In association with Common France

COMMON EUROPE CONGRESS 2026

14 - 17 June
Lyon, France

The largest conference in Europe for solutions around IBM Power (IBM i, AIX, Linux) & IBM Storage

Session: Application Modernization Database

For: Common Europe Congress 2026

Date: 06/17/2026 09:15 am

Birgitta Hauser

Modernization – Education – Consulting on IBM i

Diplom-Betriebswirt (BA)
Software and Database Architect
IBM Champion since 2020

eMail: Hauser@ModEdCon.com / Hauser@SSS-Software.de
Website: https://modedcon.com

IBM Champion since 2020

1

Landsberg am Lech

2

Agenda

Today → Future: Modernization Goals
- Moving Business Logic into Database

DDS physical Files versus SQL Tables
- Differences between DDS PF and SQL Tables
- Reverse Engineering
- Dependent Objects

From DDS to DDL
- Convert physical files to SQL Tables
- Create a new Table and a logical file with the same name as the phyiscal file

From logical Files to SQL views/indexes
- Views – Moving Business Logic into the database
- Indexes – incl. enhanced indexing technologies

Unique Identifier
- Identity Column and ROWId
- On demand generation Scalar Functions

Authorization Columns

Data Consistency
- Check Constraints
- Key Constraints / Referential Integrities
- Triggers

Commitment Control

Row and Column Access Control

Externalizing Data Access
- CRUD Functions (for Insert/Update/Delete)
- Wrapped as Micro-Services to be called from outside
- Service-Programs for View Access

3

Today → Future

4

Current Situation

Most IBM i Applications are based on DDS described files

- While programs get rewritten, **database is not upgraded**
  - Except for adding fields to existing files or new logical files
- Over time, **database structure growth**
  - Based on technologies available 30 years ago
  - Redundant data → Programming is required to keep data consistent
  - Too many access paths → Performance Issue
- Newer technologies to keep data consistent such as
  Artifical Keys (Identity Columns), Check Constraints, Referential Integrities, Triggers are **rarely used**
- **Record Level Access (RLA)** is mainly used to access data
  - Data Access is **not externalized** → Inserting/Updating/Deleting the **same file/table** in **many programs**
- **Journaling and Commitment Control are not widespread**

Application Centric Design → Goal: Data Centric = Moving Business Logic into the Database

5

Modernization Goals

Data Centric: Move business logic into database

- Create and use SQL views and User Defined Tabe Functions (UDTF) 
  - Replace temporary tables with views or Materialized Query Tables (MQT)
- Add Check Constraints, Key Constraints  and Referential Integrities 
- Add Triggers 
- Implement Journaling and Commitment Control 
- Use Row and Column Access Control (RCAC) for restricting data access 

Convert database objects from DDS desrciption into SQL

- Convert physical files into SQL Tables 
- Convert (keyed) logical files into SQL Indexes 
- Convert (unkeyed) logical files into SQL Views

Replace Native I/O with embedded SQL and/or use SQL/PL 

Externalize database access

- Write procedures for insert, update and delete  → Use Instead Of Triggers

Redesign your database

- Normalize your database → Remove redundancies 

6

Why to modernize with SQL?

SQL is Standard (ISO/IEC 9075:2023)

- **Portability** of code and skills
  - Basic commands are identical
    → but sometimes result in different dialects
  - All Database Manufacturers committed to cover the standard
    → but cover different levels
- DDS (Data Description Specifications) is only used in IBM i/Db2 for i
  → all other databases use **SQL DDL (Data Definition Language)** for generating Database Objects
- Native I/O only used by RPG and Cobol
  → **all other languages use SQL**

7

Physical Files versus SQL Tables

8

Reasons to convert

DDS is outdated (stablilized since Release V5R3M0)

- All new features are added in SQL (Data Definition Language)
  - Examples: Large Object/Boolean Data Types – Identity Column and other Auditing Columns

Architectural Differences between DDS physical files and SQL Tables

- When **writing into DDS files data** is **not checked** → validation occurs when reading data
- When **writing into SQL table data** is **checked** → no data check when reading data

→ **No invalid Data** can be added into SQL Tables
→ **Faster Reads** and **slower Writes**

Enhanced Indexing Technologies

- SQL Indexes can not be specified in a SQL Statement
  but **SQL indexes** can be used in composition **with native I/O like any keyed logical file**
- With the enhanced indexing technologies (**derived** and **sparse Indexes**)
  more powerful access paths (especially for native I/O) can be built

Differences between DDS Described physical Files and SQL Tables

	DDS	SQL DDL	
Object Name		Long SQL Name	Up to **128 Characters**
	Up to **10 Characters**	Short System Name	Up to **10 Characters**
Column Names		Long SQL Name	Up to **128 Characters**
	Up to **10 Characters**	Short System Name	Up to **10 Characters**
Keys	can be **keyed**: (Compound) **unique and non-unique keys**	**Always unkeyed**	
	Primary and Unique **Key Constraints**	Primary and Unique **Key Constraints only**	
	May result in problems when converting from DDS To DDL! **Non-unique keys access path get lost --> problems with native I/O** **Unique keys will be converted into Primary Key Constraints--> should be reserved for "artificial" unique keys**		
Record Format	**Different File** Name and **Record Format Names** for native I/O	SQL Standard **without Format** Different format can be added with the **RCDFMT keyword**	
REUSEDLT	**Default: *NO** deleted records are not removed new rows are added **at the end of the file**	**Default: *YES** deleted records are not removed but **replaced** with newly **inserted rows**	
	May cause problems after converting from DDS to DDL at least if data have to be read in the sequence of the relative record no		
Architectural Differences	**Data Validation** occurs when a row **is read** no check when a row is written **Invalid (numeric) Data can be inserted!!!**	**Data Validation** occurs when a row **is written** no check when a row is read Invalid(numeric) Data **cannot be inserted**!!!	
	Invalid numeric data in DDS Files will cause crashs and data loss when converting from DDS to DDL **Invalid numeric data must be revised before conversion** **Programs (writing invalid numeric values) must be reworked to avoid future crashes!**		

Keyed physical Files versus SQL Tables

SQL Tables do not allow any Keys

- Unique Key in PF: can be converted into a Primary Key Constraint
- Non unique Key in PF: not supported for SQL Tables

Before conversion:

- Create an **additional** logical File or better **SQL Index** for the Key in the physical File
  - Replace the keyed access with native I/O on the physical file with a keyed access to the new logical File or Index

Convert/Create the SQL Table (without Primary Key Constraint)

- Only **a single Primary Key Constraint** per Physical File/SQL Table
- Should be reserved for an **artificial unique Key** (e.g. Identity Column) for future implementation of Referential Integrities

Physical Files versus SQL Tables

SQL does not Support (DDS) Keywords

- **DATFMT/DATSEP**: **no equivalent** in SQL
  EDTCDE/EDTWRD: **no equivalent** in SQL

Reuse Deleted Records (REUSEDLT)

- Creation **REUSEDLT** DDS Default: ***NO** SQL Default: ***YES**
- Member **SIZE**: DDS Default: **10,000 +Increment** SQL Default: ***NOMAX**

