A Shallow Dive Into Database Modernization

Patrick Behr

Agenda

- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority

DISCLAIMER!

This is a SHALLOW dive.

This will be a thorough, yet simple example.
There are many important nuances that will not be discussed.
The specific details of your environment will require your vigilance and lots ‘o testing.

There are many regulations that you need to understand (HIPPA, SOX, PCI).

There's lots of help out there

- Redbooks
- White Papers
- Infocenter
- DeveloperWorks
- Forums
- Google
- USER GROUPS!

Be sure to R.T.F.M.

Read The Free Manual

A journey of 1000 miles begins with

A green screen

A DDS file

```
R ORDFILE
  CUSTOMER          5A
  ITEMNUM          10A
  DATEORD           L
  QUANTITY         10P
  ORDERBY          10A
  TOTALAMT         10P 2
  ORDRNUM           6P
K ORDRNUM
```

A journey of 1000 miles begins with

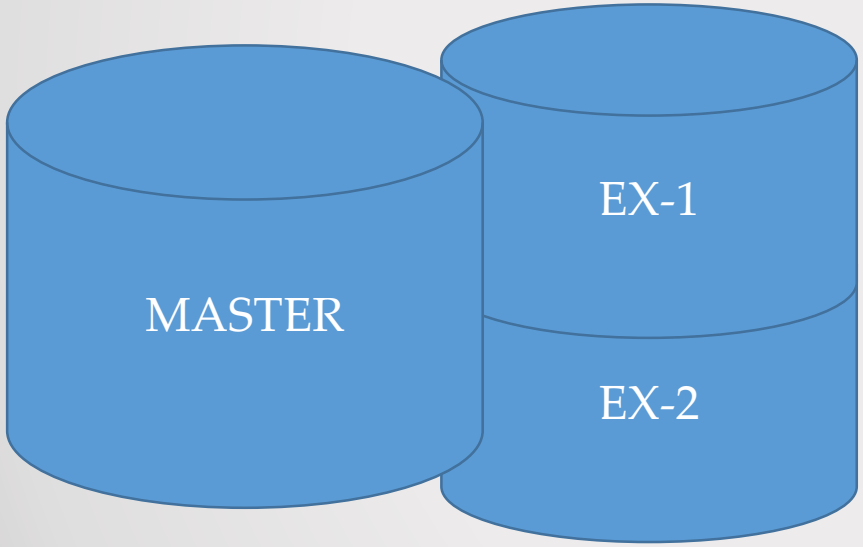
Can't we just add a field to that file…

We need to mask that sensitive data…

We need encryption…

How we've done it in the past

Create an extension file (or two)

