	QSQRN	ACME_DB2
NEW_TABLE	ACME_DB2	*FILE	73728	QSQRN	ACME_DB2

31

Wrapping an SQL View Around the UDTF

- List of journal objects

```
CREATE VIEW JOURNAL_OBJECTS AS
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE, TEXT_DESCRIPTION,
       BIGINT(OBJECT_SIZE * OBJECT_SIZE_MULTIPLIER)
       AS OBJECT_SIZE
FROM TABLE (OBJECT_DETAIL('ALL', 'ALL', 'JRN'))
AS QGYOLOBJ;
```

```
SELECT * FROM JOURNAL_OBJECTS
ORDER BY OBJECT_NAME;
```

OBJECT				
OBJECT	LIBRARY	OBJECT	TEXT DESCRIPTION	OBJECT SIZE
OBJECT NAME	NAME	TYPE		
QSQRN	ACME_DB2	*JRN	COLLECTION - created by SQL	16384
QSQRN	ACME_400	*JRN		12288

- List of journal receiver objects

```
CREATE VIEW JOURNAL_RECEIVERS AS
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE, TEXT_DESCRIPTION,
       BIGINT(OBJECT_SIZE * OBJECT_SIZE_MULTIPLIER)
       AS OBJECT_SIZE
FROM TABLE (OBJECT_DETAIL('ALL', 'ACME_DB2',
                           'JRNRCV')) AS QGYOLOBJ;
```

```
SELECT * FROM RECEIVER_OBJECTS
ORDER BY OBJECT_SIZE;
```

OBJECT				
OBJECT	LIBRARY	OBJECT	TEXT DESCRIPTION	OBJECT SIZE
NAME	NAME	TYPE		SIZE
QSQRN1040	ACME_DB2	*JRNRCV	COLLECTION - created by SQL	159744

- List of file objects

```
CREATE VIEW FILE_OBJECTS AS
SELECT OBJECT_NAME, OBJECT_LIBRARY_NAME,
       OBJECT_TYPE,
       BIGINT(OBJECT_SIZE * OBJECT_SIZE_MULTIPLIER)
       AS OBJECT_SIZE,
       JOURNAL_NAME, JOURNAL_LIBRARY_NAME
FROM TABLE (OBJECT_DETAIL('ALL', 'ACME_DB2',
                           'FILE')) AS QGYOLOBJ
WHERE SUBSTR(JOURNAL_NAME,1,1) <> ' ';
```

```
SELECT * FROM FILE_OBJECTS
WHERE SUBSTR(JOURNAL_NAME,1,1) <> ' '
ORDER BY OBJECT_NAME;
```

OBJECT					
OBJECT	LIBRARY	OBJECT	OBJECT SIZE	JOURNAL	JOURNAL
NAME	NAME	TYPE		NAME	LIBRARY
CUST_MAST	ACME_DB2	*FILE	364544	QSQRN	ACME_DB2
DB2_TABLE	ACME_DB2	*FILE	73728	QSQRN	ACME_DB2
LEGACYFILE	ACME_DB2	*FILE	20672	QSQRN	ACME_DB2
NEW_TABLE	ACME_DB2	*FILE	73728	QSQRN	ACME_DB2

32

A UDTF Data Model

- The following tables are from QGYOLOBJ API:
 - Journal Objects
 - Receiver Objects
 - File Objects
- Journal Information
 - QjoRetrieveJournalInformation
- File Statistics are from IBM i system catalogs
 - SYSTABLESTAT



33

Joining UDTFs and SQL Views

■ Example Join View

```
CREATE VIEW Journal_With_File_Information AS
SELECT Journal_library_name, Journal_name, table_name, System_Table_Name,
Object_Size, number_rows, number_deleted_rows, INSERT_operations,
UPDATE_operations, DELETE_operations, CLEAR_operations, REORGANIZE_operations,
Starting_Journal_receiver_Name_for_apply_First_Jrn_Rcvr,
Starting_Journal_receiver_Library_name_First_Jrn_Rcvr_Lib, TBL.Days_used_count
FROM TABLE(object_detail('*ALL', '*ALLUSR', '*FILE')) as OBJ
JOIN Table(Retrieve_Journal_Info(Journal_Library_name,Journal_name))as JRN
ON Journal_Library_name = jrn_lib_name
AND Journal_name = jrn_name
JOIN QSYS2.SYSTABLESTAT TBL
ON SYSTEM_TABLE_SCHEMA = Object_Library_Name
AND SYSTEM_TABLE_NAME = Object_Name;
```

UDTF

Catalog View



■ Sample SQL statement

```
SELECT * FROM Journal_With_File_Information
WHERE Journal_Library_name = 'ACME_DB2' AND Journal_name = 'QSQRN'
ORDER BY number_rows DESC;
```