→ **CHGPF** can be executed **after** the **CREATE TABLE** statement

Create Table – Statement - Examples

```
CREATE OR REPLACE TABLE HSCOMMON10.EXTABLE01
      (ExampleId          FOR COLUMN EXID     Integer
                                             Not NULL Generated Always
                                             as Identity(
                                                Start With 1, Increment By 1,
                                                No Order, No Cycle,
                                                No MinValue, No MaxValue,
                                                Cache 20),
       ExampleVarChar     FOR COLUMN EXCHAR   CHAR (20)      Not Null DEFAULT '*NONE',
       ExampleVarying     FOR COLUMN EXVCHAR  VARCHAR(256)   Allocate(50)
                                             Not Null Default '' ,
       ExampleDec         FOR COLUMN EXPACK   DEC(11, 2)     Not Null Default 999999999,99,
       ExampleNum         FOR COLUMN EXZONE   NUMERIC(11, 2),
       ExampleInt         FOR COLUMN EXINT    INTEGER        ,
       ExampleDate        FOR COLUMN EXDATE   DATE           Not NULL Default Current_Date,
       ExampleTime        FOR COLUMN EXTIME   TIME           Not NULL Default Current_Time,
       ExampleTimestamp                       TIMESTAMP      Not NULL Generated Always
                                                             For Each Row on Update
                                                             as Row Change Timestamp,
       ExampleBirthDay    FOR COLUMN EXBIRTH  DATE           Implicitly Hidden)
RcdFmt EXTABLE01F
```

- Long SQL Name
- Short System Name
- Format Name <> Table Name

Identity Column
- Automatically generated at insert

Allocate
- Reserved storage within the row
- Longer values → Overflow area
- > 80% all values

Row Change Timestamp
- Current Timestamp gets stored as soon as any value will be changed within the row

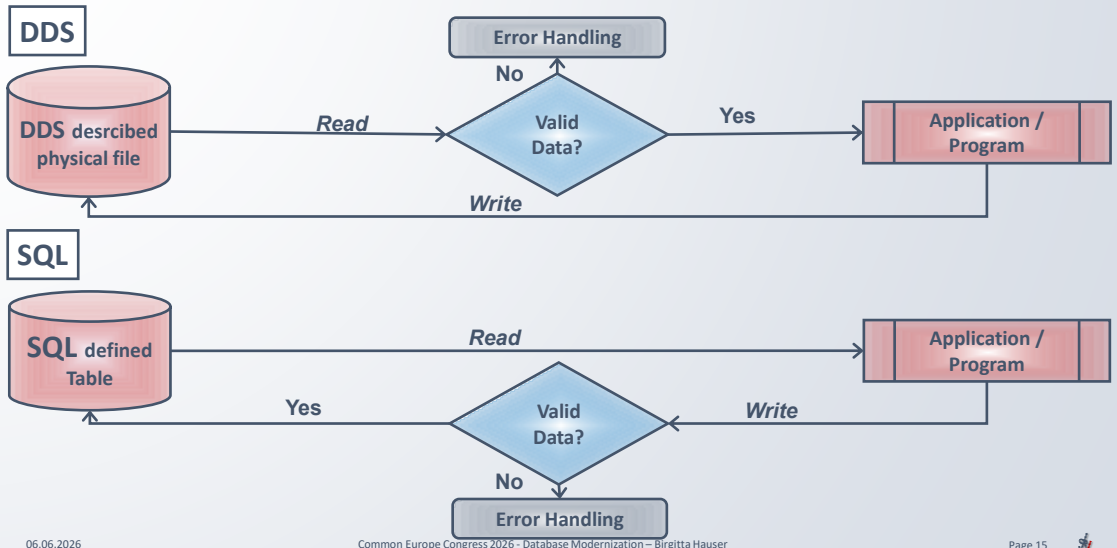
Implicitly Hidden
- Not displayed with SELECT *
- But the column is NOT Protected

13

Invalid numeric values in DDS described Files

14

Architectural Differences between DDS physical files / SQL Tables Data Validation



15

Check Invalid Numeric Values

New RELEASE 7.4 TR 2

3 New Data Validation Services/UDTFs in the SYSTOOLS Schema

- `VALIDATE_DATA` Validate **all rows** within a **single member**
  Special values `*FIRST/*LAST` allowed as member name
- `VALIDATE_DATA_FILE` Validate **all rows** in **all members** within **one file**
- `VALIDATE_DATA_LIBRARY` Validate **all rows** in **all members** in **all files** in **one schema**

Analyzes numeric Values and returns Information about Invalid Data

- Returns: **Relative Record No** and **Column Name** of the invalid Data
- If **no data is returned**, the member, file and/or library **does not include invalid Data**

16

Check for invalid numeric Data in a DDS File/Member

```
Select *
   from Table(SysTools.Validate_Data(Library_Name => 'COMDBMOD',
                                     File_Name    => 'BASEDDS',
                                     Member_Name  => '*FIRST')) x;
```

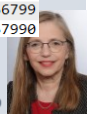
- Invalid Data in the *FIRST member of the BASEDDS file in the COMDBMOD library

LIBRARY_NAME	FILE_NAME	MEMBER_NAME	COLUMN_NAME	RELATIVE_RECORD_NUMBER	SQL_WARNING	REASON_CODE	WARNING_TEXT	VALIDATE_TIME
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	4	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.554572
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	5	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.566799
COMDBMOD	BASEDDS	BASEDDS	FLNUM2	6	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.567990

```
Select *
   from Table(SysTools.Validate_Data_File(Library_Name => 'COMDBMOD',
                                          File_Name    => 'BASEDDS')) x;
```

- Invalid Data in the BASEDDS file in the COMDBMOD library

LIBRARY_NAME	FILE_NAME	MEMBER_NAME	COLUMN_NAME	RELATIVE_RECORD_NUMBER	SQL_WARNING	REASON_CODE	WARNING_TEXT	VALIDATE_TIME
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	4	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.554572
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	5	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.566799
COMDBMOD	BASEDDS	BASEDDS	FLNUM2	6	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.567990



17

Determine Invalid Numeric Data within all DDS-Files in a Library

```
Select *
   from Table(SysTools.Validate_Data_Library('COMDBMOD')) x;
```

LIBRARY_NAME	FILE_NAME	MEMBER_NAME	COLUMN_NAME	RELATIVE_RECORD_NUMBER	SQL_WARNING	REASON_CODE	WARNING_TEXT	VALIDATE_TIME
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	4	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.554572
COMDBMOD	BASEDDS	BASEDDS	FLNUM1	5	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.566799
COMDBMOD	BASEDDS	BASEDDS	FLNUM2	6	802	6	Data conversion or data mapping error.	2020-06-26 16:38:02.567990

- Determine all invalid numeric data within the COMDBMOD library
- Attention: Query may run some time!