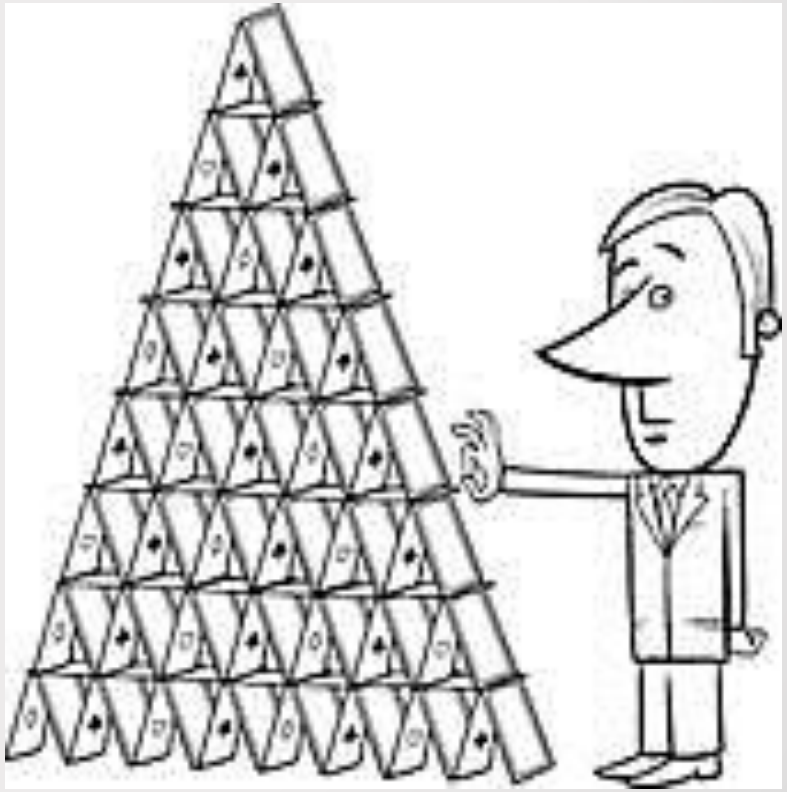


Recompile all your programs

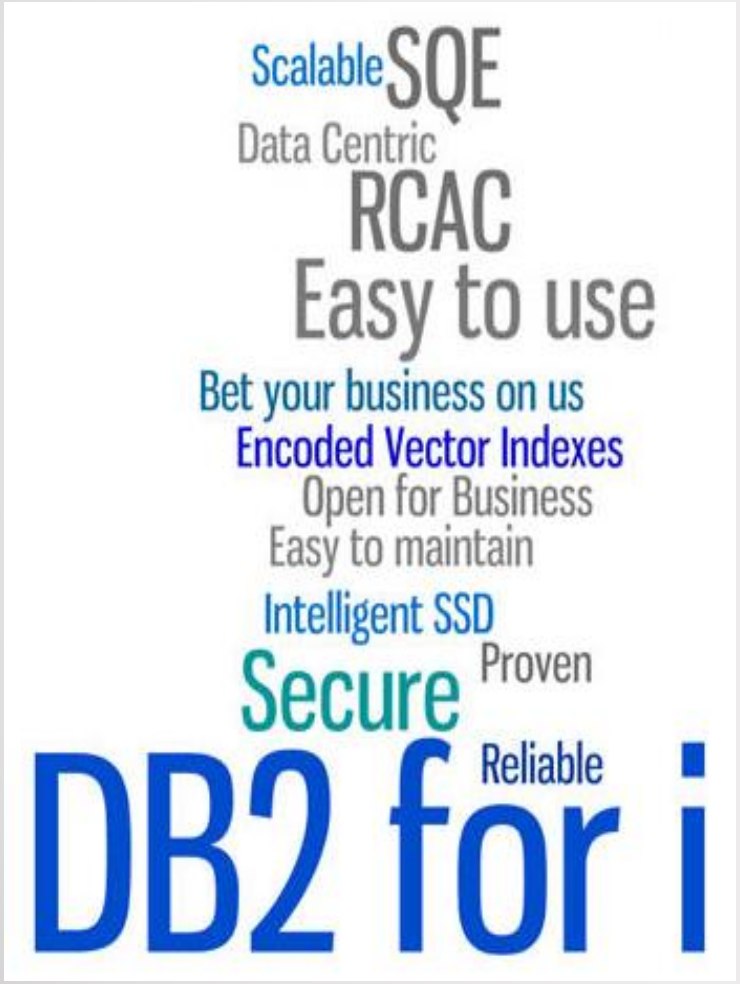
Why we shouldn’t do it that way

It's a LOT of work.

Quality of the system.



There's a better way!

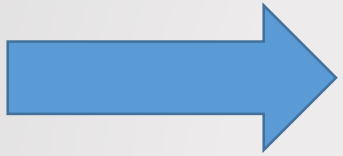


A better system.

DDS to SQL Conversion - Why

- It's the strategic choice of IBM
- Data-Centric programming
- Performance/Scalability
- Agility/Flexibility