JOURNAL LIBRARY NAME	JOURNAL NAME	TABLE NAME	SYSTEM TABLE NAME	OBJECT SIZE	NBR ROWS	NBR DELETE D ROWS	INSERT OPS	UPDATE OPS	DELETE OPS	CLEAR OPS	REORGA NIZE OPS	FIRST JRN RCVR	FIRST JRN RCVR LIB	DAYS USED COUNT	
ACME_DB2	QSQRN	ORDR_DETL	ORDR_DETL	73728	70	0	0	0	0	0	0	0	QSQRN0035	ACME_DB2	3
ACME_DB2	QSQRN	ORDR_HEAD	ORDR_HEAD	94208	54	0	0	0	0	0	0	0			2
ACME_DB2	QSQRN	PART_MAST	PART_MAST	73728	50	0	0	0	0	0	0	0	QSQRN0035	ACME_DB2	2
ACME_DB2	QSQRN	DB2_TABLE	DB2_TABLE	73728	18	0	0	0	0	0	0	0			0
ACME_DB2	QSQRN	CUST_MAST	CUST_MAST	364544	0	0	0	0	0	0	0	0			0
ACME_DB2	QSQRN	NEW_TABLE_FROM_DDS	NEW_TABLE	73728	0	0	0	0	0	0	0	0	QSQRN0035	ACME_DB2	0
ACME_DB2	QSQRN	ORDER_HISTORY	ORDR_HIST	365024	0	0	0	0	0	0	0	0			0
ACME_DB2	QSQRN	QUACKINI	QUACKINI	200896	0	0	0	0	0	0	0	0			0

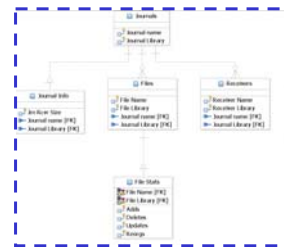
34

Modifying Objects Using SQL Instead of Triggers

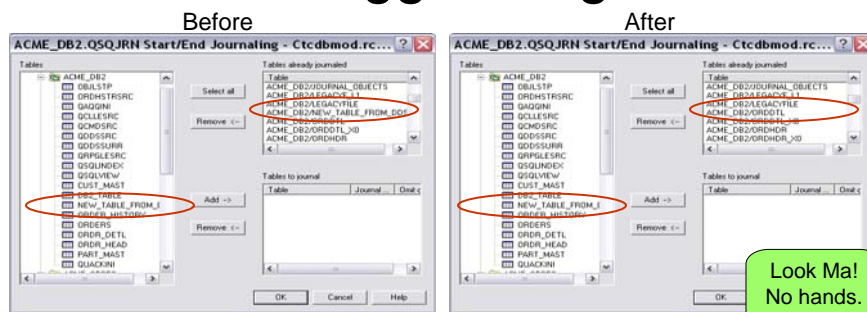
- The icing on the cake
- An Instead of Trigger enables updates/deletes or inserts for SQL views which are not considered updateable, delete-able or insert-able
 - A view of a UDTF is one such view
- The following Instead of Trigger would transform an SQL update of a view into a call to QCMDEXC
 - This would allow object changes via standard SQL UPDATE statement

```
CREATE TRIGGER END_JOURNAL_OF_FILE
INSTEAD OF DELETE ON Journal_With_File_Information
REFERENCING OLD Old_row FOR EACH ROW
BEGIN ATOMIC
  DECLARE v_Command_Length INTEGER;
  DECLARE v_Command_String VARCHAR(1000);
  SET v_Command_String = 'ENDJRNPF (' ||
  RTRIM(Journal_library_name) ||
  '/' || RTRIM(System_Table_Name) || ')';
  SET v_Command_Length = LENGTH(v_Command_String);
  CALLQSYS2.QCMDEXC(v_Command_String,v_Command_Length);
END ;
```

View Journal_With_File_Information



Instead of Trigger Magic



- The following statement:
**DELETE FROM Journal_With_File_Information
 WHERE Journal_library_name = 'ACME_DB2'
 AND Journal_name = 'QSQJRN'
 AND table_name = 'NEW_TABLE_FROM_DDS';**
- Executes the following command:
ENDJRNPF(QSQJRN/NEW_TABLE)



Where do we go from here?

- Read a manual
 - Start with the IBM i Information Center
 - <http://publib.boulder.ibm.com/infocenter/iseres/v6r1m0/index.js>
 - Link to System APIs on first page
- Read a book
 - “IBM System i APIs at Work, Second Edition”
 - Bruce Vining, Doug Pence and Ron Hawkins
- Search the web
 - It's amazing what's out there

37

Questions



38