18

TRY_CAST – Specification

New RELEASE 7.5

```
          SearchCondition
TRY_CAST ( NULL             as DataType_Definition)
          ParameterMarker
```

Converts the specified Data into the specified Data Type Definition

- **Definition analog** to the **CAST specification**
- **Contrary** to the **CAST** specification:
  - If the **data cannot be converted** into the specified data type **a NULL Value is returned**
    The **CAST** specification returns an undefined error
  - **NULL** values can easily be selected

Provides an easy way for detecting invalid numeric values (for ex. *BLANKS) in DDS described tables

Convert Invalid Numeric Values with TRY_CAST

New RELEASE 7.5

```
Select * from BaseDDS;
```

FLNUM1	FLNUM2	FLALFA
123	456,789	XXXX
555	666,777	ABCXXXXX
998	877,665	BCF
++++++++++++	965,123	4567
++++++++++++	0,123	4567
++++++++++++	++++++++++++	4567
987	654,321	LAST

- **Before: Invalid Numeric Data in the FLNUM1 and FLNUM2 Columns**

```
Update BaseDDS a
   set FLNUM1 = Coalesce(Try_Cast(FLNUM1 as Dec(3, 0)), 0),
       FLNUM2 = Coalesce(Try_Cast(FLNUM2 as Dec(6, 3)), 0)
Where exists (Select *
                from BASEDDS
                Where    Try_Cast(FLNUM1 as Dec(3, 0)) is NULL
                      or Try_Cast(FLNUM2 as Dec(6, 3)) is NULL);
```

- **Update:** **Using TRY_CAST for converting the column value into a numeric value**
  For invalid numeric data is a NULL value returned
  With the COALESCE scalar Function the NULL values are converted into the *Zero Default Value

```
Select * from BaseDDS;
```

FLNUM1	FLNUM2	FLALFA
123	456,789	XXXX
555	666,777	ABCXXXXX
998	877,665	BCF
0	965,123	4567
0	0,123	4567
0	0,000	4567
987	654,321	LAST

- **After:** **Invalid Numeric Data is converted into *Zero**
  All other data is not changed

Database Transformation

21

Modernization - Approaches

1. DDS to SQL DDL conversion

- Same Database Objects: Physical File → SQL Table (long SQL Names can be added), Logical File → View or Index

2. New SQL Table and logical File with the same key as the original physical file

- New SQL Table without Key but all columns of the original Physical File (including long SQL Names)
- New Logical File with the same Name and Key as the original Physical File
- Depending database objects: e.g. logical files, views, indexes, triggers must be modified for the new SQL Table

Enhancements (both Approaches):

- New Columns: Identity Column (artificial key) and other authorization columns
  Additional Date/Time Columns for Numeric Date/Time Columns
- Primary Key Constraint: on the Identity Column
- Trigger: To keep numeric Dates and Date/Time Values consistent
  To avoid inserting invalid numeric values
- Indexes: Replace logical files with Indexes or add new indexes with the same keys as the logicals
  New Indexes with Date/Time Columns
- Base View No direct access to the SQL Table

22

Conversion DDS to DDL

Reverse Engineering – generating the SQL Code for existing DDS Files

- IBM i Access Client Solution Wizard or **GENENERATE_SQL** Stored Procedure
- DDS physical Files → SQL **Table** - **CREATE OR REPLACE TABLE**
- DDS logical Files → SQL **View** (Default) - **CREATE OR REPLACE VIEW** - **NO Keys**
  → SQL **Index** - **CREATE INDEX** - **Keys only**

Before running the generated SQL Scripts

- **Check if the physical file is keyed** → SQL **does not support Keys** in Tables
  - Create a new logical file (or better a SQL index) with the same key columns
  - **Rework** your programs with **native I/O** → Replace the physical file with the new logical file
- **Check for and revise invalid (numeric) Values**
  - Invalid numeric values will cause a crash and Data get lost!

If DDS is converted into DDL – without any enhancements – Format Level does not change → Recompile not necessary

CREATE OR REPLACE TABLE

…OR REPLACE

- **Table replacement** → the necessary **set of ALTER** operations is performed
- **Data-Centric** → Data is preserved - Depending on the **ON REPLACE** clause
- **Dependent and linked objects** are preserved
  - Primary/Unique Key Constraints, Referential Integrities, Check Constraints
  - Indexes, Views, MQTs, DDS described logical files
  - SQL and System Triggers, Field Procedures
- **Granted object** and **access authorities** are preserved
  - Row and Column Access Control (RCAC)
- **Comments** and **Labels** preserved
- For **future enhancements** only the **SQL Script** has to be **modified and rerun**

Reverse Engineering

25

Reverse Engineer

Self-Describing Database

- Must be possible to generate the **SQL source code from** the **database object itself**

Access Client Solutions (ACS) Wizard

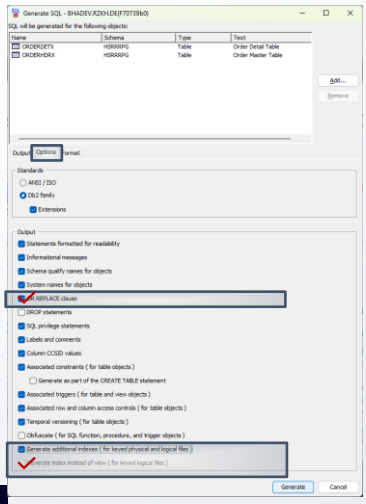
- ACS Schemas include Wizards for retrieving the SQL code for **any database object**
  → Independent whether **SQL defined or DDS described**

GENERATE_SQL Stored Procedure

- Introduced with Release 7.2 TR 1
- Provides the **same/enhanced functionality** as the ACS – Wizard, but can **called from within programs** or SQL routines

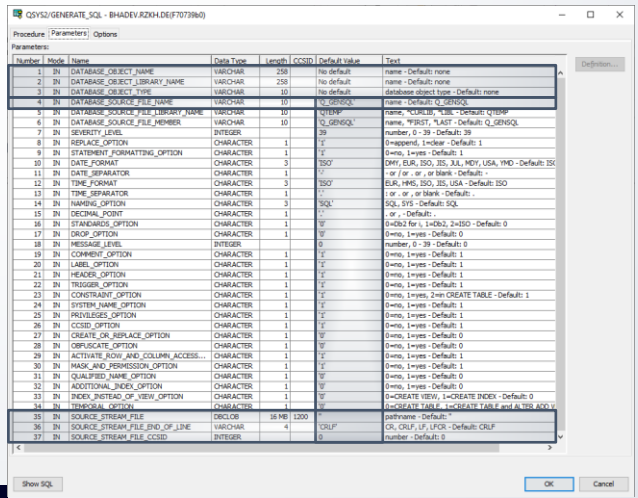
26

Reverse Engineering - Generate SQL with ACS



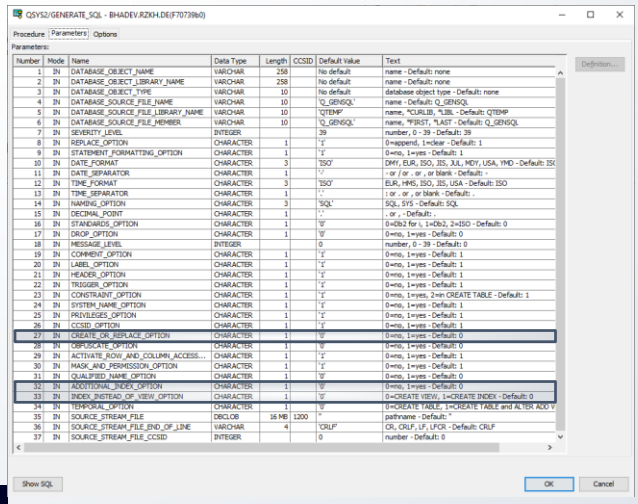
- Generate SQL – **Options**
- Select **OR REPLACE** Clause
- For keyed logical files
  - → Select **Generate Additional Indexes** or
  - → **Generate Index instead of view**

Reverse Engineering - GENERATE_SQL (Stored Procedure)