Agenda

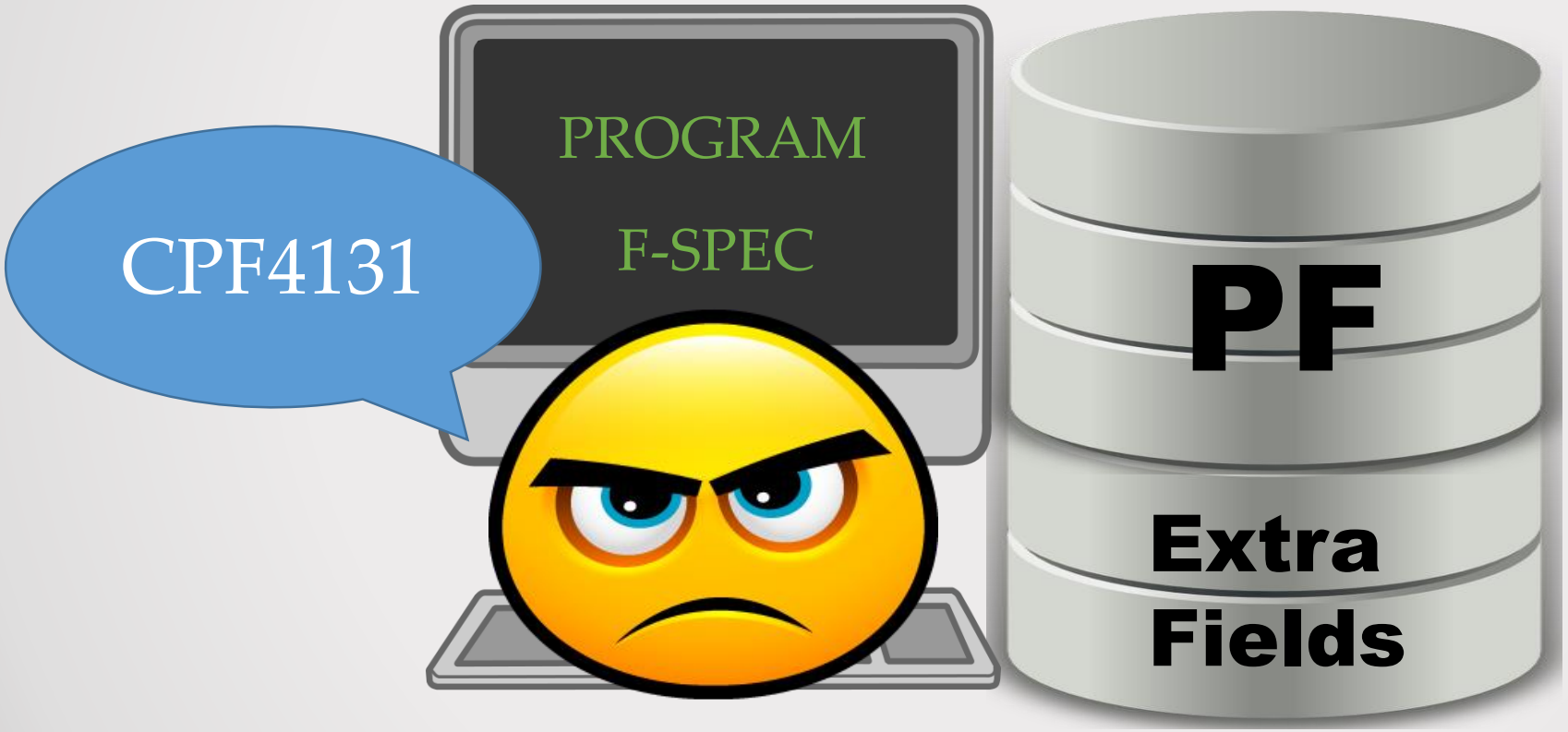


- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority

DDS to SQL Conversion - Steps

- Create a new SQL Table
- Create a logical file
-
-
-
-

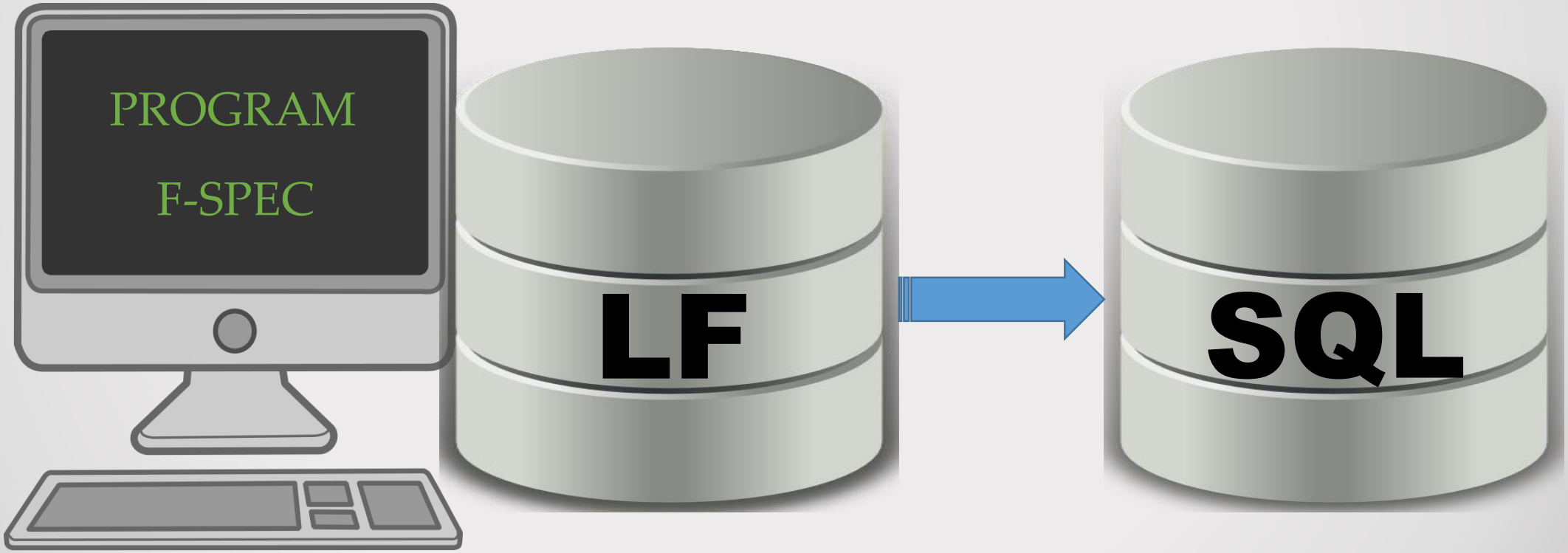
Level Checks



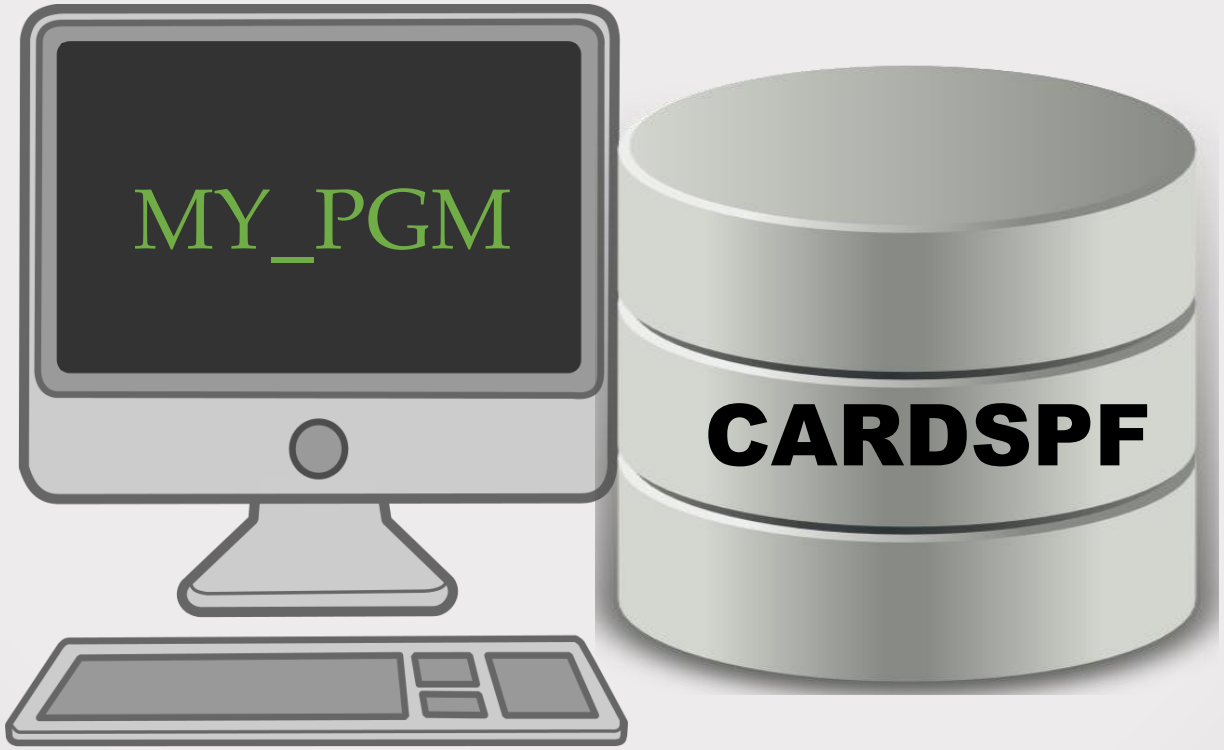
Record Format Level Identifier must match!

DDS to SQL Conversion – Step 1

DDS to SQL Conversion – Step 2



DDS to SQL Conversion – In action



DDS to SQL Conversion – In action

DSPPGMREF PGM(PBEHR/MY_PGM):

```
Object  . . . . . . . . . . . . . . . . . . . . : CARDSPF
  Library . . . . . . . . . . . . . . . . . . . :   PBEHR
  Object type . . . . . . . . . . . . . . . . . :   *FILE
  File name in program  . . . . . . . . . . . . :   CARDSPF
  File usage  . . . . . . . . . . . . . . . . . :   Input
                                                    Output

  Number of record formats  . . . . . . . . . . :        1
   Record Format   Format Level Identifier   Field Count
     CARDR              2A60B36AE6FDD                3
```

DDS to SQL Conversion – In action

DSPFD FILE(PBEHR/CARDSPF):

```
Record Format List
                                  Record  Format Level
 Format          Fields           Length  Identifier
 CARDR                 3               54  2A60B36AE6FDD
```

2A60B36AE6FDD is the "magic" number!

