

# INFO3105 Week 8 Part 1

## Review

- Steps 1, 2 and 3 with AMS (Access Method Services) and DFSort

## Processing Transactions on the KSDS Master File (PCS2PRG1)

Similar to how DBMS systems are built like MS SQL Server, MS Access, Oracle, or IBM DB2 (which we will get to learn/use shortly), data is almost always updated using indexes. With the AIX for our KSDS now loaded we can now process transactions from a separate transaction file to update existing salesperson data, by writing a new COBOL program called **PCS2PRG1 (start by copying either your PCS1PRG1 program, or your LAB2 program if you want a simpler starting point)**. Remember, one of the advantages a KSDS provides over its sequential file counterpart is that records **can be read or updated randomly (ie. Using the Primary Record Key / index)**. Since we have added an AIX to the KSDS, we need to modify our SELECT statement from the case 1 program to accommodate the new KSDS and AIX.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT SALESMASST ASSIGN TO SLSPKS

ORGANIZATION IS INDEXED

ACCESS IS RANDOM

RECORD KEY IS SALESPERSON-NO

ALTERNATE KEY IS SALESPERSON-BRANCH-NO WITH DUPLICATES

FILE STATUS IS WS-IN-STATUS.

SELECT SALESTRANS ASSIGN TO SLSTRANS

FILE STATUS IS WS-TRN-STATUS.

SELECT SALESRPT ASSIGN TO PRNT.

Index / Primary Key

Alt / Foreign Key

Sequential File Processing – Like Case 1, just  
with New / different data

Because our access is RANDOM, we need to change the way we open this file. It is opened as I-O since both input and output operations will occur on it, so in the procedure division your opens would look like this:

PROCEDURE DIVISION.

A000-INITIALIZATION.

OPEN INPUT SALESTRANS  
I-O SALESMASST  
OUTPUT SALESRPT.

Our new transactions will come from a new sequential file, and its file layout should be contained in a copy book (**CCS2SLST**) and belongs right under the FD SALESTRANS file definition:

```
*****
*              COPYBOOK CCS2SLST              *
*              SALES TRANSACTION FILE LAYOUT    *
*              FOR CASE STUDY #2               *
*****
01 SALES-TRANSACTION.
  05 TRANS-NO          PIC 9(5) .
  05 TRANS-TYPE        PIC X.
    88 TRANS-ADD        VALUE 'A' .
    88 TRANS-DEL        VALUE 'D' .
    88 TRANS-CHG        VALUE 'C' .
    88 TRANS-SALE       VALUE 'S' .
    88 TRANS-RET        VALUE 'R' .
  05 TRANS-SALESPERSON-NO PIC 9(5) .
  05 TRANS-DATE        PIC X(6) .
  05 TRANS-DATA        PIC X(34) .
```

← **Note:** TRANS-DATA field – a group of data values / multiple fields !!

Notice the second field of the transaction file is called TRANS-TYPE. This transaction file can contain 5 different types of transactions, as indicated by the 88 values:

1. Add a new salesperson record
2. Delete a salesperson record
3. Change a salesperson
4. Sale – program needs to update ytd #'s
5. Return – program needs to update ytd #'s

Now the layout of the transaction information (ie. The TRANS-DATA field) will be quite a bit different depending on the type of transaction we are dealing with (hint : This means it is important to analyze the data on the file so you understand how it maps to this file/data variable layout) .

The last field/variable in the transaction file is called **TRANS-DATA**. Instead of having 5 types of transaction files we are going to use these same 34 bytes and move them to working storage where they will be **re-mapped** by moving them into different variable names / layouts depending on the specific transaction type record in the file.

Create another copy book member called **CCS2SLSW** and place it in the **WORKING-STORAGE** section that contains:

```

*****
*                               COPYBOOK CCS2SLW
*                               WORKING STORAGE TRANSACTION FILE LAYOUT
*                               FOR CASE STUDY #2
*****
01 WS-TRANS-SALE.
    05 WS-SALES-AMOUNT          PIC 9(5)V99.
    05 WS-DISCOUNT-PCT          PIC V99.
    05 FILLER                    PIC X(25).

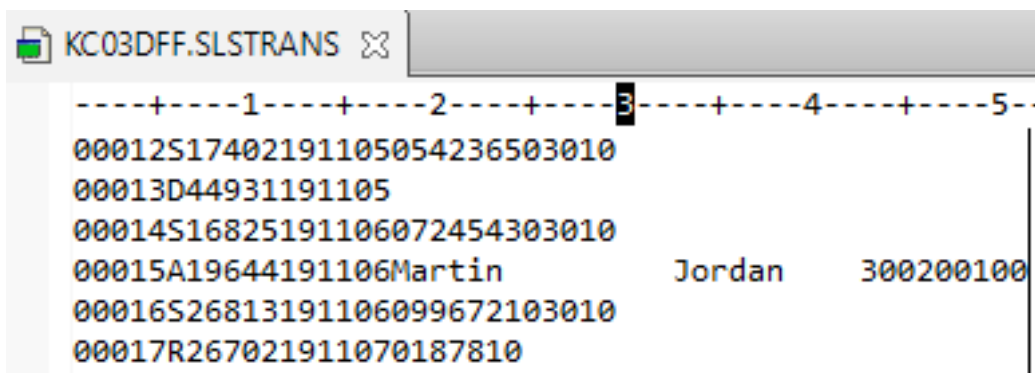
01 WS-TRANS-RETURN.
    05 WS-RETURN-AMOUNT          PIC 9(5)V99.
    05 FILLER                    PIC X(27).

01 WS-TRANS-MAINTENANCE.
    05 WS-TRANS-LAST-NAME        PIC X(15).
    05 WS-TRANS-FIRST-NAME       PIC X(10).
    05 WS-TRANS-BRANCH-NO        PIC 9(3).
    05 WS-TRANS-DEPT-NO          PIC 9(2).
    05 WS-TRANS-COMM-RATE        PIC V9999.