- **37 Input parameters**
- **3 required parameters:**
  - Database_Object_Name
  - Database_Object_Library_Name
  - Database_Object_Type
- **Default Values defined for all other Parameters**
  - Optional parameters
  - Can be passed as named parameters (Argument List)
- **Output the SQL-Script into an IFS File**
  - Database_Source_File_Name: *STMF
  - Source_Stream_File: IFS File (Path name)
  - Source_Stream_File_End_Of_Line CR, CRLF, LF, LFCR
  - Source_Stream_File_CCSID

Reverse Engineering - GENERATE_SQL (Stored Procedure)



Number	Mode	Name	Data Type	Length	CCSID	Default Value	Text
1	IN	DATABASE_OBJECT_NAME	VARCHAR	258		No default	name - Default: none
2	IN	DATABASE_OBJECT_LIBRARY_NAME	VARCHAR	258		No default	name - Default: none
3	IN	DATABASE_OBJECT_TYPE	VARCHAR	10		No default	database object type - Default: none
4	IN	DATABASE_SOURCE_FILE_NAME	VARCHAR	10		'Q_GENSQL'	name - Default: Q_GENSQL
5	IN	DATABASE_SOURCE_FILE_LIBRARY_NAME	VARCHAR	10		'QTEMP'	name, *CURLIB, *LIBL - Default: QTEMP
6	IN	DATABASE_SOURCE_FILE_MEMBER	VARCHAR	10		'Q_GENSQL'	name, *FIRST, *LAST - Default: Q_GENSQL
7	IN	SEVERITY_LEVEL	INTEGER			39	number, 0 - 39 - Default: 39
8	IN	REPLACE_OPTION	CHARACTER	1		'I'	0=append, 1=clear - Default: 1
9	IN	STATEMENT_FORMATTING_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
10	IN	DATE_FORMAT	CHARACTER	3		'ISO'	DMY, EUR, ISO, JIS, JUL, MDY, USA, YMD - Default: ISO
11	IN	DATE_SEPARATOR	CHARACTER	1		'/'	- or / or . or , or blank - Default: -
12	IN	TIME_FORMAT	CHARACTER	3		'ISO'	EUR, HMS, ISO, JIS, USA - Default: ISO
13	IN	TIME_SEPARATOR	CHARACTER	1		':'	: or . or , or blank - Default: .
14	IN	NAMING_OPTION	CHARACTER	3		'SQL'	SQL, SYS - Default: SQL
15	IN	DECIMAL_POINT	CHARACTER	1		'.'	. or , - Default: .
16	IN	STANDARDS_OPTION	CHARACTER	1		'0'	0=DB2 for i, 1=Db2, 2=ISO - Default: 0
17	IN	DROP_OPTION	CHARACTER	1		'0'	0=no, 1=yes - Default: 0
18	IN	MESSAGE_LEVEL	INTEGER			0	number, 0 - 39 - Default: 0
19	IN	COMMENT_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
20	IN	LABEL_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
21	IN	HEADER_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
22	IN	TRIGGER_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
23	IN	CONSTRAINT_OPTION	CHARACTER	1		'I'	0=no, 1=yes, 2=in CREATE TABLE - Default: 1
24	IN	SYSTEM_NAME_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
25	IN	PRIVILEGES_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
26	IN	CCSID_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
27	IN	CREATE_OR_REPLACE_OPTION	CHARACTER	1		'0'	0=no, 1=yes - Default: 0
28	IN	OBFUSCATE_OPTION	CHARACTER	1		'0'	0=no, 1=yes - Default: 0
29	IN	ACTIVATE_ROW_AND_COLUMN_ACCESS...	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
30	IN	MASK_AND_PERMISSION_OPTION	CHARACTER	1		'I'	0=no, 1=yes - Default: 1
31	IN	QUALIFIED_NAME_OPTION	CHARACTER	1		'0'	0=no, 1=yes - Default: 0
32	IN	ADDITIONAL_INDEX_OPTION	CHARACTER	1		'0'	0=no, 1=yes - Default: 0
33	IN	INDEX_INSTEAD_OF_VIEW_OPTION	CHARACTER	1		'0'	0=CREATE VIEW, 1=CREATE INDEX - Default: 0
34	IN	TEMPORAL_OPTION	CHARACTER	1		'0'	0=CREATE TABLE, 1=CREATE TABLE and ALTER ADD V
35	IN	SOURCE_STREAM_FILE	DBCLOB	16 MB	1200	''	pathname - Default: ''
36	IN	SOURCE_STREAM_FILE_END_OF_LINE	VARCHAR	4		'CRLF'	CR, CRLF, LF, LFCR - Default: CRLF
37	IN	SOURCE_STREAM_FILE_CCSID	INTEGER			0	number - Default: 0

- Important parameters for converting DDS described physical and logical files into SQL defined Database Objects
  - CREATE_OR_REPLACE_OPTION Parameter
  - ADDITIONAL_INDEX_OPTION Parameter
  - INDEX_INSTEAD_OF_VIEW_OPTION Parameter

29

DDS PF to SQL Table

30

DDS to SQL Conversion

Reverse Engineering: SQL Code for SQL Database Objects AND DDS Physical and Logical Files can be generated

Differences between DDS and SQL

- Generated SQL script: **includes comments** for everything that **cannot be translated** or **is interpreted differently**
  → needs to be **checked / reworked**
- Key: **Unique Key** converted into a **Primary Key** constraint
  Not Unique Key is not included in the **CREATE TABLE** Statement
  An **additional Index** can be created
- Keywords: **DATFMT/DATSEP**: **no equivalent** in SQL
  EDTCDE/EDTWRD: **no equivalent** in SQL
- Creation: **REUSEDLT** DDS Default: ***NO** SQL Default: ***YES**
  Member **SIZE**: DDS Default: **10,000 +Increment** SQL Default: ***NOMAX**
  → **CHGPF** can be executed **after the CREATE TABLE** statement

Reverse Engineering - Generate the SQL Code

Generate SQL Statement

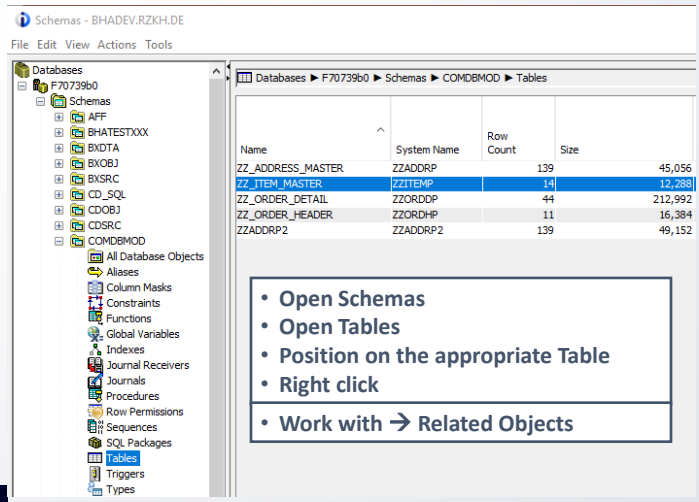
- Using **Access Client Solutions (ACS) Wizard**
- Using the **GENERATE_SQL** stored procedure

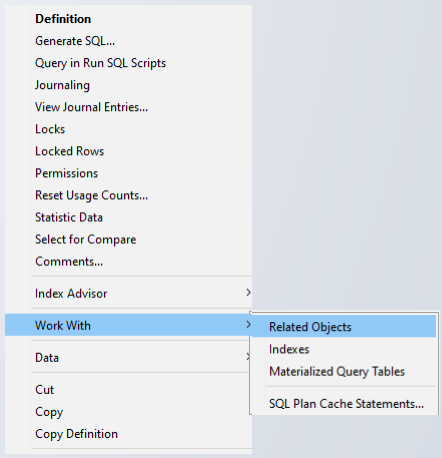
Important (when converting DDS to SQL)