DDS to SQL Conversion – Step 1

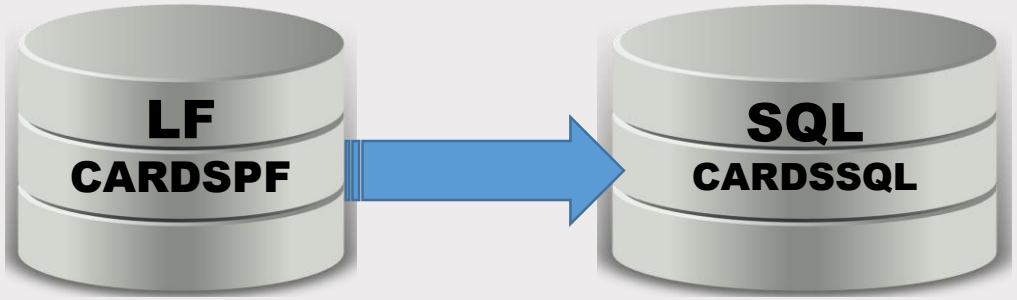

```
R CARDR
  CARDID          9S 0
  CARDHOLDER      25A
  CARDNBR         20A
K CARDID
```

```
Create Or Replace Table Credit_Cards
For System Name CARDSSQL
(
   CARD_ID         For Column CARDID      Numeric(9,0)
  ,CARD_HOLDER     For Column CARDHOLDER  Char(25)
  ,CARD_NUMBER     For Column CARDNBR     Char(20)
  ,CREATE_TS       For Column CARDCRTTS   Timestamp
     Not Null With Default CURRENT_TIMESTAMP
  ,CREATE_USER     For Column CARDCRTUSR  Timestamp
     Not Null With Default CURRENT_TIMESTAMP
  ,CHANGE_TS       For Column CARDCHGTS   Timestamp
     Not Null For Each Row On Update
               As Row Change Timestamp
  ,PRIMARY KEY( CARD_ID )
)
RCDFMT CARDR;
```

DDS to SQL Conversion – Step 2


```
                                        UNIQUE
R CARDR                                 PFILE(CARDSSQL)
  CARDID                 9S 0           TEXT('Card ID')
  CARDHOLDER            25A             TEXT('Card Holder Name')
  CARDNBR               20A             TEXT('Card Number')
K CARDID
```

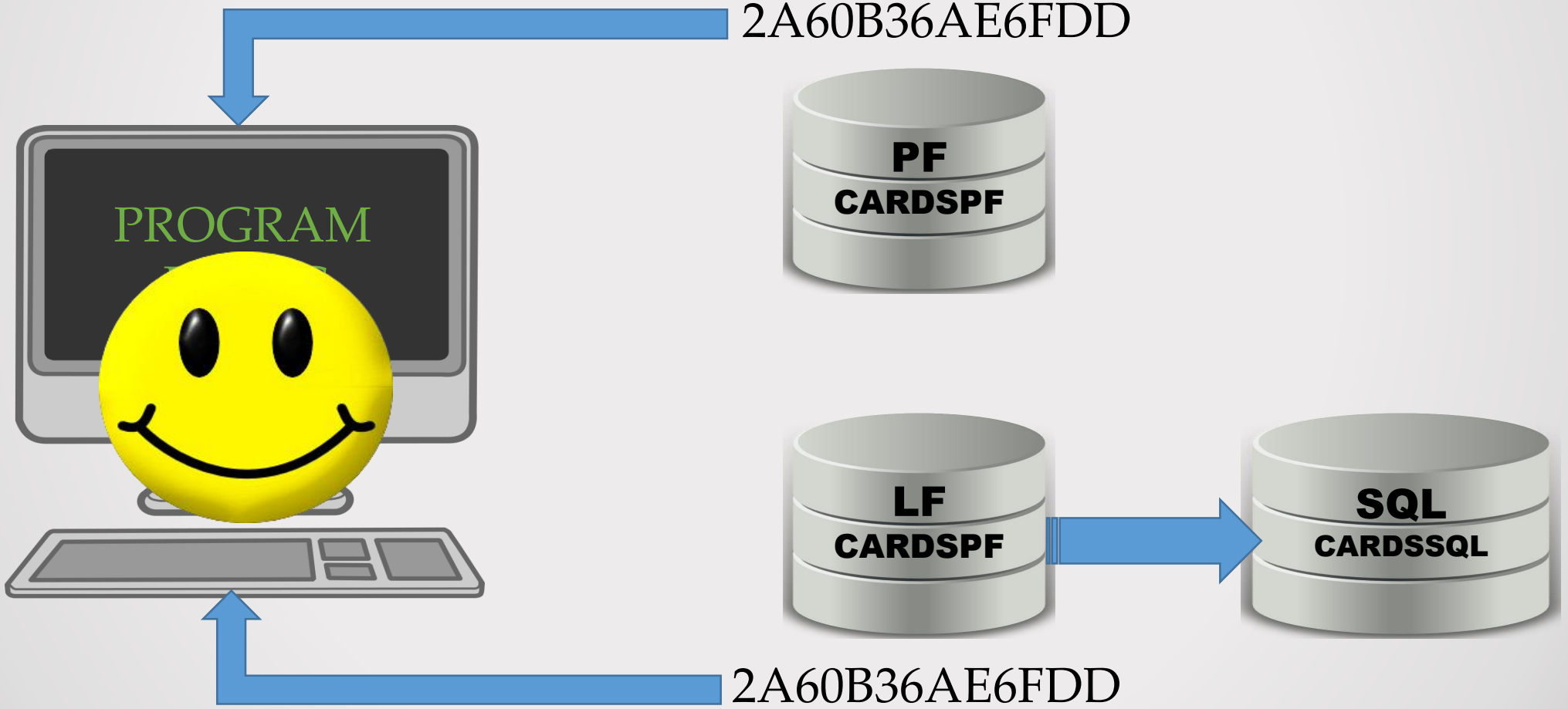
DDS to SQL Conversion – Step 2



DSPFD FILE(PBEHR/CARDSPF):

```
Based on file . . . . . . . . . . . . . . . . : CARDSSQL
  Library . . . . . . . . . . . . . . . . . . : PBEHR
  Member  . . . . . . . . . . . . . . . . . . : CARDSSQL
  Logical file format . . . . . . . . . . . . : CARDR
Record Format List
                          Record  Format Level
 Format         Fields    Length  Identifier
 CARDR               3        54  2A60B36AE6FDD
```

DDS to SQL Conversion



DDS to SQL Conversion – Step 2B


```
R CARDR                          PFILE(CARDSSQL)
                                 FORMAT(CARDSPF)
K CARDHOLDER
```

Modify any logical files referencing CARDSPF

Set PFILE to new SQL table, and share the record format of the new CARDSPF logical.