```

We see we have 3 different layouts (all 3 are 34 total bytes long), one of which our transaction will use. If it is an add, delete, or change to the actual salesperson information, we will use the WS-TRANS-MAINTENANCE field (ie. move TRANS-DATA to WS-TRANS-MAINTENANCE, then move the individual fields like WS-TRANS-LAST-NAME, etc. to update the Salesperson-Last-Name, etc. fields/processes based on the specific transaction type).

If it is a sale we use WS-TRANS-SALE and if it is a return we'll use WS-TRANS-RETURN fields in a similar way. Here is a little sample data from the FOL Transaction Data file to give you a clearer picture of how the data is remapped:



```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----
00012S17402191105054236503010
00013D44931191105
00014S16825191106072454303010
00015A19644191106Martin      Jordan      300200100
00016S26813191106099672103010
00017R267021911070187810

```

Typically what happens is that there will be some sort of user screen that adds to the transaction file and then periodically the transactions are processed against the master in a batch (**page 349** of the text demonstrates this). Why not just update the master directly? Well, sometimes if there is low contention or a small amount of records to be changed, that's exactly what happens. In other scenarios, for example payroll processing, the master file would be locked while the update is happening so the response in an interactive scenario would be poor if there were a high number of updates happening concurrently. An even more common scenario involves a file being processed from another system or company. So for case 2 we'll process the transactions as batch job.

## Using the EVALUATE clause

Most programming languages have a statement to test multiple conditions, for instance in C#/C++/Java you can use the SWITCH statement. COBOL has a similar statement called **EVALUATE**. The text discusses the EVALUATE statement on pages **168-173**. You can also look up the syntax of the EVALUATE in the IBM programmers reference manual on page **336**. Here is what a typical EVALUATE looks like:

```
EVALUATE SOME-VARIABLE
  WHEN 'X'
    PERFORM X-RTN
  WHEN 'Y'
    PERFORM Y-RTN
  WHEN 'Z'
    PERFORM Z-RTN
  WHEN OTHER
    PERFORM ERROR-RTN
END-EVALUATE
```

FYI – We'll use the EVALUATE statement to test which type of transaction the program is dealing with as per the TRANS-TYPE (see CCS2SLST above).

Your logic for this program is pretty straightforward:

- process each record in the transaction file sequentially
- determine the transaction type
- then perform the correct routine based on the transaction type

## Processing ADD Transactions

When we discover we have an add transaction, we just want to add a new record to the KSDS and it will also automatically be added to the AIX. The process is relatively straight forward you simply need to move the data (salesperson no., first name, last name, branch-no., department-no. and commission rate) from the transaction file into fields of the salesperson master. Then you need to write the new record and test the file status field after to make sure it is **'00'** or **'02'**.

The '02' indicates that the alternate key has duplicate data in the file, which for us indicates that there is already a salesperson with the same branch-no and this is therefore a valid condition. Any other status number would indicate a problem and the error should be written out. The WRITE statement looks like this for an ADD transaction (remember to move the data to the correct fields **prior** to doing this write):

```
W210-WRITE-MASTER-RECORD.
  WRITE SALESPERSON-MASTER
    INVALID KEY MOVE 'PROBLEM WRITING MASTER STATUS IS:'
      TO WS-ER-PROBLEM.
  DISPLAY 'FILE STATUS FOR ADD = ' , WS-IN-STATUS,
    ' KEY = ' , TRANS-NO.
```