- **CREATE OR REPLACE** clause must be selected / set
  → ACS Wizard: **CREATE OR REPLACE** Clause
  → **GENERATE_SQL**: **CREATE_OR_REPLACE_OPTION** Parameter
- For **keyed physical files additional indexes** should be created:
  → ACS Wizard: **Generate additional indexes (for keyed physical and logcial files)** Option
  → **GENERATE_SQL**: **ADDITIONAL_INDEX_OPTION** Parameter

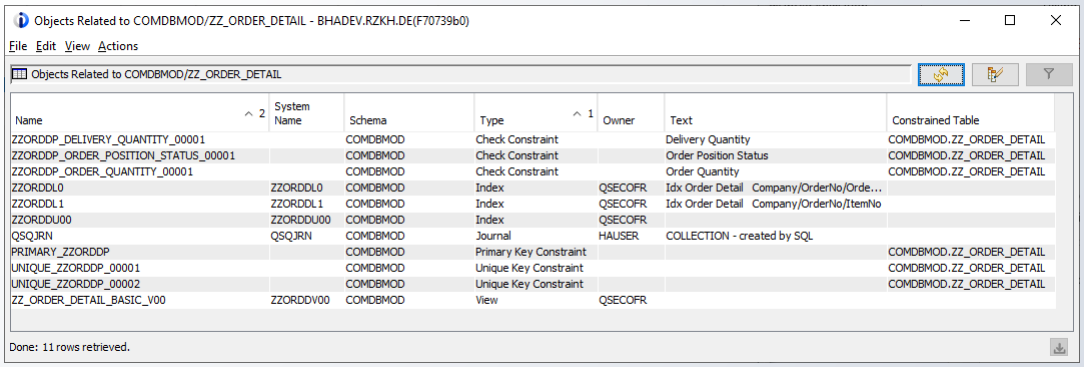
Related Objects

IBM i – Access Client Solutions (ACS) – Schemas – Related Objects





IBM i – Access Client Solutions (ACS) – Schemas – Related Objects



Name	System Name	Schema	Type	Owner	Text	Constrained Table
ZZORDDP_DELIVERY_QUANTITY_00001		COMDBMOD	Check Constraint		Delivery Quantity	COMDBMOD.ZZ_ORDER_DETAIL
ZZORDDP_ORDER_POSITION_STATUS_00001		COMDBMOD	Check Constraint		Order Position Status	COMDBMOD.ZZ_ORDER_DETAIL
ZZORDDP_ORDER_QUANTITY_00001		COMDBMOD	Check Constraint		Order Quantity	COMDBMOD.ZZ_ORDER_DETAIL
ZZORDDL0	ZZORDDL0	COMDBMOD	Index	QSECOFR	Idx Order Detail Company/OrderNo/Orde…	
ZZORDDL1	ZZORDDL1	COMDBMOD	Index	QSECOFR	Idx Order Detail Company/OrderNo/ItemNo	
ZZORDDU00	ZZORDDU00	COMDBMOD	Index	QSECOFR		
QSQJRN	QSQJRN	COMDBMOD	Journal	HAUSER	COLLECTION - created by SQL	
PRIMARY_ZZORDDP		COMDBMOD	Primary Key Constraint			COMDBMOD.ZZ_ORDER_DETAIL
UNIQUE_ZZORDDP_00001		COMDBMOD	Unique Key Constraint			COMDBMOD.ZZ_ORDER_DETAIL
UNIQUE_ZZORDDP_00002		COMDBMOD	Unique Key Constraint			COMDBMOD.ZZ_ORDER_DETAIL
ZZ_ORDER_DETAIL_BASIC_V00	ZZORDDV00	COMDBMOD	View	QSECOFR		

- **Dispays all related objects, including logical files, Indexes, Views, Check Constraints, Key Constraints and Referential Integrities**

RELATED_OBJECTS Table Function in the SYSTOOLS Library

```
RELATED_OBJECTS (Library_Name  => System_Library_Name,
                 File_Name     => System_Table_Name)
```

Returns a list of all Objects that depend on the specified Database File

- Returns all objects that are directly dependent on the input database file:

  Dependent Object Types: `KEYED/UNKEYED LOGICAL FILE, INDEX, VIEW, ALIAS, FOREIGN KEY, MATERIALIZED QUERY TABLE, HISTORY TABLE, TEXT INDEX (Omni-Find), VARIABLE, TRIGGER, PROCEDURE, FUNCTION, MASK, PERMISSON, XML SCHEMA`

- Each dependent **view or global variable** is **processed recursively** until no more dependents are found

Parameters

- 1st Parameter: Library_Name VarChar(10) – System Library Name
- 2nd Parameter: File_Name VarChar(10) – System Table Name

RELATED_OBJECTS Table Function in the SYSTOOLS Library

```
Select *
  from Table(SysTools.Related_Objects(Library_Name => 'COMDBMOD',
                                      File_Name    => 'ZZORDDP'));
```

SOURCE_SCHEMA_NAME	SOURCE_SQL_NAME	SQL_OBJECT_TYPE	SCHEMA_NAME	SQL_NAME	LIBRARY_NAME	SYSTEM_NAME
COMDBMOD	ZZ_ORDER_DETAIL	INDEX	COMDBMOD	ZZORDDU00	COMDBMOD	ZZORDDU00
COMDBMOD	ZZ_ORDER_DETAIL	INDEX	COMDBMOD	ZZORDDL0	COMDBMOD	ZZORDDL0
COMDBMOD	ZZ_ORDER_DETAIL	INDEX	COMDBMOD	ZZORDDL1	COMDBMOD	ZZORDDL1
COMDBMOD	ZZ_ORDER_DETAIL	VIEW	COMDBMOD	ZZ_ORDER_DETAIL_BASIC_V00	COMDBMOD	ZZORDDV00
COMDBMOD	ZZ_ORDER_DETAIL_BASIC_V00	VIEW	COMDBMOD	ZZ_ORDER_DETAIL_ITEM_V01	COMDBMOD	ZZORDDV01
COMDBMOD	ZZ_ORDER_DETAIL_ITEM_V01	VIEW	COMDBMOD	ZZ_ORDER_ADDRESS_ITEM_V02	COMDBMOD	ZZORDHV02

- Displays all dependent Database Objects
- Does not include: Key Constraints and Check Constraints

Logical Files vs SQL Views and Indexes

Logical Files

DDS described logical files

- Can be **used** in a **SQL Statement**, but **treated** like a **SQL View** (**Keys** are **not considered**)
  → **should be avoided**

SQL provides 2 different types of logical files

- **View** Can be specified in **SQL Statements**
  Can be accessed **with native I/O** (but is never keyed)
  → only **restricted use** with **native I/O**
- **Index** Is only used by the **Query Optimizer**
  but **cannot** be specified within **SQL Statements**
  Can be accessed **with native I/O** like **any keyed DDS described file**

DDS described logical Files

Based on either DDS described physical Files or SQL Tables

- **Do not include any data** but an description how to access these data

Keyed (logical) Files

- Include an **access path** for accessing the data in a **pre-defined sequence** → **Binary Radix Tree Index**

Field/Column Selections

- **All** or a **sub-set** of the **fields/columns** in the base physical file(s)/table(s) → **View / Index** (Db2 for i enhancement)
- **Additional fields/columns** in conjunction with predefined **keywords** (e.g. SST)

Row Selection

- **Select / Omit** clauses → **View / Sparse Index**

Joined (keyed) logical files

- **Multiple physical files** and/or SQL tables can be joined together → **View**
- **Key Columns** only from **one** of the joined physical files

Format

- **Single** or **Multiple format** logical files can be defined

No Index equivalent in SQL for Keyed Joined Logical Files !
With multi-format LF a conversion of the PF to DDS not possible !