DDS to SQL Conversion in 3 minutes…

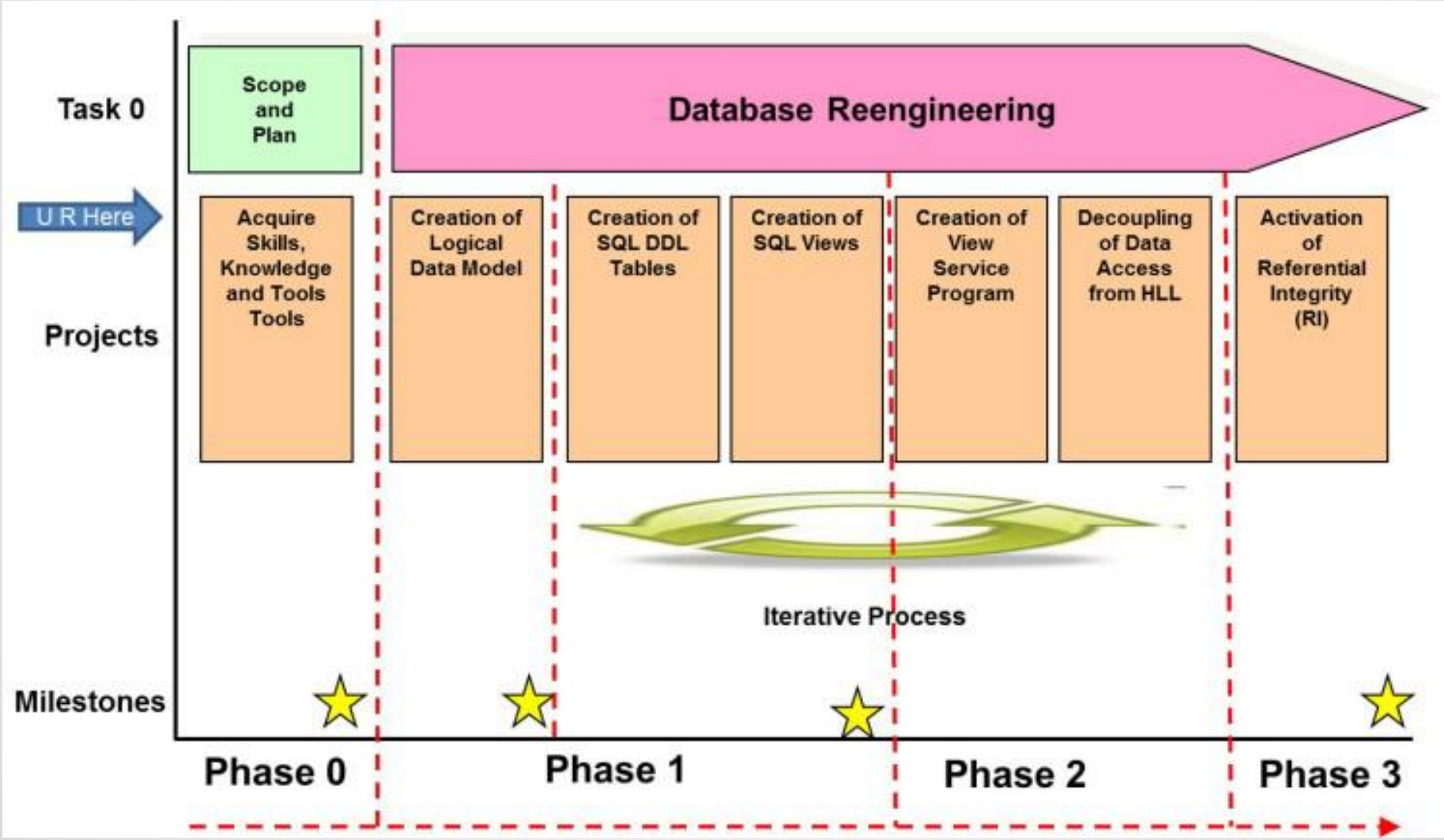
dds_conversion video

A journey of 1000 miles …

Can't we just add a field to that file…

You bet!

DDS to SQL Conversion - Roadmap



DDS to SQL Conversion - RTFM

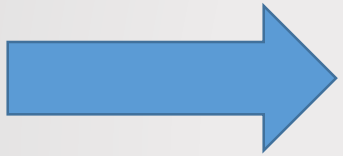
Be sure to R.T.F.M.

Modernizing Database Access; The Madness Behind the Methods
By Dan Cruikshank

Modernizing IBM eServer iSeries Application Data Access
IBM Redbook

Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between
IBM Redbook

Agenda



- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority

Field Masking with RCAC

	What you have	What you want
Credit Card:	1234567891234	**********1234
Birthday:	10 / 06 / 1981	10 / 06 / ####
SSN:	123-45-6789	XXX-XX-6789

Field Masking with RCAC - Steps

- Register with QIBM_DB_SECADM function
- Create a mask function
- Activate the mask function
-
-
-

Register with QIBM_DB_SECADM

Works even if user has *ALLOBJ

Separation of Duties:
- Authority to access data *ALLOBJ
- Authority to RCAC

Register with QIBM_DB_SECADM

Only users with QIBM_DB_SECADM function can manage RCAC rules.

```
CHGFCNUSG  FCNID(QIBM_DB_SECADM)
           USER(QSECOFR)
           USAGE(*ALLOWED)
```

Create a mask function

Create Mask *mask_name*
On *FILE*
For Column *FIELD*
Return
Case When *some_condition* Then *FIELD*
        Else *masked_value*
End
Enable;

Create a mask function

```
Create Mask Credit_Card_Number_Mask
On Credit_Cards
For Column CARD_NUMBER
Return  Case  When
Verify_Group_For_User(Current_User, 'ACCOUNTING') = 1
        Then CARD_NUMBER
        Else  '*********' || Right(Trim(CARD_NUMBER), 4)
End
Enable;
```

Activate the mask function

```
Alter Table Credit_Cards
Activate Column Access Control;
```

```
Alter Table Credit_Cards
Deactivate Column Access Control;
```

```
Drop Mask Credit_Card_Number_Mask;
```

Field Masking With RCAC

Field masking with RCAC in 2 minutes…

Field masking video

Field Masking With RCAC

- Requires 7.2 and IBM Advanced Data Security for i (5770SSI option 47)
- RCAC will affect CPYF, CRTDUPOBJ, etc.
  Make sure that your HA/Backup solution will work.
  (RCAC is not applied to the journal receiver process).
- Triggers have access to data outside of RCAC,
  So they must be defined as “SECURE”
- Masking is applied to the final result set.
  Selection, grouping, ordering based on unmasked values
- BE CAREFUL WITH UPDATES!!!