## Lab 10 - 4%

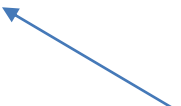
1. Update the JCS2LDRN JCL member from last time to now have a step 4:
  - Step 1 – define KSDS
  - Step 2- execute sort to load KSDS
  - Step 3 – build AIX
  - **Step 4 – execute newly created PCS2PRG1 program – provided in FOL too**
2. Code the PCS2PRG1 program and helper files(copybooks) :
  - Create the 2 additional copybooks (**CCS2SLST** and **CCS2SLSW**), you already have **CCS1SLP** which needs to be copied into **CCS2SLSP** and modified to include a 5 Digit Alternate Key field - SALESPERSON-BRANCH-NO PIC X(5) so that it maps to 52 bytes (vs the original 50 bytes)
    - Tip: Create a hard copy printout of the file layouts/sample data (so you can practice visualizing what the data looks like as your program executes/is built... And/OR Create the data definition lines for these copybooks right in your PCS2PRG1 program first, and once working, then copy that code into/create the copybooks ... this way IDZ can see the fields/help you with syntax errors (you can hover over them to see if they exist & their structure ... ) while coding.
  - Create, edit and load a sequential file to contain the transaction data called SALESTRANS (source is on FOL labelled Transaction data). Get the file record length by either (a: paste into an IDZ member with the #'s across the top, or BEST : b: sum the data lengths in the copybook/file layout)
  - Create a new jcl job stream/member in **KC03XXX.JCL** - with Jobcard, to compile and link the PCS2PRG1 program (copy RIGHT from FOL OR copy from the Case 1 compile and link – ( 1<sup>st</sup> step only) JCL & change all member names from \*cs1\* to \*cs2\* for program and copybook names), this needs to be a **separate job from** the JCS2LDRN jcl and should account for all 3 copy books – ONCE you start using them – you can comment out for now ... or you will see S013 abends & a NO ACTIVE MODULE FOUND error message in the output :

```
COBOL.SYSLIB DD DSN=&SYSUID..COPY(CCS2SLSP),DISP=SHR
COBOL.SYSLIB DD DSN=&SYSUID..COPY(CCS2SLST),DISP=SHR
COBOL.SYSLIB DD DSN=&SYSUID..COPY(CCS2SLSW),DISP=SHR
COBOL.SYSIN DD DSN=&SYSUID..COBOL(PCS2PRG1),DISP=SHR
LKED.SYSLMOD DD DSN=&SYSUID..LOAD(PCS2PRG1),DISP=SHR
```
  - Read the sequential transaction file from start to finish in this new PCS2PRG1 program (start with code you already have in Case 1 for reading and processing a sequential file)
  - Determine the type of transaction
    - Process only the **Add** transactions for this Lab. As proof that the adds worked add the following DISPLAY code to your write routine:

```
W210-WRITE-MASTER-RECORD.
WRITE SALESPERSON-MASTER
INVALID KEY MOVE 'PROBLEM WRITING MASTER STATUS IS:'
TO WS-ER-PROBLEM.
DISPLAY 'FILE STATUS for ADD = ', WS-IN-STATUS,
' KEY = ', SALESPERSON-LAST-NAME, TRANS-NO
```
    - You do not need to generate a format report yet.

3. Update the JCS2LDRN jcl to now execute a 4<sup>th</sup> step that executes our new transaction processing program PCS2PRG1 – this JCL also included for you in FOL. Note now you will need to Submit 2 Jobs every time you change your program – 1 Job to Compile your code & this JCS2LDRN to execute your compiled/executable code in KC03xxx.LOAD(PCS2PRG1).
  - **Make sure you include the new path for the AIX**, even though we are not using it yet, it still needs to be present in the JCL or you will get a missing DD error:

```
// *****  
// * JCL TO EXECUTE COBOL PROGRAM MODULE  
// *****  
//STEP4 EXEC PGM=PCS2PRG1  
//STEPLIB DD DSN=&SYSUID..LOAD,DISP=SHR  
//PRNT DD SYSOUT=*  
//SLSPKS DD DSN=KC0307F.SLSPKSDS,DISP=SHR  
//SLSPKS1 DD DSN=KC0307F.SLSP.BRN.PATH,DISP=SHR  
//SLSTRANS DD DSN=KC0307F.SLSTRANS,DISP=SHR  
/*
```



AIX path reference

4. Test your results by checking that your new KC03xxx.SLSPKSDS file has the new records properly added with all values initialized as you would expect, using the host File Manager F option as you have done in the past. Note if you rerun the JCSLDRN job to Delete/define this file while you are viewing this file in the FM editor, you will get a JCL error – file in use!  
**Submit to the FOL dropbox a screen capture of your KC03xxx.SLSPKSDS** file showing at least 1 of the ADD's
5. **Submit to the FOL submission dropbox the most recent compile of your PCS2PRG1 – Note no need to worry much about coding style at this point – this is just to ensure I can give partial marks if there are compile/etc. issues with your code .**
6. **Submit to the FOL submission dropbox** a text file (or Word doc - no ZIP files please/thanks) with the entire JES Spool results for the JCL execution of **JCS2LDRN** and the “DISPLAY”s will show up there, partial sample display output should look like:

```
FILE STATUS FOR ADD = 02 KEY = 00015  
FILE STATUS FOR ADD = 02 KEY = 00025
```