DDS LF to SQL Index Conversion Example

41

DDS to SQL – Example - Convert logical Files into Indexes

DDS described logical files are interpreted as Views

- **Views:** **unkeyed** logical files
- **Logical files:** traditionally used with **native I/O** for **keyed data access**
  Most logical files include **solely key information**
  Join information and/or SELECT/OMIT clauses are less often used
- **SQL indexes**: **Key Information**
  Can be **used with native I/O** like any keyed logical file

Conversion logical files into SQL indexes better solution

- ACS Option: **Generate index instead of view** (for keyed LF)
- `GENERATE_SQL`: `Index_Instead_Of_View_Option` Parameter

42

Convert Logical Files into SQL Indexes

- Convert the logical Files for the Address Master (ZZADDRP) into SQL Indexes

```
DROP INDEX HSCOMMON10.ZZADDRL0 ;

CREATE UNIQUE INDEX HSCOMMON10.ZZADDRL0
    ON HSCOMMON10.ZZ_ADDRESS_MASTER ( CUSTOMER_NUMBER ASC )
    RCDFMT ZZADDRF ;

LABEL ON INDEX HSCOMMON10.ZZADDRL0
    IS 'Idx Address Master CustNo                    UNIQUE' ;
```

- Delete existing logical file
- Create UNIQUE Index
- Add Index Description

```
DROP INDEX HSCOMMON10.ZZADDRL1 ;

CREATE INDEX HSCOMMON10.ZZADDRL1
    ON HSCOMMON10.ZZ_ADDRESS_MASTER ( COUNTRY ASC ,        CITY ASC ,           ZIPCODE ASC ,
                                      CUSTOMER_NAME1 ASC , CUSTOMER_NAME2 ASC , CUSTOMER_CONTACT ASC )
    RCDFMT ZZADDRF ;

LABEL ON INDEX HSCOMMON10.ZZADDRL1
    IS 'Idx Address Master Country/City/ZipCod/Name1/Name2' ;
```

- Delete existing logical file
- Create Non UNIQUE Index
- Add Index Description

43

44

View – Non-Keyed logical File

Why and When to use?

- Colums have to be **renamed**, build **new columns**
- **Data conversion** or derivations are required
  → e.g. numeric date fields
- Control access to **sensitive data**
  - View without the sensitive data columns
  - Add special preferences to the view
- **Move business logic into database**
  - A view can be used in SQL statements whereever a table can be used
  - **Mask complexity** from users/programmers
  - **Reduces source code** independent which programming language is used → **Reusability**
- **Isolate** programs and procedures from **changes to the physical table**

View – Non-Keyed logical File

Everything that is allowed in a SELECT statement can be used in a view with the exception of ORDER BY

- **Column selection** and generating new columns
- All types of **JOIN** expressions (Inner Join, Left/Right/Full Outer Join, Exception Join, Cross Join)
- **WHERE** conditions
- **GROUP BY** (including multidimensional grouping) and **HAVING** clauses
- **Scalar Functions** / User Defined Functions / User Defined Table Functions
- **OLAP** Specifications
- **CASE** Expressions
- **UNION** / EXCEPT / INTERSECT
- **Common Table Expressions** (CTE) / nested **Sub-Selects**
- **Recursive** CTEs and **Hierarchical** Query Clauses
- Use of **Global Variables**

A view can be built over an other view → Nested Views

- Break complex queries into multiple logical steps

Non-keyed → Access Path Maintenance *REBLD

- Thousands of views can exists **without any performance decrease**

View – Unkeyed logical File - Examples

```
Create View HSCOMMON10.SalesQuart as
Select   CustNo,
         Year(SalesDate) as SalesYear,
         Cast(sum(case when Quarter(SalesDate)= 1 then Amount else 0 end) as Dec(11, 2)) as Q1,
         Cast(sum(case when Quarter(SalesDate)= 2 then Amount else 0 end) as Dec(11, 2)) as Q2,
         Cast(sum(case when Quarter(SalesDate)= 3 then Amount else 0 end) as Dec(11, 2)) as Q3,
         Cast(sum(case when Quarter(SalesDate)= 4 then Amount else 0 end) as Dec(11, 2)) as Q4,
         Cast(sum(Amount) as Dec(13, 2)) as Total
   from  Sales
   group by CustNo, Year(SalesDate);;
```

View: Quarterly Sales

```
Create View HSCOMMON10.SalesQCust
    as Select a.CustNo as ACustNo, SalesYear, Q1, Q2, Q3, Q4, Total,
              b.*
         from SalesQuart a Left outer join AddressX b on a.CustNo = b.CustNo;;
```

View: Joining the quarterly Sales View with the Address Master

```
Select SalesYear, ACustNo, CustName1, City, Q1, Q2, Q3, Q4, Total
from SalesQCust
Where SalesYear = 2009 and ACustNo in ('10003', '10005')
```

Accessing the SalesQCust View

SALESYEAR	ACUSTNO	CUSTNAME1	CITY	Q1	Q2	Q3	Q4	TOTAL
2009	10003	... Goldbach GmbH	... Alzenau...	733,00	1057,91	2227,45	571,50	4589,86
2009	10005	... Alzenauer Dönertreff	... Alzenau...	1475,00	285,75	1587,50	393,70	3741,95

View – Unkeyed logical File - Examples

```
Create Or Replace View ComSQLQry.EXCRATEV1 as
 Select  a.*, Cast(Round(1,00 / ExcRate, 4) as Dec(7, 4)) RevRate
 From XMLTable(
        XMLNamespaces(DEFAULT 'http://www.ecb.int/vocabulary/2002-08-01/eurofxref',
                              'http://www.gesmes.org/xml/2002-08-01' AS "gesmes"),
        'gesmes:Envelope/Cube/Cube/Cube'
        Passing XMLParse(DOCUMENT
                  HTTP_GET('https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml', '')
               )
        Columns Subject   VarChar(30) Path '../../../gesmes:subject',
                Sender    VarChar(30) Path '../../../gesmes:Sender/gesmes:name',
                ExcDate   Date        Path '../@time',
                Currency  Char(3)     Path '@currency',
                ExcRate   Dec(10, 4)  Path '@rate'      ) a;
```

- Accessing a Webservice of the European Central Banc: Exchange Rates

```
Select * from EXCRATEV1;
```

SUBJECT	SENDER	EXCDATE	CURRENCY	EXCRATE	REVRATE
Reference rates	European Central Bank	2022-03-25	USD	1,1002	0,9089
Reference rates	European Central Bank	2022-03-25	JPY	134,0700	0,0075
Reference rates	European Central Bank	2022-03-25	BGN	1,9558	0,5113
Reference rates	European Central Bank	2022-03-25	CZK	24,6450	0,0406