Field Masking With RCAC

Be sure to R.T.F.M.

Row and Column Access Control Support in IBM DB2 for i
IBM Redbook

RCAC in DB2 For i, Part 2: Column Masks
by Michael Sansoterra, IT Jungle

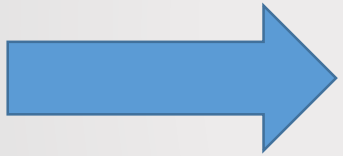
A journey of 1000 miles …

We need to mask that sensitive data…

No problem!

Agenda

- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority



Encryption with FIELDPROC

	What you have	What you want
Credit Card:	1234567891234	0xde015724b081ea7003d
Birthday:	10 / 06 / 1981	0xfd8b695b39e0
SSN:	123-45-6789	0x96a45cbcf9ca9425cd

Encryption with FIELDPROC

- Data, index, and journals stored on hard disks or tapes are transformed. No one can get the plain text data without the FieldProc program.
- No change to the original table definition is needed (read as: “no recompiles”).

Encryption with FIELDPROC - Steps

- Write the field procedure (program)
- Associate the FIELDPROC with the column
-
-
-
-

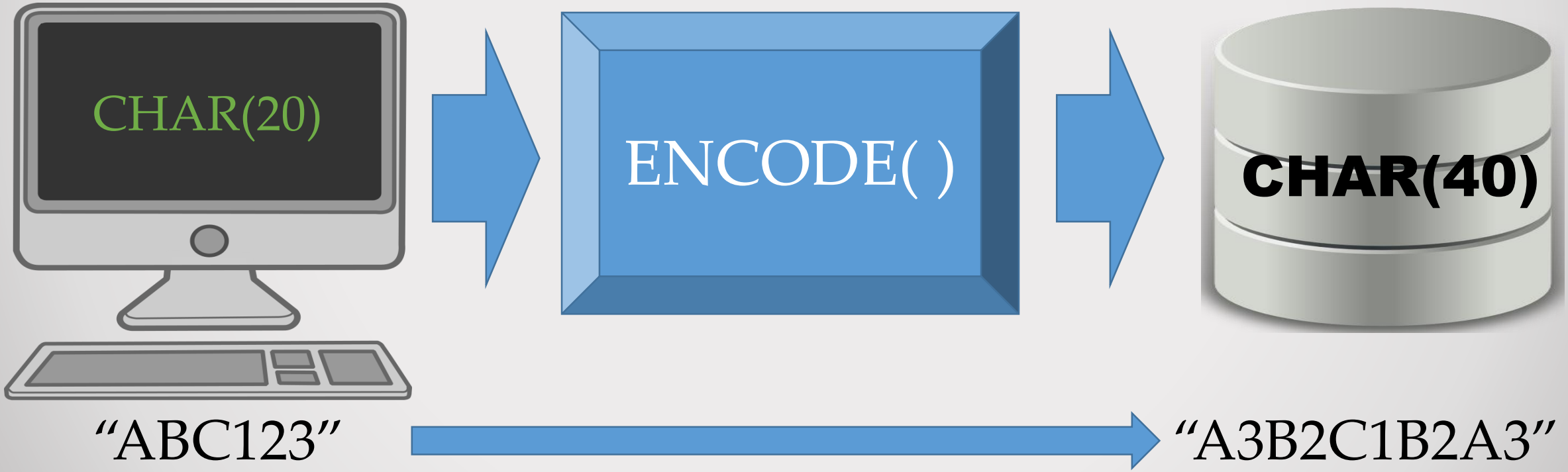
Encryption with FIELDPROC

- A field procedure is an exit program designed to transform values in a single column.
- You are responsible for writing the procedure.
- DB2 will call your field procedure whenever data is written/retrieved from the database.

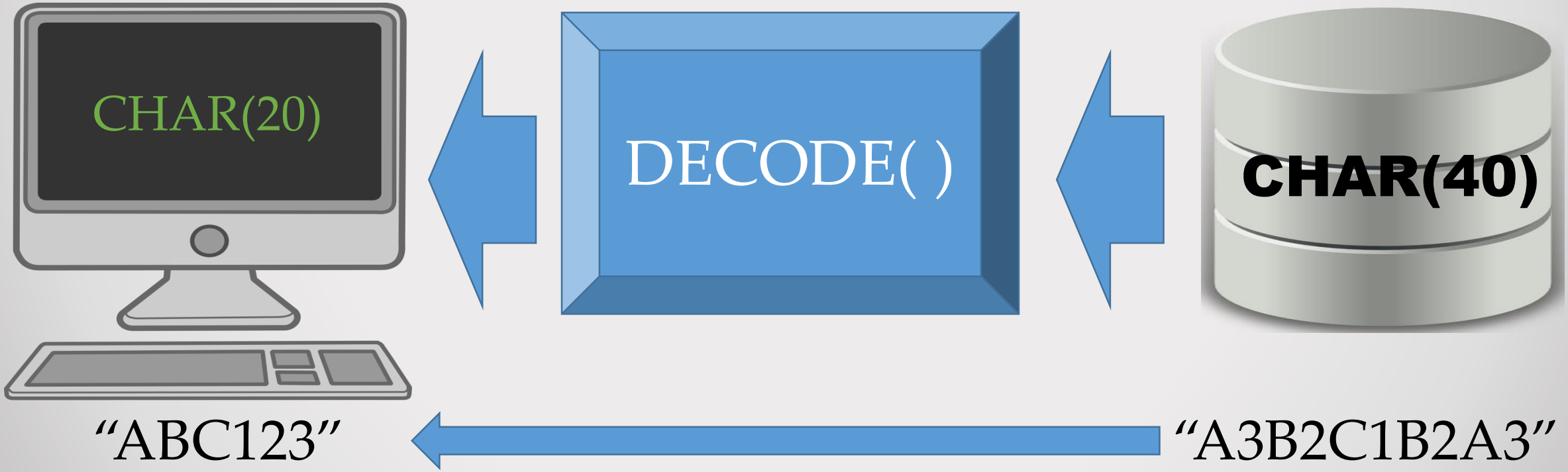
Encryption with FIELDPROC

- A field procedure has three functions:
  - Define how the encoded data will be stored on disk
  - Provide a procedure to encode data
  - Provide a procedure to decode data

Encryption with FIELDPROC



Encryption with FIELDPROC



Field Procedure parms

Parms:

functionCode	int(5)
optParms	likeds(SQLFOPVD)
decodeType	likeds(SQLFPD)
decodeData	char(20)
encodeType	likeds(SQLFPD)
encodeData	char(40)
sqlState	char(5)
sqlMsgText	likeds(SQLFMT)

Field Procedure parms

Parms:

- **functionCode**
- optParms
- decodeType
- decodeData
- encodeType
- encodeData
- sqlState
- sqlMsgText

when functionCode = 8;
Define the encoded field

when functionCode = 0;
Encode the data

when functionCode = 4;
Decode the data

FIELDPROC – Define encoded field

Parms:

functionCode → functionCode = 8

optParms

decodeType → DB2 sends in decoded data type

decodeData

encodeType ← You send back encoded data type

encodeData

sqlState

sqlMsgText

SQL field definition data structure

SQLFPD (Field Data Type) Data Structure:

SQLFST	= SQL Data Type
SQLFBL	= Length in bytes
SQLFL	= Length in characters
SQLFP	= Field precision
SQLFS	= Scale
SQLFC	= CCSID
SQLFAL	= Allocated Length

FIELDPROC – Define encoded field

Parms:

functionCode
optParms
decodeType
decodeData
encodeType
encodeData
sqlState
sqlMsgText

```
// Make encoded type same as decoded...
encodeType = decodeType;

// Change the length to 40 characters
encodeType.SQLFL = 40;     // length
encodeType.SQLFBL = 40;  //bytes
```

FIELDPROC – Encode data

Parms:

functionCode → functionCode = 0

optParms

decodeType

decodeData → DB2 sends in decoded data

encodeType

encodeData ← You send back encoded data

sqlState ← You send back sqlState

sqlMsgText

FIELDPROC – Encode data

Parms:

functionCode
optParms
decodeType
decodeData
encodeType
encodeData
sqlState
sqlMsgText

```
// Called on write/update
when functionCode = ENCODE;

// logic to encrypt the data goes here...
encodeData = EncodeCard(decodeData);
sqlState = '00000';
```

FIELDPROC – Decode data

Parms:

functionCode → functionCode = 4

optParms

decodeType

decodeData ← You send back decoded data

encodeType

encodeData → DB2 sends in encoded data

sqlState ← You send back sqlState

sqlMsgText

FIELDPROC – Decode data

Parms:

functionCode
optParms
decodeType
decodeData
encodeType
encodeData
sqlState
sqlMsgText

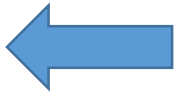
```
// Called on read
when functionCode = DECODE;

// logic to decrypt the data goes here...
decodeData = DecodeCard(encodeData);
sqlState = '00000';
```

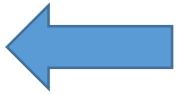
FIELDPROC - Program

```
  Ctl-Opt  debug option(*NODEBUGIO : *SRCSTMT );
  Ctl-Opt  dftactgrp(*no) actgrp(*caller);
  Ctl-Opt  thread(*serialize);

/COPY QSYSINC/QRPGLESRC,SQLFP

  dcl-pr MY_FLDPROC                    extpgm('MY_FLDPROC');
     functionCode                      int(5) const;
     optParms                          likeds(SQLFOPVD);
     decodeType                        likeds(SQLFPD);
     decodeData                        char(20);
     encodeType                        likeds(SQLFPD);
     encodeData                        char(40);
     sqlState                          char(5);
     sqlMsgText                        likeds(SQLFMT);
  end-pr;
```

FIELDPROC - Program

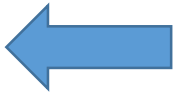
```
when functionCode = INITIALIZE;

    if  decodeType.SQLFST <> 452
    and decodeType.SQLFST <> 453;
        // Return error for unsupported data type
         sqlState = '38001';
         sqlMsgText = 'Invalid data type for fieldproc';
    else;
        // The Encoded value has almost all of the same attributes
        // as the decoded value...just need to change the length.
         encodeType = decodeType;
         encodeType.SQLFL = 40;
         encodeType.SQLFBL = 40;
    endif;
```

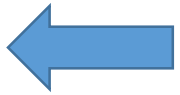




FIELDPROC - Program

```
when functionCode = ENCODE;

     // make sure we don't encode a masked value !!
      if %subst(decodeData:1:6) = '******';
           sqlstate = '09501';
           sqlMsgText = 'Encoding not valid for a masked value';
      else;
          // your logic to encrypt the data goes here...
           encodeData = EncodeCard(decodeData);
           sqlState = '00000';
      endif;
```

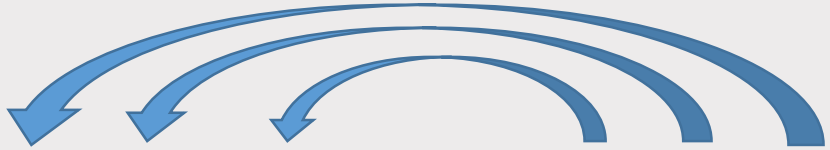




FIELDPROC – Encode logic

// Encode procedure

Take characters from the end of the string and insert them between existing characters