- Display the current Exchange Rates by accessing the EXCRATEV1 View

```
Create Or Replace View COMSQLQRY.EXCRATEV2 as
Select s.*, Cast(Round(Amount * ExcRate, 4) as Dec(11, 4)) Amount_USD
   from Sales s cross join (Select ExcRate from ExcRateV1 Where Currency = 'USD') ;
```

- Determine the current sales in a foreign (US-Dollar) currency

View – Modernization - What to do and how to start

Create a basic view containing all fields/columns in the same sequence as in the based physical file/table

- **No longer access** the **based physical file/table directly**
  → Using **Instead Of Triggers** will allow a future redesign of your database

Create additional views based on this basic view

- Joins with other (basic) views: e.g. Join OrderHeader, Addresses, OrderPositions, ItemMaster

Create a view for each problem to solve

- Move business logic into database: e.g. Outstanding Orders, accumulated stocks / item no
- If circumstances change, only one or several views must be updated

Force your programmers and query users to only access these views

- Use them with embedded SQL, ODBC/JDBC, Query/400, Db2 WebQuery
- When accessing those views, do not use SELECT *
  - If a view is changed, recompilation is only necessary if new columns are added and used
  - Listing Columns may result in performance gains (Less logical DB access, IOA access)

49

Unique Identifier

50

Advantages of Unique Identifires/Surrogate Keys over Natural Keys

Stability
- **Surrogate keys remain unchanged** over time
- **Natural keys** may need to be **modified** or **extended**
  → **extensive changes** in dependent tables, views, programs/procedures …

Uniqueness
- **A primary key** or **unique constraint** on a surrogate key **guarantees unique identification** even if the (unique) natural key changes.

Simplicity
- **Foreign key relationships** are **easier to manage** and **reduce the risk of incorrect table joins** based on partial natural keys.

Performance
- Integer or numeric **surrogate keys** generally **provide faster processing** and more efficient indexing than complex composite natural keys.

Unique Identifier – Automatic Generation

Identity Column Attribute (for numeric columns) or ROWID
- **Identity Column** Not guaranteed to be unique
  primary key or **unique** index/constraint **must be defined**
- **ROWID** Unique without key constraint or unique index
  but **not in ascending** or **descending order**
- Will be generated as soon as a **row is written**
  independent which interface (**SQL or native I/O**) is used
- **Can only be defined within SQL defined Tables**

Unique Identifier – On Demand Generation

Sequence Object

- Generated with: **CREATE OR REPLACE SEQUENCE**
- Get value with: **NEXT VALUE FOR SequenceName**

GENERATE_UNIQUE() Scalar function

- Returns a **unique** 13 byte bit character string **(CHAR(13) for BIT DATA)**
- includes the **UTC** (universal time coordinated) and the **system serial number**

```
Values(Generate_Unique(), Hex(Generate_Unique()), Timestamp(Generate_Unique()));
```

00001	00002	00003
□bi□°Ñ□²□p-□	0782561105AF6934EA0399B001	2026-06-06 12:01:33.170104

GENERATE_UUID() Scalar Function

- Formatted string of a **Universally Unique Identifier (UUID)** based on the **version 4 Algorithm**
- result is a **CHAR(36)** value in the format **xxxxxxxx-xxxx-4xxx-xxxx-xxxxxxxxxxxx**

GENERATE_UUID_BINARY() Scalar Function

- returns the **UUID value** to as **BINARY(16)**

```
Values(Generate_UUID(), Generate_UUID_Binary());
```

00001	00002
AABE0EA6-C36D-449F-BB53-F18395B07B4C	0A1781CFD6B242C0965463AEFE667D76

How to retrieve the last inserted value in an Identity Column?

IDENTITY_VAL_LOCAL() Scalar Function

- returns the **most recently assigned** value for **an identity column**
  - Function will be called without parameter
    → Last insert for the current unit of work in a table with identity column
- **Last identity value** on the **same level**
  - In the meantime inserted rows by other jobs are ignored
  - An activated trigger within the same job gets its own level
- Works also **after** a **native I/O WRITE**!

Data Change References - Composition Insert/Select

Insert with Sub-Select as Final Table

- Insert must be specified within the **FROM Clause**
- **Insert** and **subsequent Select** within the **same SQL Statement**:

  Allows to determine all inserted rows with all automatically generated values (for example Identity Columns or Change Date)
- Can be used within **Embedded SQL or SQL/PL** → Declare Cursor

ORDER BY INPUT SEQUENCE

- Return the rows in the result table in the **same sequence as inserted**

55

Generated Columns for Auditing Row

56

GENERATED ALWAYS Columns - Row Level Auditing

Generated Always Columns

- Set by the Database Manager with any Insert/Update Operation

Types of Generated Always Columns

- Identity Columns
- Row Change Timestamp Columns
- Temporal Tables: Row Begin / Row End Columns
  Transaction Start Id Column
  Data Change Operation Column
- Generated Expression Columns
  - Special Registers
  - Built-In-Global Variables

GENERATED ALWAYS Columns - Generated Expressions

Generated Expressions are based on

- Special Registers

Special register	Data type for the column
CURRENT_CLIENT_ACCTNG	VARCHAR(255)
CURRENT_CLIENT_APPLNAME	VARCHAR(255)
CURRENT_CLIENT_PROGRAMID	VARCHAR(255)
CURRENT_CLIENT_USERID	VARCHAR(255)
CURRENT_CLIENT_WRKSTNNAME	VARCHAR(255)
CURRENT_SERVER	VARCHAR(18)
SESSION_USER	VARCHAR(128)
USER	VARCHAR(18)

- Built-in-Global Variables

Built-in global variable	Data type for the column
QSYS2.JOB_NAME	VARCHAR(28)
QSYS2.SERVER_MODE_JOB_NAME	VARCHAR(28)
SYSIBM.CLIENT_HOST	VARCHAR(255)
SYSIBM.CLIENT_IPADDR	VARCHAR(128)
SYSIBM.CLIENT_PORT	INTEGER
SYSIBM.PACKAGE_NAME	VARCHAR(128)
SYSIBM.PACKAGE_SCHEMA	VARCHAR(128)
SYSIBM.PACKAGE_VERSION	VARCHAR(64)
SYSIBM.ROUTINE_SCHEMA	VARCHAR(128)
SYSIBM.ROUTINE_SPECIFIC_NAME	VARCHAR(128)
SYSIBM.ROUTINE_TYPE	CHAR(1)