ABCD1234

A4B3C2D11D2C3B4A

FIELDPROC – Associate FIELDPROC

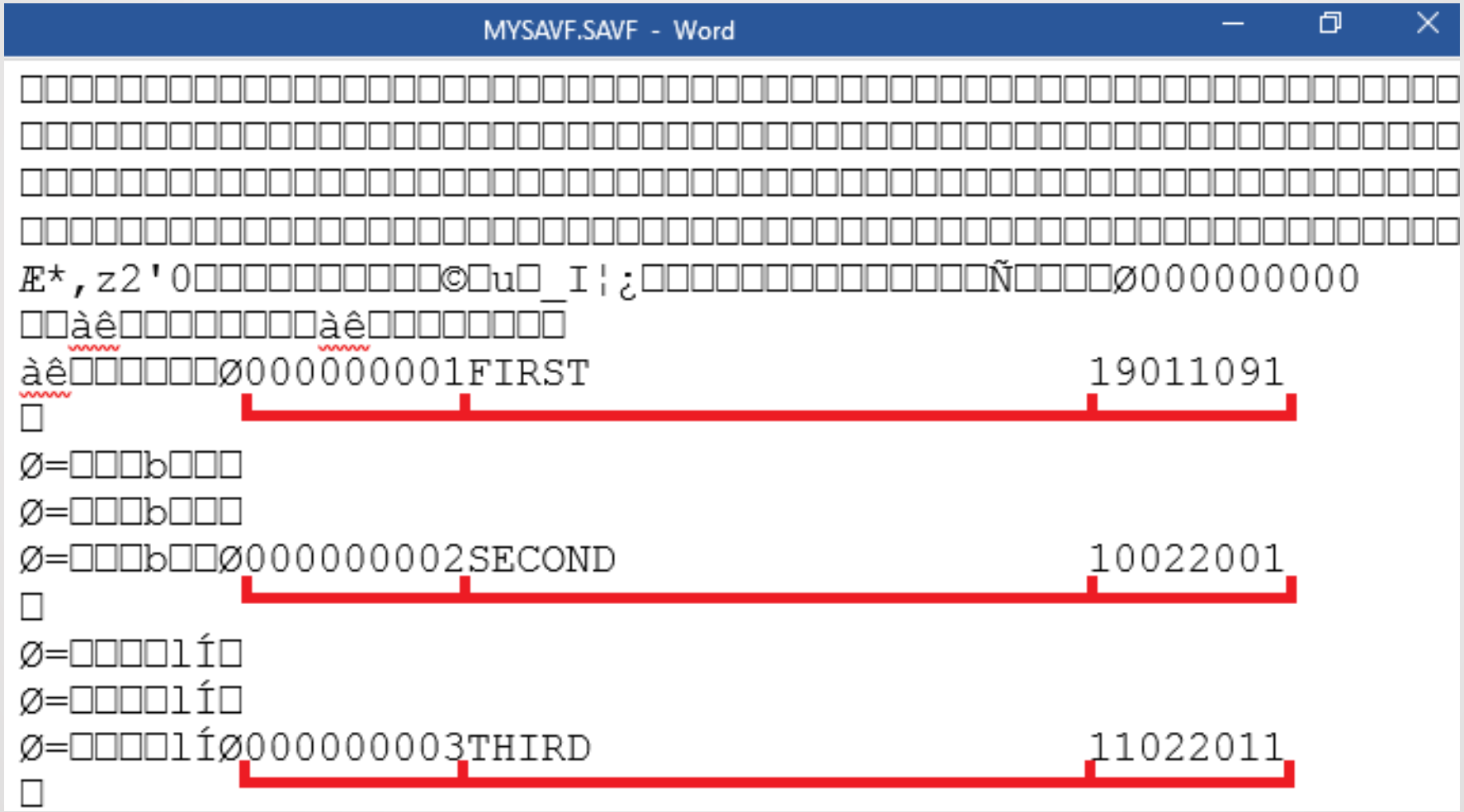
```
-- Associate the FIELDPROC with the column
Alter Table CARDSSQL
Alter Column CARD_NUMBER
Set FieldProc MY_FLDPROC;
```

Encryption with FIELDPROC

Encryption with FIELDPROC in 1½ minute…

Fieldproc video

Encryption with FIELDPROC



Encryption with FIELDPROC

Index will use the ENCODED value!!!

Be sure you understand the impact of encrypting key fields...some operations (i.e. SETLL + READ) may not work as expected.

```
Order By: Number

Card#       Cardholder Name          Card Number
000000002   SECOND                   1020
000000003   THIRD                    1021
000000001   FIRST                    1019
```

Encryption with FIELDPROC

Be sure to R.T.F.M.

Security Guide for IBM i V6.1
IBM Redbook

IBM System i Security: Protecting i5/OS Data with Encryption
IBM Redbook

Encryption enhancements: Field procedure support in DB2
ibm.com/developerworks

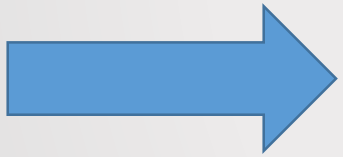
A journey of 1000 miles …

We need encryption…

We can do that!

Agenda

- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority



Adopted Authority

We need object-level authority; our credit card file should not be accessible to the public...at all.

But *some* users still need to have access to the full credit card number …*sometimes*.

Adopted Authority

How can we give authority to a user only when they really need it?

Grant authority to the program instead of the user!

Adopted Authority- Steps

- Change the object owner of the program
- Change the program to run as the owner
-
-
-
-

Adopted Authority

Change the object owner to be the group profile that has the required authority:

```
CHGOBJOWN OBJ(MY_PGM)
          OBJTYPE(*PGM)
          NEWOWN(ACCOUNTING)
```

Adopted Authority

Change the object to run a *OWNER:

```
CHGPGM   PGM(MY_PGM)
         USRPRF(*OWNER)
```

Adopted Authority

Adopted Authority in 3 minutes…

Authority video

Shallow Dive

Be sure to R.T.F.M.

This was a SHALLOW dive. We covered a lot but there was plenty that wasn't covered. Be sure to understand your requirements and your environment, and RTFM!

There are lots of resources available which can help you along the way. There are also lots of vendors who have already RTMF'd and know what they're doing and can set you right.

Summary

- DDS to SQL Conversion
- RCAC (Field Masking)
- FIELDPROC (Encryption)
- Adopted Authority