Table with Generated Columns

```
Create Or Replace Table HSCOMMON10.MYADDTABLE
  (UNQID        Integer       Generated Always as Identity
                              (Start with 100, Increment by 10,
                               No Order,       No Cycle,
                               No Minvalue,    No MaxValue,
                               Cache 20),
   INTVAL       Integer       Not NULL  Default 0,
   CHGTIMESTMP  Timestamp(6)  Generated Always For Each Row On Update
                                as Row Change Timestamp
                              Not NULL,

   -- Special Registers
   CHGAccTng    VarChar(255)  Generated Always as (Current Client_AccTng),
   ChgAPPName   VarChar(255)  Generated Always as (Current Client_ApplName),
   ChgPGMId     VarChar(255)  Generated Always as (Current Client_ProgramId),
   ChgUserId    VarChar(255)  Generated Always as (Current Client_UserId),
   ChgWrkStn    VarChar(255)  Generated Always as (Current Client_WrkStnName),
   ChgCurrSvr   VarChar(18)   Generated Always as (Current Server),
   ChgSessUsr   VarChar(128)  Generated Always as (Session_User),
   ChgUser      VarChar(18)   Generated Always as (User),

   -- System Global Variables
   ChgJob       VarChar(28)   Generated Always as (QSYS2.Job_Name),
   ChgSvrMode   VarChar(28)   Generated Always as (QSYS2.Server_Mode_Job_Name),
   ChgHost      VarChar(255)  Generated Always as (SYSIBM.Client_Host),
   ChgIPAddr    VarChar(128)  Generated Always as (SYSIBM.Client_IPAddr),
   ChgPort      Integer       Generated Always as (SYSIBM.Client_Port),
   ChgPckName   VarChar(128)  Generated Always as (SYSIBM.Package_Name),
   ChgPckSch    VarChar(128)  Generated Always as (SYSIBM.Package_Schema),
   ChgPckVers   VarChar(64)   Generated Always as (SYSIBM.Package_Version),
   ChgRoutSch   VarChar(128)  Generated Always as (SYSIBM.Routine_Schema),
   ChgRoutSpc   VarChar(128)  Generated Always as (SYSIBM.Routine_Specific_Name),
   ChgRoutTyp   Char(1)       Generated Always as (SYSIBM.Routine_Type)
  );
```

- Identity Column
- Generated Expression Colums
- Row Change Timestamp
- Special Registers
- System Global Variables

Constraints and Referential Integrities

Data Integrity

What is data integrity?

- Data integrity is the principle of **ensuring data values** between tables are **kept in a state** that makes sense to the business.

Methods to accomplish data integrity:

- **Constraints** are **rules enforced** by the **database manager**
  - **Check Constraints** Check values within a column
  - **Key Constraints** Prevent from duplicate rows/records
  - **Referential Integrities** Defines dependencies between tables - Primary and Foreign Key Constraints
- **Trigger** are **programs activated** by the **database manager**
  - depending on the **trigger time** (before / after)
    the **trigger event** (insert / update / delete / instead of)

Advantages

- The definition of **business roules** can be **centralized**
- **Reduced Source Code** Business roules are moved into the database
- SQE optimizer: **Constraint awarness**.

Methods to accomplish Data Integrity - Triggers

Forcing Business Rules

- Rules that cannot be handled by check constraints
- Examples: After having delivered all positions,
  copy Order Header and Order Positions into a History Table
  Date Trigger to keep the date and numeric date data consistent

Checking Data Integrities

- Example: User enters an order → check if he is responsible for the customer

Checking Data Consistence over multiple Tables

- Example: Delivery Date is stored in multiple Tables
  when changing the delivery date in any table, it is also modified in all other tables

Integrating new Technologies

- Example: After an Order is entered, an eMail/SMS will be sent to the customer

Commitment Control

63

Commitment Control

Condensing multiple operation to a single Transaction

- A **Transaction** includes **multiple individual modifications** on objects, which are handled as **a single Action**
- **Examples** for Transactions:
  - Accounting: Credit / Debit
  - Order: Part Delivery not allowed

Commitment Control ensures

- **ALL changes** within a transaction are executed
- **NONE of the changes** within a transaction is executed
- When cancelling a transaction all changes within the transaction are reset

Begin/End of a Transaction: `COMMIT`
Reset a Transaction: `ROLLBACK`

64

Row And Column Access Control

65

What is RCAC?

RCAC = Row and Column Access Control

- **Additional layer** of **Data Security** (available with Db2)
  - Directly linked with the database tables
- **Complementary** to **Object Level Security**
- **Limits access** to **only** the **required data**
  - Controls access to a table at the **row** and/or **column level**
  - ***ALLOBJ users** can **no longer freely access all** of the **data** in the database

Provides 2 different approaches

- Access permissions for **rows** → `CREATE PERMISSION`
- Provides masks for **column contents** → `CREATE MASK`

IBM Advanced Data Security feature for i

- **Must be installed** → No-charge feature, option 47
- Required on **both development** and **production systems**

66

Externalizing Data Access

67

Externalizing Data Access –Functions for Insert / Update / Delete

Standard Procedures in Service Programs for Insert/Update/Delete and Single Row Access

- Parameter for Insert/Update: Data Structure **Base View**
- Return Value for Insert: **Unique Key** (e.g. Identity Colum or compound key) or Relative Record No
- Parameter for Delete/Single Row: **Unique Key** and/or **Relative Record No**
- Return Value for Single Row: **Indicator** i.e. Found or Not
- Output Parameter for Single Row: Initialized Data Structure **Base View**
- Additional Parameters: Handling Record Locks, Execution with/without Commitment Control, Send Escape Message if an Error occurs, Update Always

→ **Base Source Code can be automatically generated based on Templates**

Additional Procedures (must be manually completed)

- Delete: Check Procedure, whether a record/row can be deleted or not
- Insert/Update: Check Procedure for Input Values / Initialization with Default Values

These Standard Functions must be used for ALL Insert/Update/Delete Operations

68

CRUD Functions → Web-/Micro Services

Executing (RPG) CRUD-Funktionen?

- Calling from RPG (or ILE Language): **Direct** Call
- Other calls: **Web-/Micro-Service**

Web-/Micro-Services

- Embedding **CRUD functions** in **Web-/Micro-Services**
  - **GET** (Read) / **PUT** (Write) / **POST** (Update) / **DELETE** (Delete) Methods
- **Wapper Program** for Web/Micro-Services
  - **Receiving** and **Decomposing** the passed **JSON/XML Data**
  - **Populating the Data Structures** for calling/passing to the CRUD Functions
  - **Executing** the appropriate CRUD-Function
  - If an **error occurs** → **(re)send** the error message

CRUD Functions and Data Consitency?

CRUD Functions are not directly linked with the database (table)

- Can be **circumvented**!!!

Solution: Before Insert/Update/Delete Trigger

- Checking the **Call Stack** (IBM Service: Table Function **STACK_INFO**)
  - If the **CRUD**-Function is **found** → **INSERT/UPDATE/DELETE** Request **is performed**
  - If the **CRUD**-Function is **not found** → **Error** (request is not performed)

Any Questions?

Special Thanks to

Holger Scherer – RZKH Rechenzentrum Kreuznach

- For providing an IBM i-System enabling the creation of the samples/code used in my presentations
- http://www.rzkh.de

RZKH RECHENZENTRUM KREUZNACH

Your data is save! … in the bunker

Speaker‘s Biography

Birgitta Hauser
Diplom-Betriebswirt (BA)
Database and Software Architect

Birgitta Hauser worked on the IBM i and its predecessors since 1992. She graduated with a business economics diploma, and started programming on the AS/400 in 1992. She worked and works as traditional RPG Programmer but also as Database and Software Engineer, focusing on IBM i application and database modernization.

Currently she is self-employed and works in Consulting and Application and Database Modernization on IBM i and Db2 for i. Since July, 2019 she is occasionally working for Fresche Solutions Inc. (Montréal) as a contractor.

She also works in education as a trainer for RPG and SQL developers.

Since 2002 she has frequently spoken at the COMMON User Groups and other IBM i and Power Conferences in Germany, other European Countries, USA and Canada.

In addition, she is co-author of two IBM Redbooks and also the author of several articles and papers focusing on RPG and SQL for the ITP Verlag (a German publisher), IT Jungle Guru and IBM DeveloperWorks.

In 2015 she received the John Earl Speaker Scholarship Award. In 2018 she received the Al Barsa Memorial Scholarship Award.

IBM Champion since 2020

74

75